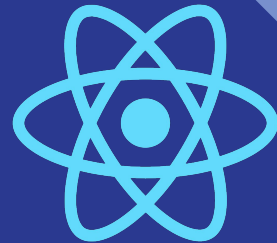


React CONTEXT


by FORMAR

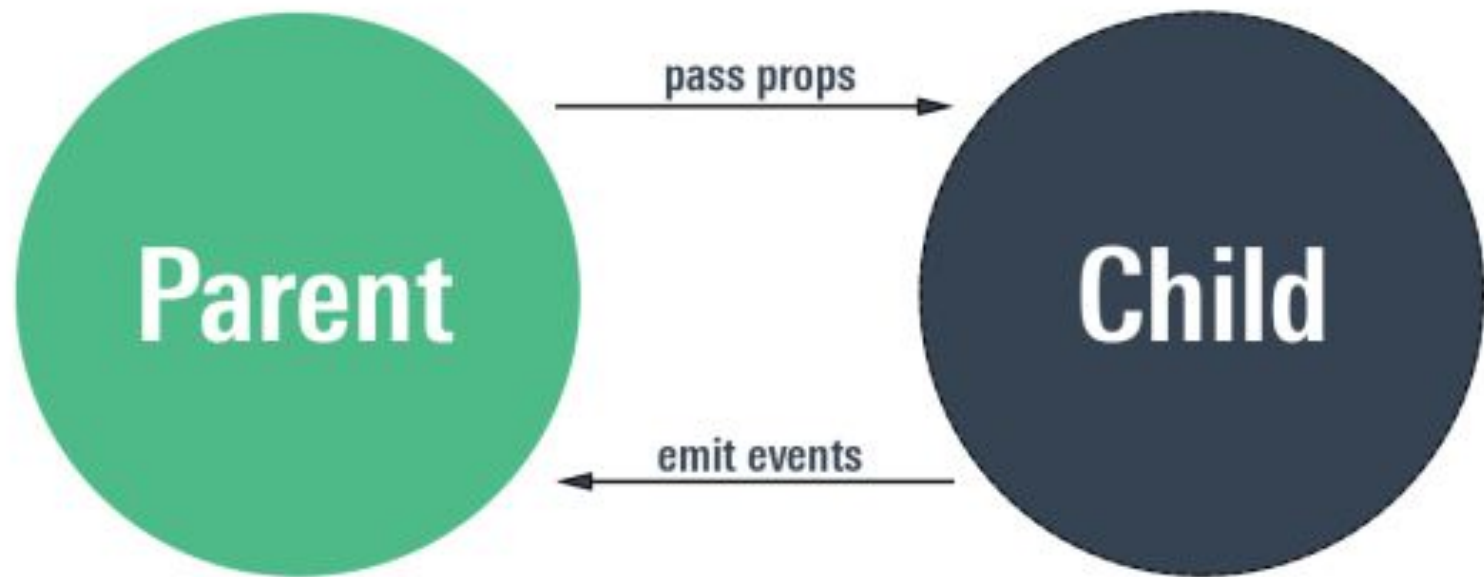


El ***state*** pasado por *props*

Manejar el ***state*** es una parte esencial del desarrollo de aplicaciones en React.

Una forma común de manejar el estado es pasar *props*. Pasar *props* es enviar datos de un componente a otro.






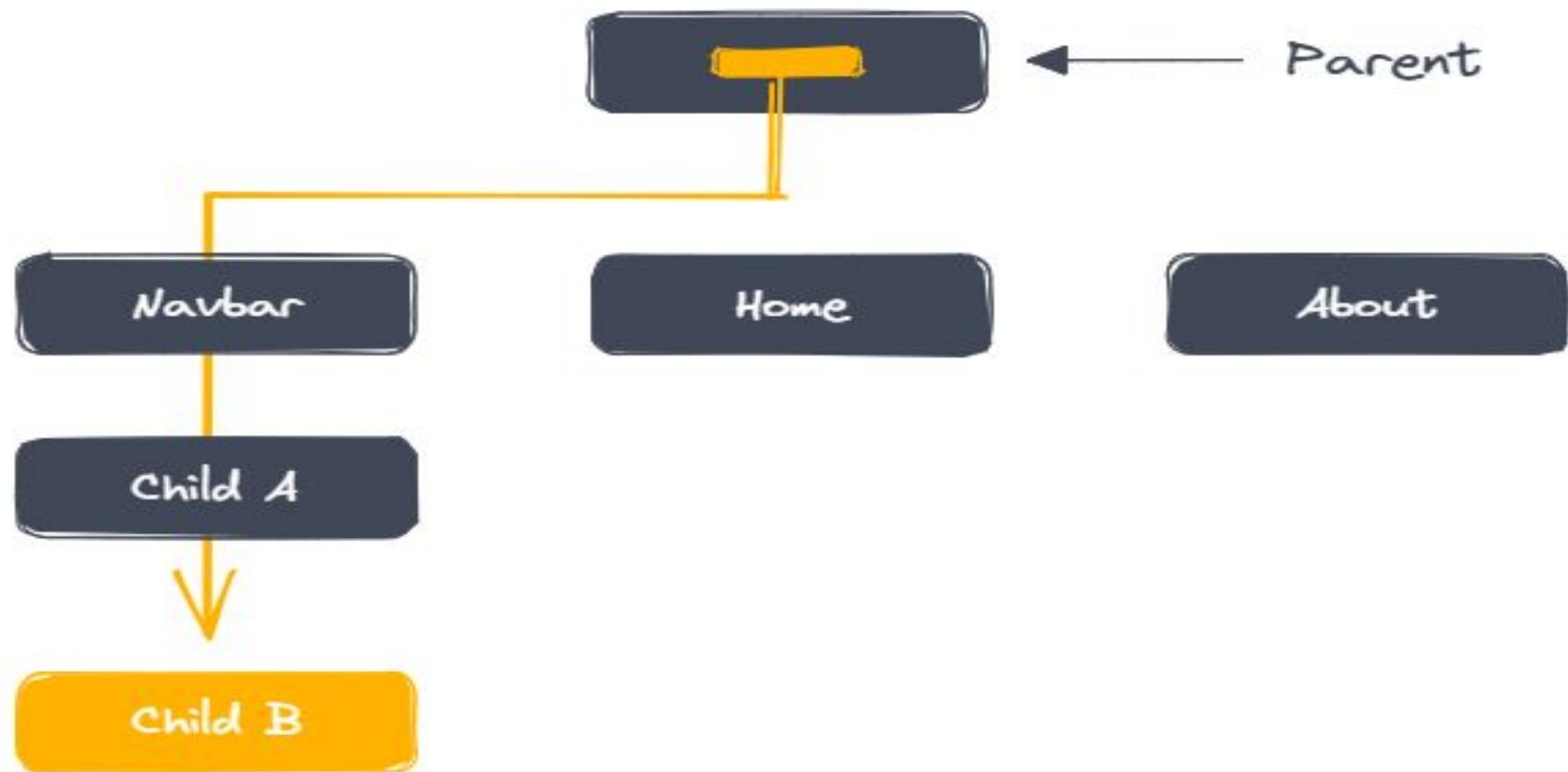
El problema de pasar *props*

Pero puede ser molesto pasar *props* cuando:

1. Se tiene que enviar los mismos datos a muchos componentes, o
2. Los componentes están muy lejos unos de otros.

Esto puede hacer que una aplicación sea más lenta y más difícil de trabajar.

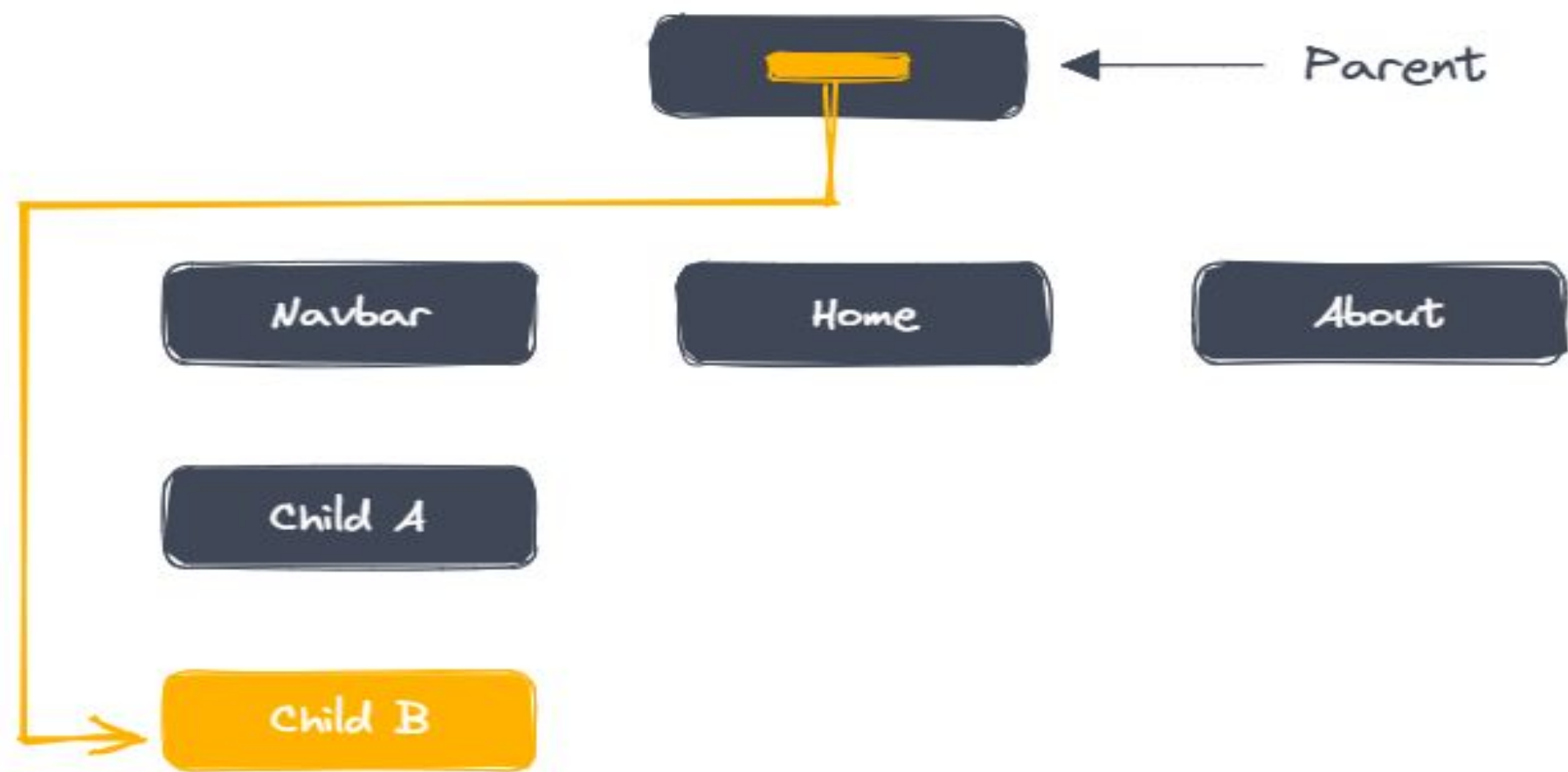




La solución: **CONTEXT**

Afortunadamente, React proporciona una función integrada conocida como ***CONTEXT*** que ayuda a "teletransportar" datos a los componentes que los necesitan sin pasar *props*.

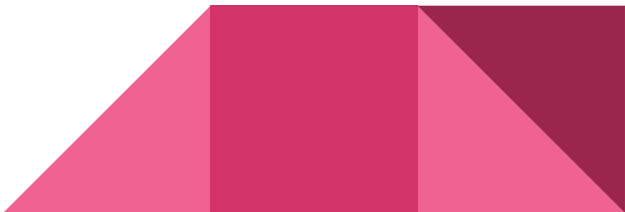


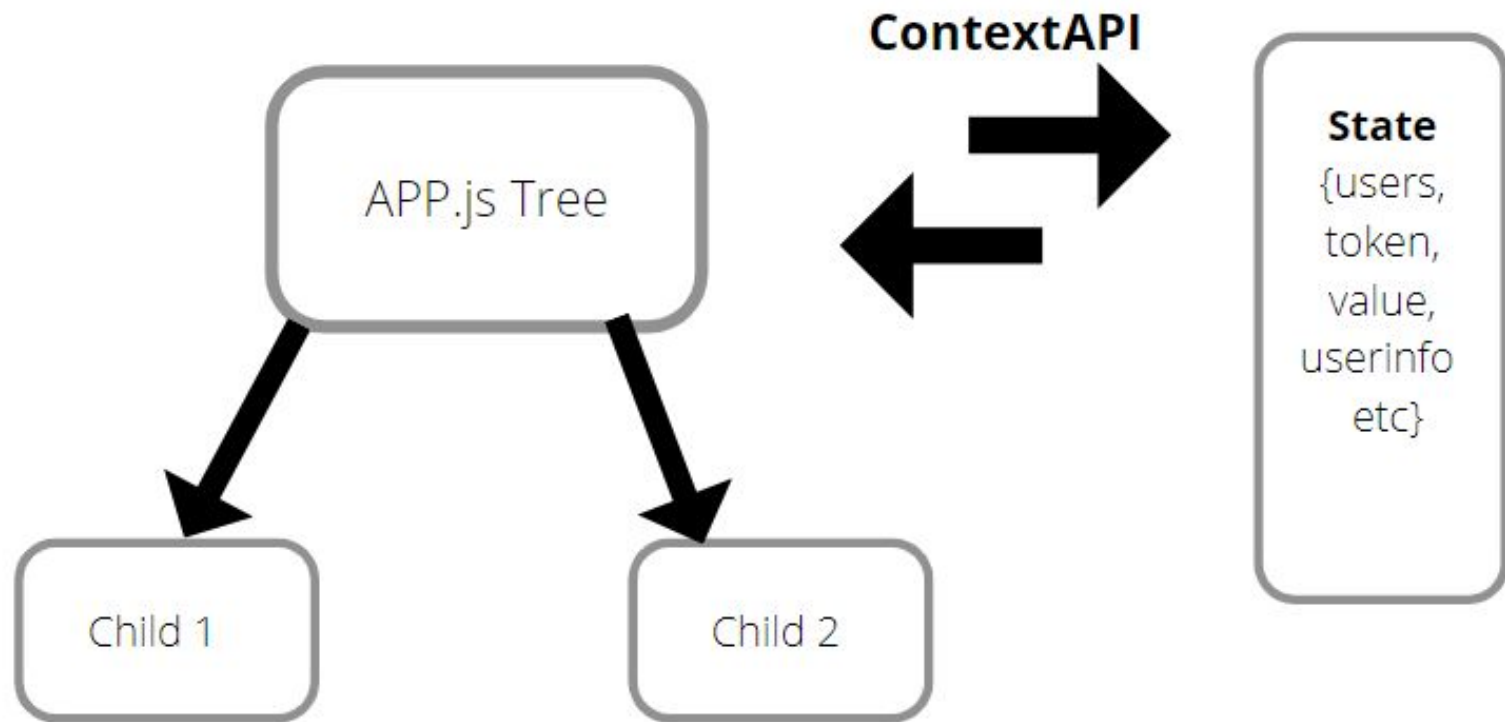


¿Cómo funciona **CONTEXT**?

Con CONTEXT puede crear un "contexto" que contenga la información que se necesite disponer.

Luego, puede usar ese contexto a todos los componentes, sin tener que pasar la información a través de *props*



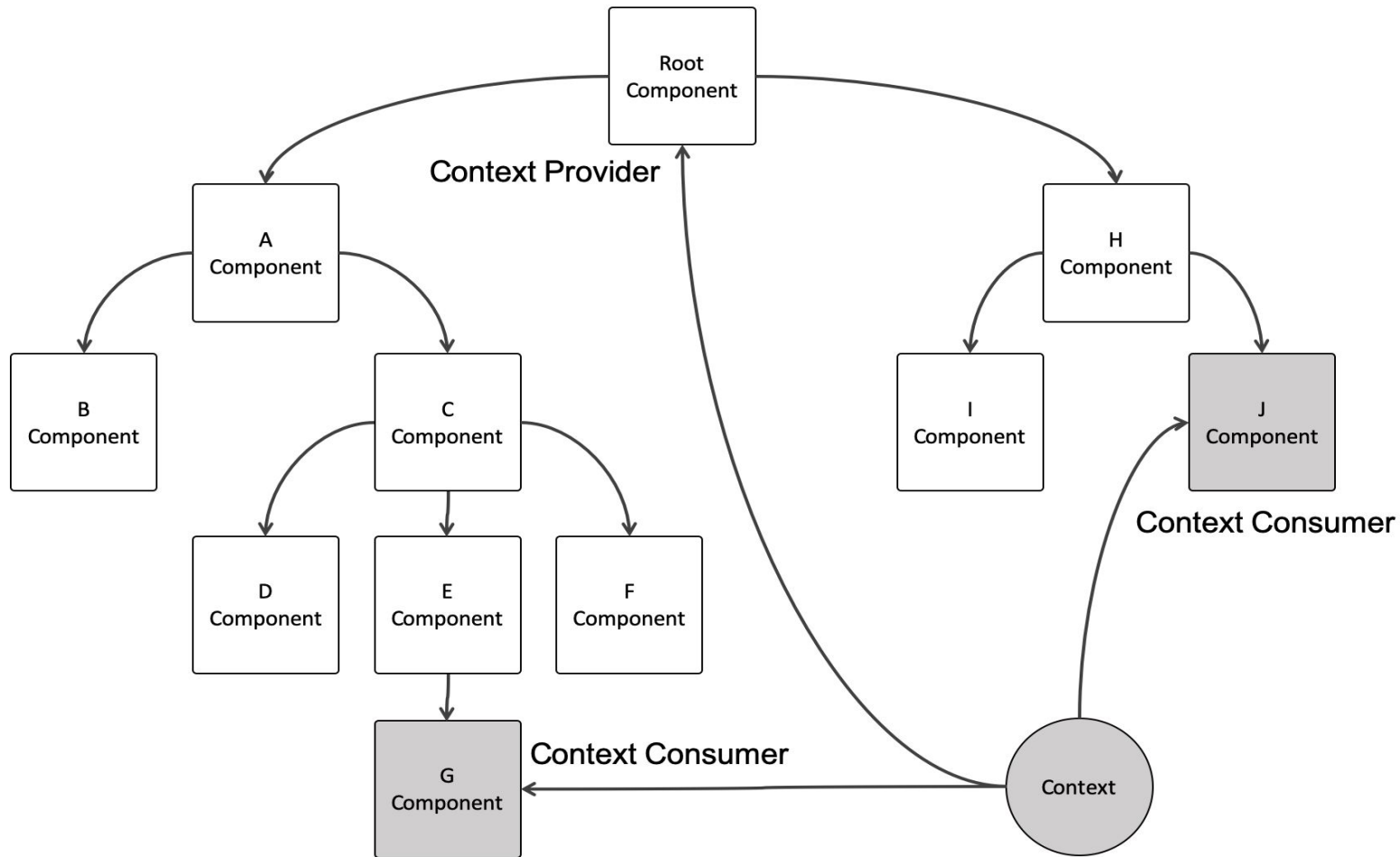


¿Cómo funciona **CONTEXT**?

CONTEXT consta de dos partes principales: el *proveedor (Provider)*, y el *consumidor (Consumer)*.

- El *proveedor* crea y administra el contexto.
- El *consumidor* se utiliza para acceder al contexto





Implementación


Crear un Context Object

Primero, necesita crear un objeto de contexto usando la **createContext()**. Este objeto de CONTEXT contendrá los datos que desea compartir en su aplicación.

```
import { createContext } from 'react';  
  
export const MyContext = createContext( "" );
```

Envuelva componentes con un **Provider**

Una vez que haya creado un objeto de contexto, debe envolver los componentes que necesitan acceso a los datos compartidos con un componente **Provider**. Este acepta una propiedad "**value**" que contiene los datos compartidos, al que acceden todos los componentes hijos del **Provider**.



```
import { useState, React } from "react";
import { MyContext } from "../MyContext";
import MyComponent from "../MyComponent";

function App() {
  const [text, setText] = useState("");
  return (
    <div>
      <MyContext.Provider value={{ text, setText }}>
        <MyComponent />
      </MyContext.Provider>
    </div>
  );
}

export default App;
```

Consuma el **Context**

Ahora que hemos creado el componente *Provider*, necesitamos consumir el *Context* en otros componentes. Para hacer esto, usamos el hook **useContext** de React.





```
import { useContext } from 'react';
import { MyContext } from './MyContext';

function MyComponent() {
  const { text, setText } = useContext(MyContext);
  return (
    <div>
      <h1>{text}</h1>
      <button onClick={() => setText('Hello, world!')}>
        Click me
      </button>
    </div>
  );
}
export default MyComponent;
```

Casos de uso


Caso de uso: Tema

Puede usar *Context* para almacenar el tema actual de su aplicación y ponerlo a disposición de todos los componentes. De esta manera, siempre que el usuario cambie el tema (como al habilitar el modo oscuro), todos los componentes se actualizarán.



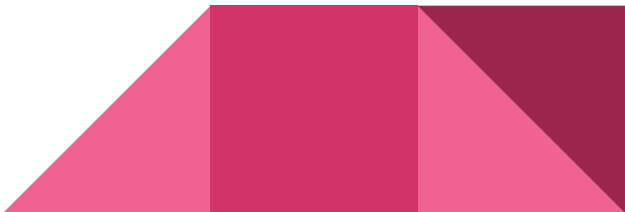
Caso de uso: Autenticación de usuario

También puede utilizar *Context* para almacenar el estado de autenticación de un usuario y transmitirlo a todos los componentes que lo necesiten. De esta manera, puede restringir fácilmente el acceso a ciertas partes de su aplicación según el estado de autenticación.




Caso de uso: Soporte multilingüe

Puede almacenar el idioma actual de su aplicación en el contexto y transmitirlo a todos los componentes que lo necesiten. De esta manera, puede cambiar fácilmente entre diferentes idiomas sin tener que pasar el idioma como apoyo a todos los componentes.



Caso de uso: Acceder a datos externos

Finalmente, puede usar la API de contexto para almacenar datos recuperados de fuentes externas, como API o bases de datos, y ponerlos a disposición de todos los componentes. Esto puede simplificar su código y facilitar la administración de datos en su aplicación.





Buenas prácticas

Definir su contexto en un archivo aparte

Es una buena práctica definir su objeto de contexto en un archivo separado para mantener su código organizado y fácil de mantener.




Utilice solo para el manejo de estado global

Evite usarlo para el estado al que solo se necesita acceder dentro de un solo componente, ya que puede generar problemas de rendimiento y complejidad innecesarios.



Use Providers con moderación

Si bien los *providers* pueden ser una herramienta poderosa para administrar el estado global, generalmente es una buena idea usarlos con moderación. En su lugar, considere usar props para pasar datos a través de su árbol de componentes siempre que sea posible.



Use valores predeterminados

Al crear un nuevo contexto, es una buena idea proporcionar un valor predeterminado que se usará si no hay un proveedor presente. Esto puede ayudar a prevenir errores inesperados y hacer que su código sea más sólido.

