



FULL STACK

**Comenzamos en unos
minutos**

ACADEMY
by NUMEN

Introducción a los Hooks

- parte 2

A person with dark hair tied back, wearing large white headphones, is seated at a desk. They are looking at a laptop. In the background, a large monitor displays a video call with two participants. The entire scene is overlaid with a semi-transparent red filter. The text 'useContext' is centered in white, bold, sans-serif font.

useContext

CONTEXT API

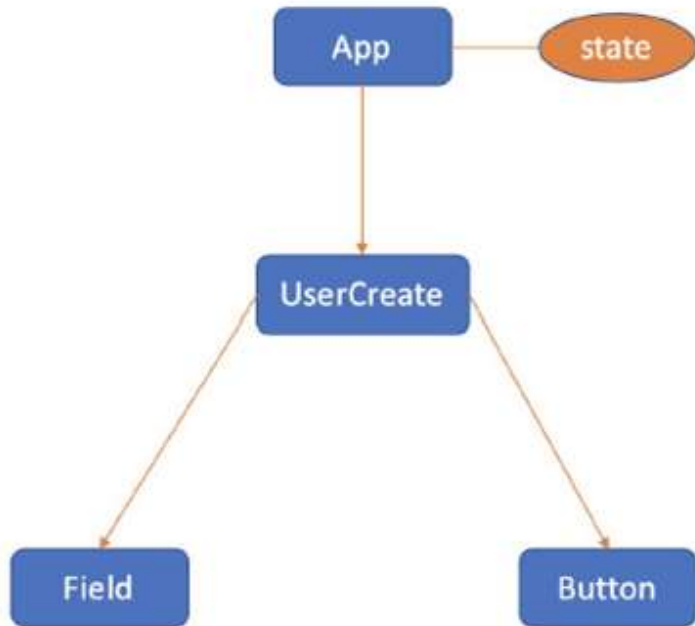
🔥 Context permite compartir el estado en un árbol de componentes.

⚠️ Context se usa principalmente cuando algunos datos tienen que ser accesibles por muchos componentes en diferentes niveles de anidamiento. Por lo tanto, debe ser aplicado con moderación porque hace que la reutilización de componentes sea más difícil.

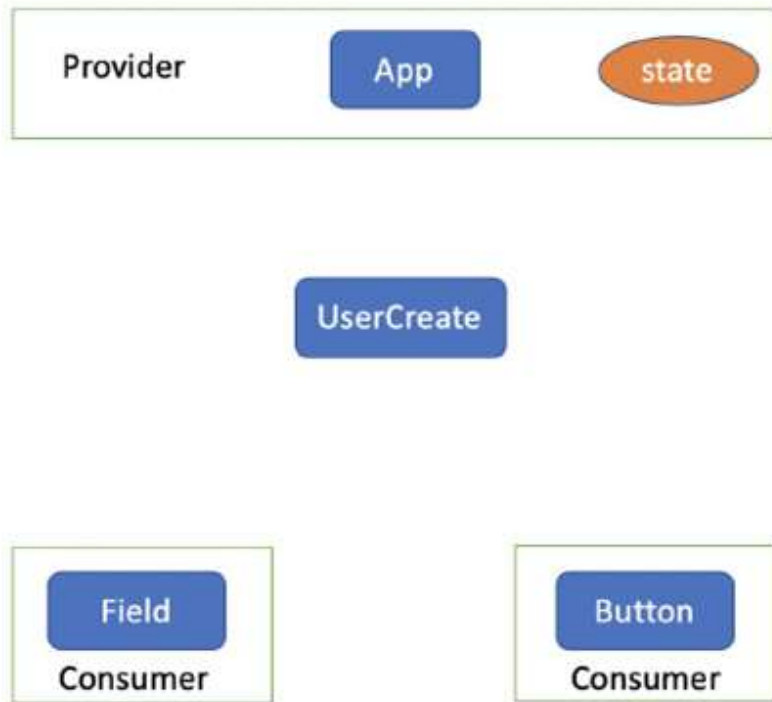
☑ En una aplicación típica de React, los datos se pasan de arriba hacia abajo (de padre a hijo) a través de props, pero esta forma puede resultar incómoda para ciertos tipos de props (por ejemplo, localización, el tema de la interfaz) que son necesarias para muchos componentes dentro de una aplicación. Context proporciona una forma de compartir valores como estos entre componentes sin tener que pasar explícitamente una prop a través de cada nivel del árbol.

☑ Context está diseñado para compartir datos que pueden considerarse “globales”, por lo que podría considerarse similar a una variable global.

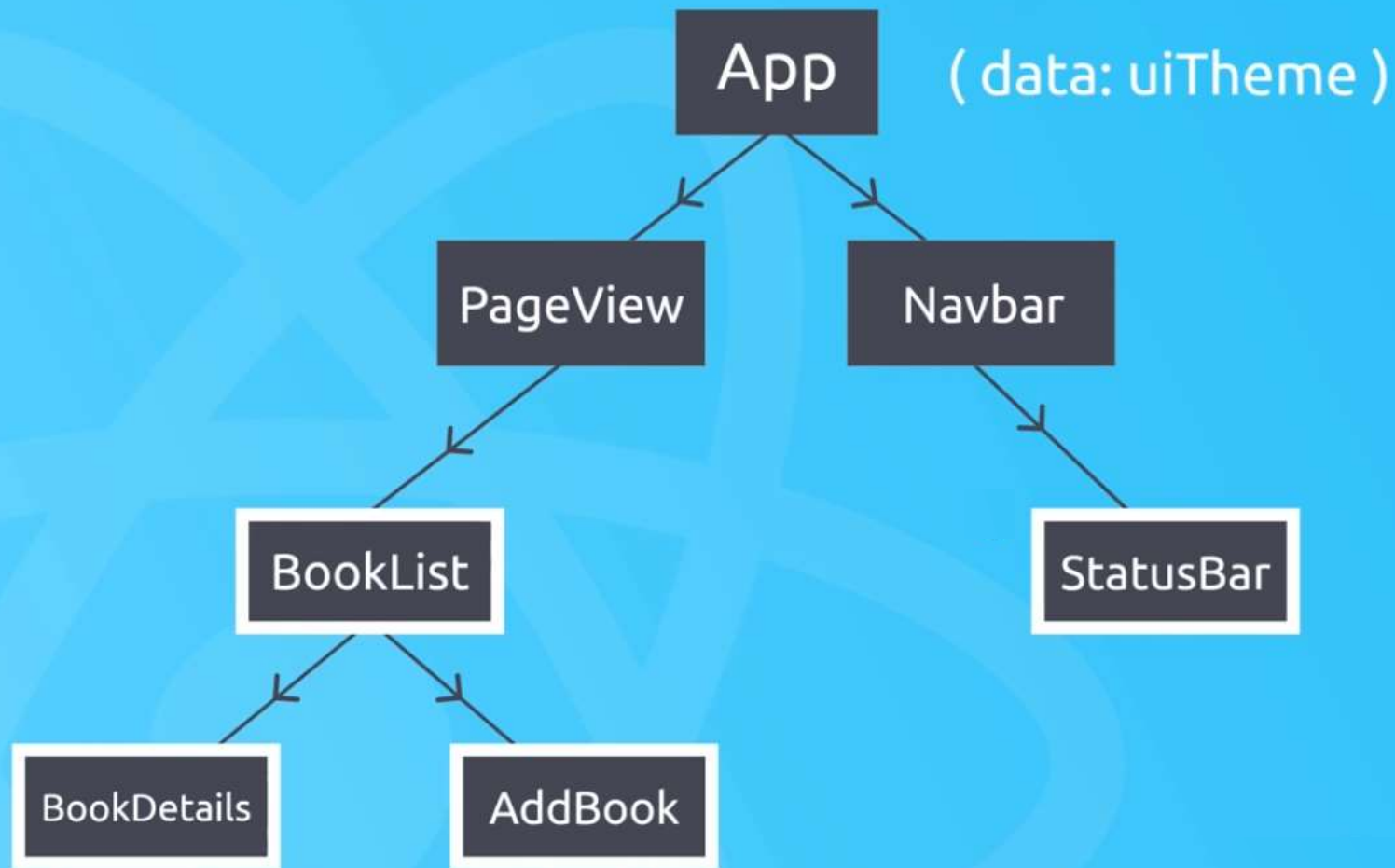
Props Data Flow



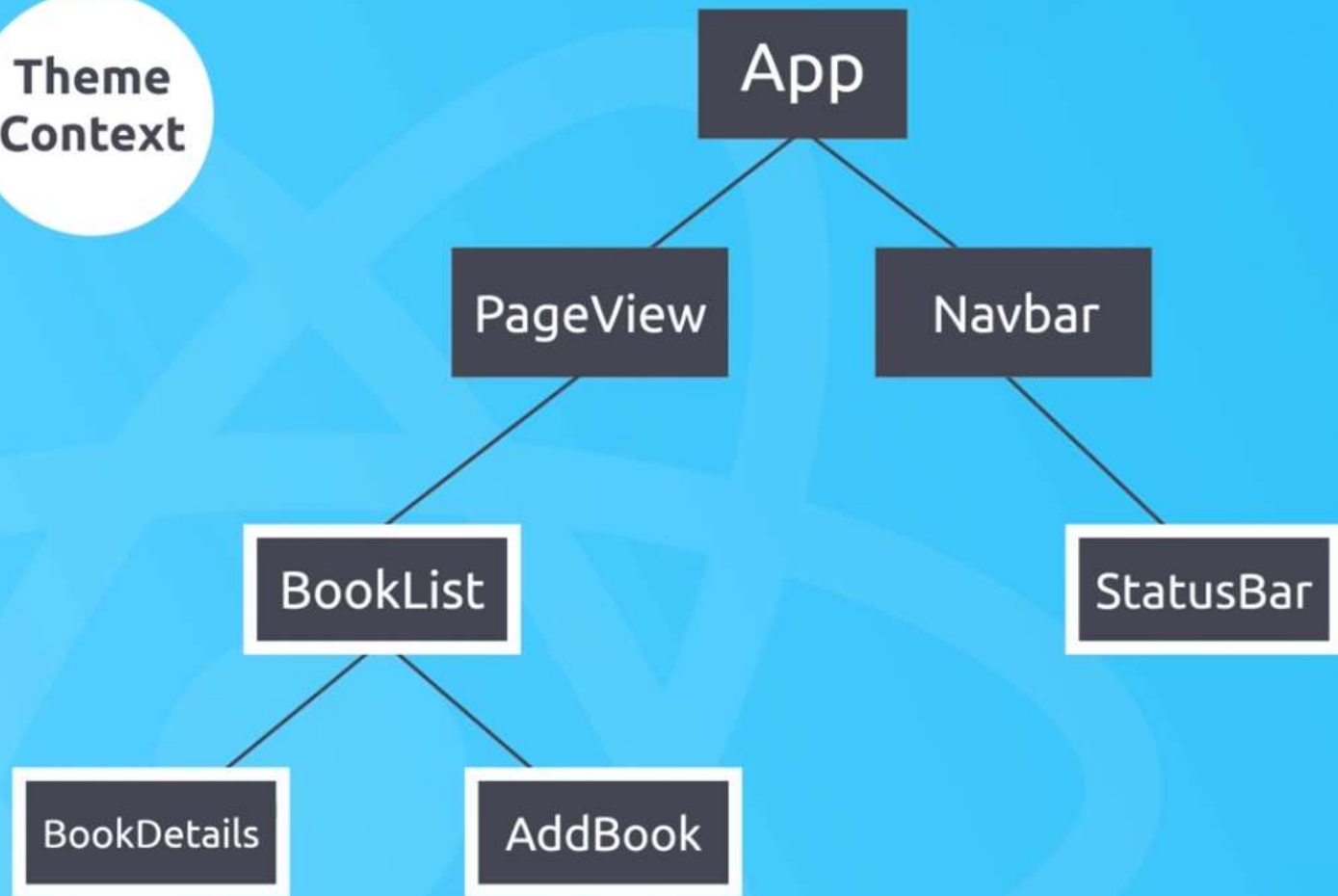
Context Data Flow







**Theme
Context**





☑️ useState es una excelente manera de configurar un estado simple dentro de un componente. Sin embargo, cuando el estado comienza a volverse más complejo y se comparte entre varios componentes, suele ser mejor cambiar a useReducer ya que facilita la escritura de interacciones de estado complejas sin crear un gran desorden de código complejo.

☑️ useReducer nos permite centralizar el estado y manejarlo a través de acciones que son despachadas según indique la función reductora.

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

En primer lugar, importamos
createContext y useState.

```
export const BookContext = createContext()
```

Luego creamos y exportamos
nuestro Contexto.


```
const BookContextProvider = (props) => {  
  const [books, setBooks] = useState([  
    {id: 1, title: 'Código Da Vinci', author: 'Dan Brown'},  
    {id: 2, title: 'Harry Potter', author: 'J. K. Rowling'},  
    {id: 3, title: 'El Principito', author: 'Antoine de Saint-Exupéry'}  
  ])  
  
  return (  
    <BookContext.Provider value={books}>  
      {props.children}  
    </BookContext.Provider>  
  )  
}
```

Después creamos un estado
global que vamos a disponibilizar
a través de context.

```
export default BookContextProvider;
```

Finalmente, creamos el componente
proveedor del estado global y pasamos
el mismo a través del valor único que
recibe la propiedad **value**.

```
function App() {  
  
  return (  
    <>  
      <BookContextProvider>  
        <BookList />  
      </BookContextProvider>  
    </>  
  )  
}  
  
export default App;
```

 El paso siguiente para implementar Context es disponibilizar el estado global que hemos creado. ¿Cómo hacemos esto? Tenemos que colocar dentro del Context Provider todos los componentes que queremos que tengan acceso a dicho estado global.

```

import { useContext } from "react"
import { BookContext } from "../context/BookContextProvider"

const BookList = () => {
  const books = useContext(BookContext)

  return (
    <ul>
      {books.map(book => {
        return (
          <li key={book.id}>
            <h3>{book.title}</h3>
            <p>{book.author}</p>
          </li>
        )
      })}
    </ul>
  )
}

export default BookList;

```



Luego entramos al componente que queremos que tenga acceso al estado global y creamos una variable a la que vamos a asignar dicho estado para ser utilizado.

Para acceder al estado “en contexto” usamos el hook `useContext`, pasándole como parámetro el contexto, que ya previamente importamos.

¿Y si hacemos nuestro sitio claro/oscuro?

```
const Header = () => {  
  return (  
    <header>  
      <form >  
        <fieldset>  
          <label htmlFor="light">Light</label>  
          <input type="radio" name="same" id="light" value="light"/>  
          <label htmlFor="dark">Dark</label>  
          <input type="radio" name="same" id="dark" value="dark"/>  
        </fieldset>  
      </form>  
    </header>  
  );  
}  
  
export default Header;
```

Debemos crear, en la hoja **index.css**, una clase llamada **dark** para agregar las propiedades que deseamos que posea el modo oscuro.

Por ejemplo: color: yellow; background-color: black;

Vamos a manejar el claro/oscuro por medio de radio botones.

Para poder manipularlos le pondremos a cada uno un atributo **value** donde uno será el botón para modo claro ("light") y el otro para modo oscuro ("dark")


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

export const ThemeContext = createContext();

const ThemeProvider = ({children}) => {
  const [theme, setTheme] = useState("light");

  const handleTheme = (e) => {
    if (e.target.value === "light") {
      setTheme("light")
    } else {
      setTheme("dark")
    }
  };

  const data = {theme, handleTheme};

  return <ThemeContext.Provider value={data}>{children}</ThemeContext.Provider>
}

export default ThemeProvider;
```

Como en el ejemplo pasado usaremos un `useState` para manejar el estado.

Este empezará inicialmente como “light” para eventualmente cambiar a “dark”.

Además crearemos una función manejador que detecte que botón se está oprimiendo y en base a eso seleccione a que valor de estado actualizar.

```
import { useContext } from "react";
import ThemeContext from "../context/ThemeProvider";

const Header = () => {

  const {theme, handleTheme} = useContext(ThemeContext)
  return (
    <header className={theme}>
      <form >
        <fieldset>
          <label htmlFor="light">Light</label>
          <input type="radio" name="same" id="light" value="light" onClick={handleTheme}/>
          <label htmlFor="dark">Dark</label>
          <input type="radio" name="same" id="dark" value="dark" onClick={handleTheme}/>
        </fieldset>
      </form>
    </header>
  );
}

export default Header;
```

Importamos el contexto en nuestro componente de radio botones y por medio del hook useContext extraemos el estado estado y la función manejadora.

Colocamos la referencia a la función manejadora en el evento de los botones y el estado en el atributo className para obtener CSS dinámico.

React Router DOM

¿Qué es RRD?

React Router DOM es un paquete npm que permite implementar enrutamiento dinámico en una aplicación web.

Es una biblioteca de enrutamiento del lado del servidor y del cliente con todas las funciones para React que se utiliza para crear aplicaciones de una sola página (SPA), es decir, aplicaciones que tienen muchas páginas o componentes, pero la página nunca se actualiza, sino que el contenido se obtiene dinámicamente en función de la URL.



Hoja de Rutas (Parte 1)

```
import { BrowserRouter as Router, Route, Routes } from
"react-router-dom";

const Rutas = () => {
  return (
    <Router>
      <Routes>
        <Route path="/" />
        <Route path="/Servicios" />
        <Route path="/Contacto" />
      </Routes>
    </Router>
  );
}

export default Rutas;
```

Al acceder al BrowserRouter de RRD, podemos extraer de él los 3 componentes principales que usaremos para ensamblar nuestro sistema de rutas.

Ellos son:

- **Router**: Es el componente que envuelve todos los elementos pertenecerán al sistema de rutas.
- **Routes**: Se utiliza para envolver únicamente los componentes de ruta (Route), separándolos de todos los otros componentes que pertenecen al sistema de rutas pero no son rutas (como un componente Navigation).
- **Route**: Es el componente que utilizamos para crear nuestras rutas a otras páginas (componentes)

Hoja de Rutas (Parte 2)

```
import Rutas from "../components/Rutas";

function App() {
  return (
    <>
      <Rutas />
    </>
  );
}

export default App;
```

Hoja de Rutas

(Parte 3)

Las propiedad **path** nos permite darle un nombre en el directorio a una ruta, siendo la barra inclinada el nombre por defecto de la ruta principal.

La propiedad **element** nos permite ingresar el contenido que deseamos mostrar en la página de ese directorio. Ahí es donde debemos pasar el componente a renderizar.

```
import { BrowserRouter as Router, Route, Routes } from "react-router-dom";
import Inicio from "../pages/Inicio";
import Servicios from "../pages/Servicios";
import Contacto from "../pages/Contacto";
import Error404 from "../pages/Error404";

const Rutas = () => {
  return (
    <Router>
      <Routes>
        <Route path="/" element={<Inicio />} />
        <Route path="/Servicios" element={<Servicios />} />
        <Route path="/Contacto" element={<Contacto />} />
        <Route path="*" element={<Error404 />} />
      </Routes>
    </Router>
  );
}

export default Rutas;
```

Hoja de Rutas

(Parte 4)

```
import { Link, NavLink } from "react-router-dom"

const NavBar = () => {
  return (
    <nav>
      <ul>
        <li><Link to="/">Inicio</Link></li>
        <li><Link to="/Servicios">Servicios</Link></li>
        <li><Link to="/Contacto">Contacto</Link></li>
      </ul>

      <ul>
        <li><NavLink to="/" activeClassName="active">Inicio</NavLink></li>
        <li><NavLink to="/Servicios" activeClassName="active">Servicios</NavLink></li>
        <li><NavLink to="/Contacto" activeClassName="active">Contacto</NavLink></li>
      </ul>
    </nav>
  );
}

export default NavBar;
```


Hoja de Rutas

(Parte 5)

```
import { BrowserRouter as Router, Route, Routes } from "react-router-dom";
import Inicio from "../pages/Inicio";
import Servicios from "../pages/Servicios";
import Contacto from "../pages/Contacto";
import NavBar from "../layout/NavBar";
import Error404 from "../pages/Error404";

const Rutas = () => {
  return (
    <Router>
      <NavBar />
      <Routes>
        <Route path="/" element={<Inicio />} />
        <Route path="/Servicios" element={<Servicios />} />
        <Route path="/Contacto" element={<Contacto />} />
        <Route path="*" element={<Error404 />} />
      </Routes>
    </Router>
  );
}

export default Rutas;
```

Al colocar la Barra de Navegación dentro del Router le estamos indicando a React que deseamos que ese componente se visualice por igual en todas las páginas del sitio.

Del mismo modo podríamos agregar cualquier otro componente que querramos que adquiera la misma visibilidad.

ACADEMY
by NUMEN