Ya has cubierto componentes, estado local (useState) y efectos secundarios/peticiones API (useEffect).

El siguiente gran pilar de React que debes dominar es la **gestión de estado global**.

¿Qué pasa cuando un componente (como el Header) necesita saber algo que está en otro componente "hermano" (como el UserDirectory)? Hasta ahora, la única forma sería "levantar el estado" al componente padre (App.js) y pasarlo hacia abajo con props. Pero si la aplicación crece, terminarás pasando props a través de 5 o 10 niveles de componentes. Esto se llama **"prop drilling"** y es muy difícil de mantener.

La solución moderna en React para esto es la **Context API**.

En este tutorial, crearemos un interruptor de **Modo Claro / Modo Oscuro** que afectará a toda la aplicación, sin tener que pasar la prop theme a cada componente individualmente.

Tutorial: Gestión de Estado Global con Context API (useContext)

Objetivo del Proyecto:

Implementar un "proveedor de contexto" (Context Provider) que almacene el tema actual (claro u oscuro) y una función para cambiarlo. Luego, cualquier componente en la aplicación, sin importar cuán profundo esté, podrá "consumir" este contexto para adaptar su estilo y activar el cambio de tema.

Paso 1: Crear el Contexto del Tema

El "Contexto" es como un canal de radiodifusión global para tus datos. Primero, necesitamos crear ese canal.

1. Dentro de tu carpeta src, crea una nueva carpeta llamada context.
2. Dentro de src/context, crea un nuevo archivo llamado ThemeContext.js.
3. Añade el siguiente código:

JavaScript

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


// 1. Creamos el Contexto
// Le damos un valor inicial (opcional) que puede ser usado por los consumidores
// si no están envueltos en un Provider.
const ThemeContext = createContext({
```

```
  theme: 'light',
  toggleTheme: () => {},
});

// 2. Creamos el Componente "Proveedor"
// Este componente envolverá nuestra aplicación y proveerá el estado real.
export const ThemeProvider = ({ children }) => {
  const [theme, setTheme] = useState('light'); // 'light' o 'dark'

  // Función para cambiar el tema
  const toggleTheme = () => {
    setTheme(currentTheme => (currentTheme === 'light' ? 'dark' : 'light'));
  };

  // 3. Pasamos el estado actual y la función para cambiarlo
  // a todos los componentes hijos a través del 'value' prop.
  return (
    <ThemeContext.Provider value={{ theme, toggleTheme }}>
      {children}
    </ThemeContext.Provider>
  );
};

export default ThemeContext;
```

Desglose:

- **createContext()**: Crea el objeto de contexto (ThemeContext).
- **ThemeProvider**: Este es un componente personalizado que hemos creado.
- **useState**: Usamos el estado *dentro* del proveedor para almacenar el valor actual del tema.

- **ThemeContext.Provider**: Es el componente mágico. Cualquier componente hijo dentro de él podrá acceder al objeto value.
- **{children}**: Es una prop especial que representa a todos los componentes hijos que ThemeProvider envolverá.

Paso 2: Envolver la Aplicación con el Proveedor

Para que toda nuestra aplicación pueda "escuchar" el contexto del tema, debemos envolverla con el ThemeProvider que acabamos de crear. El mejor lugar para hacer esto es en src/index.js.

1. Abre src/index.js.
2. Importa tu ThemeProvider y envuelve el componente <App />:

JavaScript

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import { ThemeProvider } from './context/ThemeContext'; // <-- Importamos

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <ThemeProvider> {/* <-- Envolvemos nuestra App */}
      <App />
    </ThemeProvider>
  </React.StrictMode>
);
```

¡Listo! Ahora cada componente dentro de App tiene acceso al contexto del tema.

Paso 3: Consumir el Contexto con el Hook useContext

Ahora, vamos a "sintonizar" nuestro canal de contexto en cualquier componente que lo necesite.

Primero, creemos un componente simple para el interruptor.

1. Crea una nueva carpeta src/components/ThemeSwitcher.
2. Dentro, crea ThemeSwitcher.js:

JavaScript

```
import React, { useContext } from 'react';
import ThemeContext from '../../context/ThemeContext'; // Importamos el contexto

const ThemeSwitcher = () => {
  // 3. Usamos el hook useContext para consumir el contexto
  const { theme, toggleTheme } = useContext(ThemeContext);

  return (
    <button onClick={toggleTheme} className="theme-switcher">
      Cambiar a Modo {theme === 'light' ? 'Oscuro' : 'Claro'}
    </button>
  );
};

export default ThemeSwitcher;
```

¡Observa la Magia!

- **import { useContext } from 'react';**: Importamos el hook useContext.
- **const { theme, toggleTheme } = useContext(ThemeContext);**: Esta línea le dice a React: "Oye, conéctame al ThemeContext más cercano y dame el value que está proveyendo".
- No hay props. No hay "prop drilling". Simplemente funciona.

Paso 4: Aplicar el Tema a la Aplicación

Vamos a hacer que nuestros componentes existentes reaccionen al cambio de tema.

1. **Modifica src/App.js:**

Vamos a añadir el ThemeSwitcher y hacer que el fondo de la aplicación cambie.

JavaScript

```
import React, { useContext } from 'react';
import './App.css';
import Header from './components/Header/Header';
import UserDirectory from './components/UserDirectory/UserDirectory';
import ThemeSwitcher from './components/ThemeSwitcher/ThemeSwitcher'; // Importamos el interruptor
import ThemeContext from './context/ThemeContext'; // Importamos el contexto

function App() {
  const { theme } = useContext(ThemeContext); // Consumimos el contexto

  // Añadimos una clase 'dark' al div principal si el tema es oscuro
  return (
    <div className={`App ${theme}`}>
      <Header />
      <ThemeSwitcher /> {/* Añadimos el interruptor */}
      <main>
        <UserDirectory />
      </main>
    </div>
  );
}

export default App;
```

2. **Modifica src/App.css (o tu CSS global):**

Ahora, añadamos los estilos para el modo oscuro.

CSS

```
/* ... tus estilos existentes ... */
```

```
.App {
  text-align: center;
  background-color: #ffffff; /* Modo Claro por defecto */
  color: #282c34;
  min-height: 100vh;
  transition: background-color 0.3s ease, color 0.3s ease;
}

/* Estilos del Modo Oscuro */
.App.dark {
  background-color: #282c34;
  color: #ffffff;
}

/* Puedes hacer lo mismo para otros componentes */
.App.dark .app-header {
  background-color: #20232a;
}

.App.dark .user-card {
  background-color: #3a3f4b;
  border-color: #555;
}

.App.dark .user-card h3 {
  color: #f1f1f1;
}

.theme-switcher {
```

```
  padding: 10px 15px;
  margin: 10px;
  cursor: pointer;
  border: none;
  border-radius: 4px;
  font-weight: bold;
}
```

Probar

Ejecuta npm start. Ahora deberías ver tu aplicación y el nuevo botón "Cambiar a Modo Oscuro". Al hacer clic, verás cómo App.js y todos los demás componentes que has estilizado (como Header y UserDirectory) reaccionan instantáneamente al cambio de tema, todo gracias al Context API.

Resumen de Conceptos Clave

- **createContext**: Para crear un "canal" de datos global.
- **Provider (<ThemeContext.Provider>)**: El componente que *suministra* el valor del estado a todos sus descendientes.
- **useContext**: El hook que permite a cualquier componente *consumir* o "sintonizar" ese valor sin recibir props.
- **Prop Drilling (El Problema Evitado)**: Ya no necesitas pasar el estado del tema a través de App -> Header o App -> UserDirectory. Cada uno lo toma directamente del contexto.