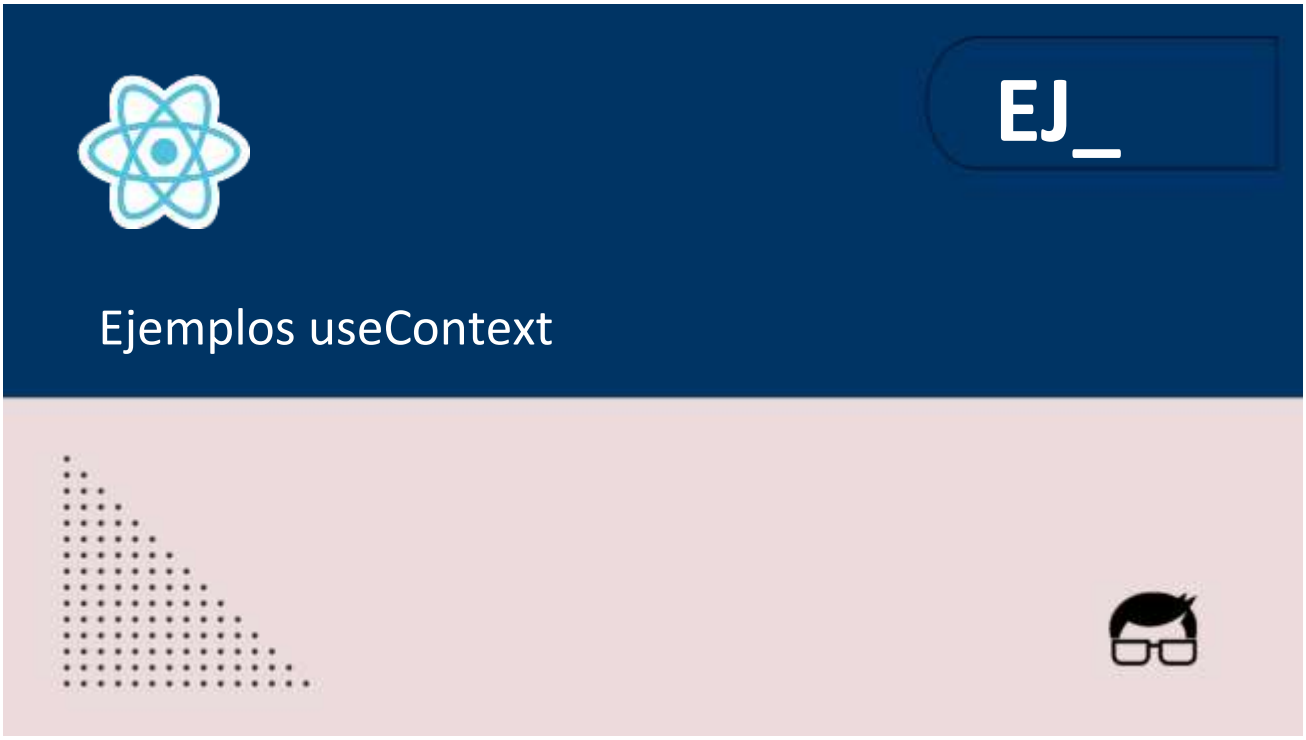

ejemplo de código HTML que incorpora los ejemplos de uso de useContext en React. En este caso, integraré el ejemplo del tema oscuro/claro, ya que es un buen punto de partida para entender cómo funciona useContext. También incluiré los scripts necesarios para usar React y Babel.

```
<!DOCTYPE html>
<html>
<head>
  <title>Ejemplo de React useContext</title>
  <!-- Incluir React y ReactDOM -->
  <script src="https://unpkg.com/react@17/umd/react.development.js"></script>
  <script src="https://unpkg.com/react-dom@17/umd/react-
dom.development.js"></script>
  <!-- Babel para interpretar JSX -->
  <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
</head>
<body>
  <div id="root"></div>

  <script type="text/babel">
    const { useContext, useState, createContext, useEffect } = React;

    // Crear un Contexto para el tema
    const ThemeContext = createContext();

    function App() {
      const [theme, setTheme] = useState('claro'); // 'claro' o 'oscuro'

      useEffect(() => {
```

```
        // Cambiar los estilos del body según el tema
        document.body.style.backgroundColor = theme === 'claro' ? 'white' : 'black';
        document.body.style.color = theme === 'claro' ? 'black' : 'white';
      }, [theme]);

      return (
        <ThemeContext.Provider value={{ theme, setTheme }}>
          <Toolbar />
        </ThemeContext.Provider>
      );
    }

    function Toolbar() {
      return (
        <div>
          <ThemedButton />
        </div>
      );
    }

    function ThemedButton() {
      const { theme, setTheme } = useContext(ThemeContext);

      return (
        <button
          onClick={() => setTheme(theme === 'claro' ? 'oscuro' : 'claro')}
        >
          Cambiar a tema {theme === 'claro' ? 'oscuro' : 'claro'}
        </button>
      );
    }

    ReactDOM.render(<App />, document.getElementById('root'));
  </script>
</body>
</html>
```

el código HTML con el ejemplo de React que utiliza **useContext** para cambiar el tema (claro y oscuro) del documento, incluyendo los estilos del cuerpo (**body**) de la página.

Estructura del Documento HTML

1. **Inclusión de React y ReactDOM**: Se cargan las bibliotecas de React y ReactDOM desde una CDN. Estas son necesarias para crear componentes de React y renderizarlos en el DOM.
2. **Inclusión de Babel**: Babel es incluido para permitir escribir código JSX directamente en el navegador. JSX es una extensión de la sintaxis de JavaScript que es comúnmente usada en React.

3. **Contenedor del Componente React**: El ** ** es donde se montará el componente de React.

Código JavaScript (JSX) para React

1. **Importación de Hooks de React**: Se importan **useContext**, **useState**, **createContext**, y **useEffect** de React. Estos hooks son esenciales para manejar el estado y el contexto, así como para responder a cambios en el ciclo de vida del componente.

2. **Creación del Contexto del Tema**: Se crea un nuevo contexto llamado **ThemeContext** usando **createContext()**. Este contexto permitirá compartir el estado del tema (claro u oscuro) entre los componentes.

3. **Componente App**:

    - Se utiliza **useState** para mantener el estado del tema, inicializado en **'claro'**.

    - **useEffect** se utiliza para actualizar los estilos del cuerpo de la página cada vez que el tema cambia. Dependiendo del tema actual, se establecen los colores de fondo y texto del cuerpo (**body**).

    - **ThemeContext.Provider** envuelve el componente **Toolbar**, pasándole el estado del tema y la función para actualizarlo (**setTheme**).

4. **Componente Toolbar**: Este componente es simplemente un contenedor para **ThemedButton**. No utiliza directamente el contexto, pero forma parte de la estructura de la aplicación.

5. **Componente ThemedButton**:

    - Aquí se utiliza **useContext** para acceder al contexto del tema creado (**ThemeContext**), permitiendo al componente acceder al estado del tema actual y a la función **setTheme** para cambiarlo.

    - El botón cambia el tema de claro a oscuro y viceversa cuando se hace clic en él.

6. **Renderizado con ReactDOM**: **ReactDOM.render** se usa para montar el componente **App** en el elemento **div** con **id="root"** en el documento HTML.

Funcionamiento General

Cuando se carga la página, el tema inicial es claro (fondo blanco y texto negro). Al hacer clic en el botón, el tema cambia a oscuro (fondo negro y texto blanco), y viceversa. Este cambio se realiza a través del contexto, que es accesible por todos los componentes que lo consumen, permitiendo una actualización eficiente y centralizada del estado del tema en toda la aplicación.

Más ejemplos

useContext es un hook en React que permite consumir datos de un contexto en componentes funcionales, facilitando el manejo del estado y evitando tener que pasar props a través de varios niveles de componentes. Aquí te presento tres ejemplos básicos para ilustrar cómo puedes usar useContext.

Ejemplo 1: Tema Oscuro/Claro
Este ejemplo muestra cómo usar useContext para manejar un tema oscuro/claro en una aplicación.

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

// Crear un Contexto para el tema
const ThemeContext = createContext();

function App() {
 const [theme, setTheme] = useState('claro');

 return (
  <ThemeContext.Provider value={{ theme, setTheme }}>
   <Toolbar />
  </ThemeContext.Provider>
 );
}

function Toolbar() {
 return (
  <div>
   <ThemedButton />
  </div>
 );
}

function ThemedButton() {
 const { theme, setTheme } = useContext(ThemeContext);

 return (
  <button
   onClick={() => setTheme(theme === 'claro' ? 'oscuro' : 'claro')}
  >
   Cambiar a tema {theme === 'claro' ? 'oscuro' : 'claro'}
  </button>
 );
}
```


Ejemplo 2: Usuario Autenticado
Este ejemplo muestra cómo compartir información sobre un usuario autenticado en toda la aplicación.

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

// Crear un Contexto para el usuario
const UserContext = createContext();

function App() {
 const [user, setUser] = useState(null);

 return (
  <UserContext.Provider value={{ user, setUser }}>
   <LoginComponent />
   <UserProfile />
  </UserContext.Provider>
 );
}

function LoginComponent() {
 const { setUser } = useContext(UserContext);

 const handleLogin = () => {
  // Lógica de inicio de sesión (simulada)
  setUser({ name: 'Usuario', isLoggedIn: true });
 };

 return <button onClick={handleLogin}>Iniciar Sesión</button>;
}

function UserProfile() {
 const { user } = useContext(UserContext);

 if (!user) {
  return <p>No hay usuario autenticado</p>;
 }

 return <p>Bienvenido, {user.name}</p>;
}
```


Ejemplo 3: Internacionalización (i18n)
Este ejemplo demuestra cómo implementar la internacionalización en una aplicación React usando useContext.

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

// Contexto para el idioma
const LanguageContext = createContext();

const texts = {
  es: {
    welcome: "Bienvenido"
  },
  en: {
    welcome: "Welcome"
  }
};

function App() {
  const [language, setLanguage] = useState('es');

  return (
    <LanguageContext.Provider value={{ language, setLanguage }}>
      <LanguageSelector />
      <WelcomeMessage />
    </LanguageContext.Provider>
  );
}

function LanguageSelector() {
  const { setLanguage } = useContext(LanguageContext);

  return (
    <div>
      <button onClick={() => setLanguage('es')}>ES</button>
      <button onClick={() => setLanguage('en')}>EN</button>
    </div>
  );
}

function WelcomeMessage() {
  const { language } = useContext(LanguageContext);

  return <p>{texts[language].welcome}</p>;
}
```

.