

React Nivel Avanzado: Context API y Global State

En este nivel, aprenderás a manejar estados globales en React usando **Context API**. Context es una herramienta poderosa que permite compartir datos entre componentes sin necesidad de pasar props manualmente a cada uno. Aprenderás cómo implementarlo, cuándo usarlo, y lo combinaremos con hooks como **useReducer** para centralizar la lógica de estado.

Temas que Cubriremos

1. ¿Qué es Context API y cuándo usarlo?
 2. Creación de Contexts, Providers y Consumers.
 3. Uso de **useContext** para acceder al estado global.
 4. Combinación de Context API con **useReducer**.
 5. Ejemplo práctico: Carrito de compras con Context API y Reducer.
-

1. ¿Qué es Context API y Cuándo Usarlo?

Context API es una solución nativa de React para manejar el estado global de una aplicación, evitando el **prop drilling** (tener que pasar props manualmente a través de múltiples niveles de componentes). Es ideal cuando:

- Necesitas compartir datos o funciones entre varios componentes en diferentes niveles del árbol de componentes.
- Quieres evitar un manejo excesivo de props en componentes intermedios.
- Tienes estados que son esenciales para varias partes de la aplicación (ej. autenticación, temas, carrito de compras).

La estructura básica de Context es:

1. Creación del Contexto:

```
const MiContexto = React.createContext();
```

2. Proveer datos globales:

```
<MiContexto.Provider value={datos}>
  {children}
</MiContexto.Provider>
```

3. Consumir los datos con **useContext**:

```
const datos = React.useContext(MiContexto);
```

2. Creación de Contexts, Providers y Consumers 🌐

Estructura de Context:

Un contexto se compone principalmente de un **Provider**, que proporciona el estado, y un **Consumer**, que lo consume. En React moderno, utilizamos `useContext` para simplificar el consumo.

Ejemplo:

```
import React, { createContext, useState } from "react";

const TemaContext = createContext();

function TemaProvider({ children }) {
  const [temaOscuro, setTemaOscuro] = useState(false);

  const cambiarTema = () => setTemaOscuro(!temaOscuro);

  return (
    <TemaContext.Provider value={{ temaOscuro, cambiarTema }}>
      {children}
    </TemaContext.Provider>
  );
}

export { TemaContext, TemaProvider };
```

Consumiendo el Contexto:

```
import React, { useContext } from "react";
import { TemaContext } from "../TemaContext";

function BotonTema() {
  const { temaOscuro, cambiarTema } = useContext(TemaContext);

  return (
    <button onClick={cambiarTema}>
      Cambiar a {temaOscuro ? "Claro" : "Oscuro"}
    </button>
  );
}
```

3. Uso de `useContext` para Acceder al Estado Global 🌐

El hook `useContext` simplifica el acceso al contexto en cualquier componente. Es útil cuando necesitas consumir datos o ejecutar funciones proporcionadas por el **Provider**.

```
const datos = useContext(MiContexto);
```

Esto evita el uso del componente `Consumer`, lo que hace que el código sea más legible y fácil de manejar.

4. Combinación de Context API con `useReducer`

Combinar `useReducer` con Context API es una técnica avanzada para manejar estados globales con lógica centralizada. Es ideal para aplicaciones con múltiples acciones, como un carrito de compras o una lista de tareas.

Ejemplo: Reducer para un Carrito de Compras

4. Definir el Reducer y el Contexto:

```
import React, { createContext, useReducer } from "react";

const CarritoContext = createContext();

const reducer = (state, action) => {
  switch (action.type) {
    case "AGREGAR_PRODUCTO":
      return [...state, action.payload];
    case "ELIMINAR_PRODUCTO":
      return state.filter((producto) => producto.id !== action.payload);
    default:
      return state;
  }
};

const CarritoProvider = ({ children }) => {
  const [carrito, dispatch] = useReducer(reducer, []);

  return (
    <CarritoContext.Provider value={{ carrito, dispatch }}>
      {children}
    </CarritoContext.Provider>
  );
};

export { CarritoContext, CarritoProvider };
```

5. Consumir el Carrito en Componentes:

```
import React, { useContext } from "react";
import { CarritoContext } from "../CarritoContext";

function ListaProductos() {
  const { carrito, dispatch } = useContext(CarritoContext);

  const agregarProducto = () => {
    const producto = { id: 1, nombre: "Producto 1" };
    dispatch({ type: "AGREGAR_PRODUCTO", payload: producto });
  };

  return (
    <div>
      <button onClick={agregarProducto}>Agregar Producto</button>
      <ul>
        {carrito.map((producto) => (
          <li key={producto.id}>

```

```
        {producto.nombre}{" "}
        <button
          onClick={() =>
            dispatch({ type: "ELIMINAR_PRODUCTO", payload: producto.id })
          }
        >
          Eliminar
        </button>
      </li>
    ))}
  </ul>
</div>
);
}
```

5. Ejemplo Práctico: Carrito de Compras

En este ejemplo práctico, usamos Context API para manejar el estado global de un carrito, y combinamos con **useReducer** para manejar las acciones de agregar y eliminar productos. Este enfoque es ideal para aplicaciones que necesitan una lógica de estado bien organizada.

Resumen

En este nivel, aprendiste:

6. Cómo usar Context API para compartir datos globalmente.
7. Cómo usar **useContext** para simplificar el acceso al contexto.
8. Combinar Context API con **useReducer** para manejar estados complejos.
9. Aplicar estos conceptos en un caso práctico como un carrito de compras.

Estos conocimientos son fundamentales para construir aplicaciones React más grandes y escalables. ¡Sigue practicando y construyendo proyectos para dominar esta metodología! 