



¿QUIÉN DIJO QUE **REACT** NO TIENE UNA **SUPER RUTA** PARA **APRENDER** TODOS SUS **SECRETOS**?



Una pequeña aventura
con mis alpacas por
React moderno,
TypeScript, estado y
backend.

Desliza para
explorar.





🌌 ¿Qué es realmente React?



React es **declarativo**. Le dices **qué quieres ver**, no cómo dibujarlo paso a paso.



Todo se **construye** con **componentes** (como piezas de LEGO reutilizables).



Componente



Composición



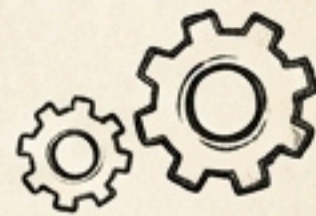
Interfaz

La magia del Virtual DOM:
Estado cambia ⚡ → React compara 🔍
Actualiza **SOLO** lo necesario del DOM real 🔄.

¡Ah... así que React
no era magia!



🚫 ¿Qué pasó con Create React App?



Aquí tenemos un **viejo conocido...** que ya se jubiló.



LETRERO 1: Vite ⚡

Para aplicaciones
cliente (SPA).
Ultrarrápido, ligero y sin
configuración compleja.

LETRERO 2: Next.js 🌐

Para renderizado en
servidor (SSR).
Necesidades avanzadas de
SEO y arquitectura full-stack.

¡Mi favorito para
empezar rápido!



🧠 Bienvenida al nuevo modelo mental (React 19)

Antes pasábamos horas **optimizando a mano** para que React no **re-renderizara** cosas de más.

~~useMemo | useCallback | React.memo~~



¡Menos código boilerplate, más tiempo para café! ☕

¡Entra el React Compiler! ✨

En el enfoque moderno, el compilador automatiza gran parte de este trabajo pesado.

Ojo, no significa que jamás debas usarlos, pero ya no son tu preocupación número uno.



##

“¿USEFFECT PARA TODO? ¡ALPACA, NO!”

El Gran Aprendizaje: Si un valor puede calcularse a partir de props o estado existente... **¡hazlo durante el render!**



✗ MAL (Renderizados innecesarios):

```
const [nombreCompleto, setNombreCompleto] = useState('');  
useEffect(() => {  
  setNombreCompleto(nombre + ' ' + apellido);  
}, [nombre, apellido]);
```

✓ BIEN (Sencillo y limpio):

```
const nombreCompleto = nombre + ' ' + apellido;
```

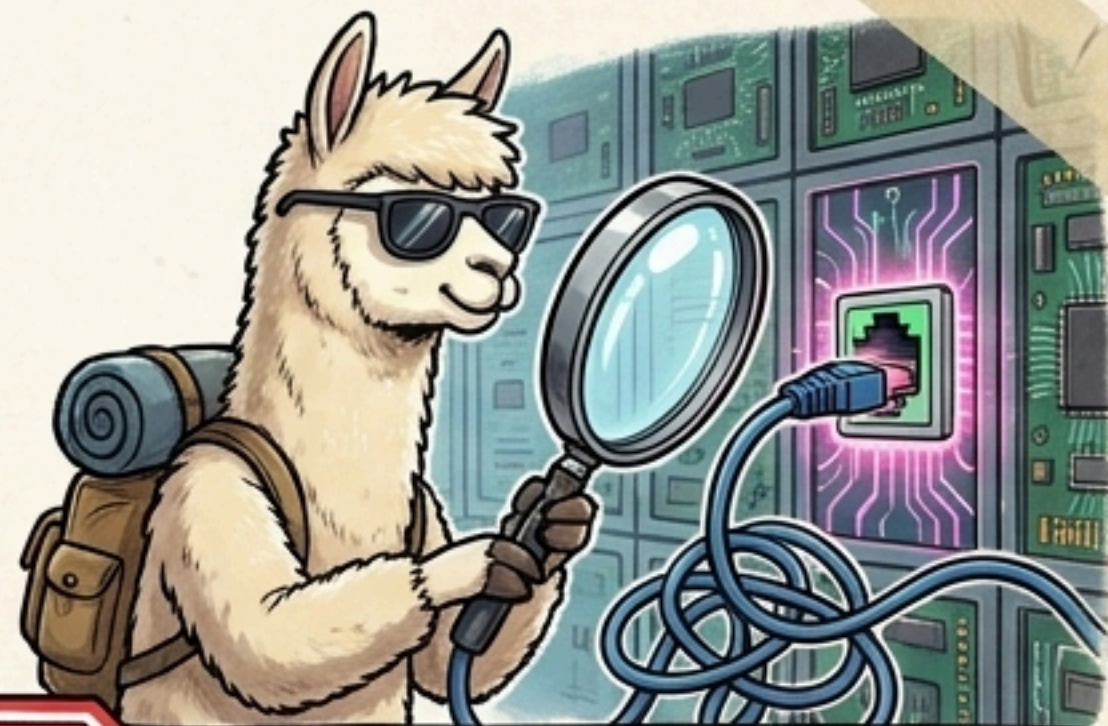
Esta alpaca acaba de descubrir que no todo necesita un efecto.



🧩 Ok... entonces, ¿cuándo SÍ uso useEffect?

- Está pensado para **sincronizar React con sistemas externos:**

🌐 WebSockets | 💾 localStorage |
📡 Librerías externas | ⌚ Suscripciones.

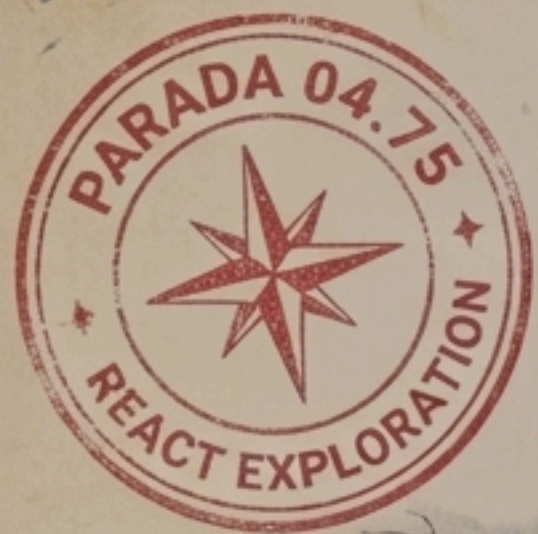


No olvides **limpiar** tus consultas (*cleanup*) e **ignorar** respuestas antiguas para evitar condiciones de carrera.



🔑 Hay un truco que parece pequeño, pero cambia mucho:

¿Necesitas reiniciar el estado interno de un componente al **cambiar de entidad** (ej. de usuario)? No uses un `useEffect`. Simplemente cambia la prop `key` del componente. **React** destruirá el anterior y creará uno nuevo limpio. ✨



✨ Kit de Herramientas Secretas (React 19)



use()

Desenvolver promesas y contextos directamente en el renderizado. *Adiós al infierno de los wrappers.*



useOptimistic()

Para mostrar una respuesta inmediata en la interfaz mientras tu operación asíncrona (lenta) termina en el fondo.



useActionState()

El fin de sufrir con los formularios. Maneja el estado, errores y carga asociados a *Server Actions* de forma nativa.



🎯 ¿Dónde debería vivir cada estado? (La Mochila)

GLOBAL DEL CLIENTE

- 👉 Zustand (❤️ favorito, simple y directo).
- 👉 Redux Toolkit (si estás en arquitecturas empresariales estrictas).

Respira. No uses Context API para datos que cambian cada milisegundo.



ESTADO LOCAL

- 👉 useState y, si la lógica se complica, useReducer.

ESTADO DERIVADO

- 👉 ¡Cáculalo al vuelo durante el render! (Sin estado extra).

ESTADO DEL SERVIDOR

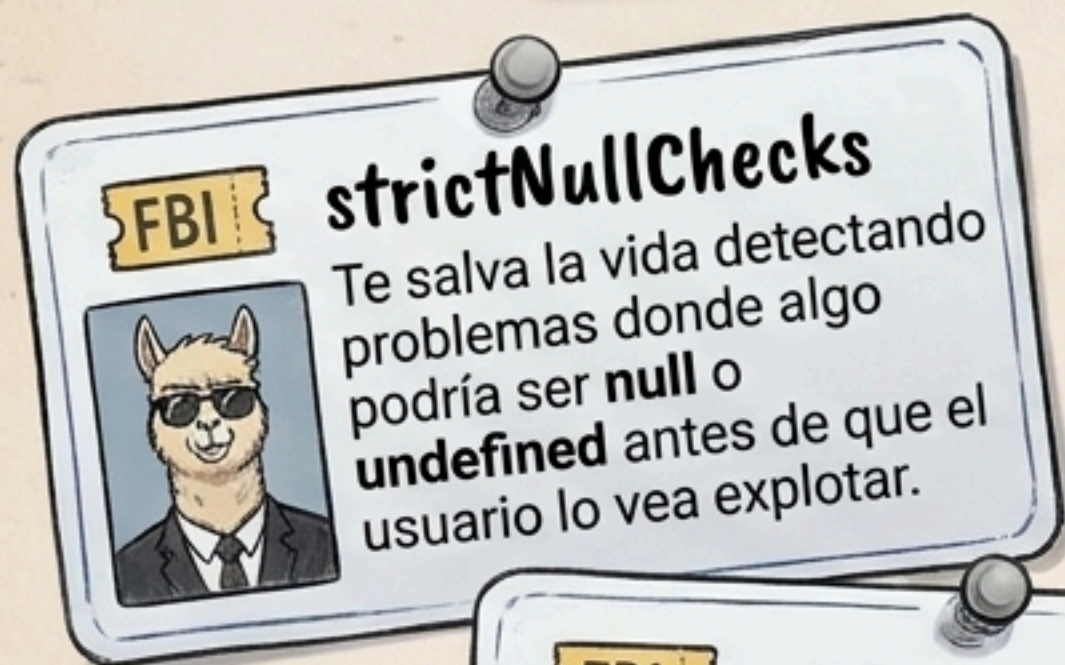
- 👉 TanStack Query. Caché, sincronización de fondo y deduplicación automáticas.





🛡️ La Alpaca Guardaespaldas: TypeScript

React moderno necesita buenos límites.
Configura siempre tu proyecto con:
`"strict": true`



TypeError:
Cannot read
properties of
undefined



any entró a la fiesta... y TypeScript pidió identificación.

Abusar de any destruye la magia de TypeScript.
En lugar de rendirte, usa tu caja de herramientas utilitaria:



 **Record<string, unknown>:**
Cuando no sabes qué trae el objeto, pero sabes que es un objeto.

No me conviertas en cualquier cosa



 **Partial:**
Hace que todas las propiedades sean opcionales.

 **Required:**
Obliga a que todas estén presentes.

 **Pick:**
Elige solo las propiedades que necesitas.

 **Omit:**
Quita solo las que te estorban.



🚨 Pasamos al Backend (Node.js + Express)

En el backend, un error mal manejado tumba toda la aplicación. Tu escudo principal es el **Middleware Global de Errores**.



```
app.use((err, req, res, next) => {
  console.error("¡Ups!", err.stack);
  res.status(500).send('Algo salió mal en el reino backend.');
```

⚠️ **¡ATENCIÓN!** Este middleware de (err, req, res, next) 4 parámetros **DEBE** estar siempre al **FINAL** de toda la pila de rutas y middlewares para atraparlos todo.





✨ Los Asesinos Silenciosos y el Aterrizaje Perfecto

Los Asesinos Silenciosos:
`uncaughtException` y
`unhandledRejection`.



- ✓ 1. Registrar el error crítico en los logs.
- ✓ 2. Dejar de aceptar nuevas peticiones HTTP.
- ✓ 3. Terminar conexiones a la base de datos limpiamente.
- ✓ 4. Salir con `process.exit(1)`.
- ✓ 5. Dejar que tu aliado externo (PM2 o Docker) reinicie la app por ti.

LA RUTA ALPACA PARA DOMINAR REACT



✨ React no es una montaña de APIs.
Es una forma diferente de pensar.



La mejor ruta para aprender React no es memorizarlo todo.
Es entender por qué cada herramienta existe
y cuándo realmente necesitas usarla.

