Context

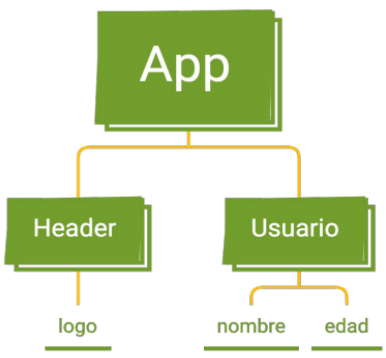
Objetivos

- Comprender que son los estados locales
- ¿Que es Prop Drilling?
- Conocer Contex API
- Cómo funciona la API de context de React
- Ventajas y desventajas de context API
- Usar Context para traspasar datos entre componentes

Estado local

En React, el estado local de un componente es una variable que se almacena dentro de ese componente. El estado local se puede usar para almacenar datos que cambian con el tiempo, como el número de elementos en una lista o el valor de un campo de entrada.
Cada componente tiene sus propios estados, los cuales no se comparten ni se pueden acceder directamente a ellos desde otro componente.

Para poder hacer modificaciones de estados en otros componentes, es posible pasar un estado y/o la función que lo modifica a un componente hijo, para eso simplemente los pasamos como props. Esto a medida que un proyecto va creciendo nos genera un problema llamado Prop Drilling.



- En este diagrama tenemos la componente App, Header y Usuario.
- Header no puede cambiar el estado nombre o el estado edad de un Usuario, por lo menos no directamente.

{desafío} latam_

Prop Drilling

En React, el prop drilling es el proceso de pasar datos a través de varios componentes anidados para llegar al componente que los necesita. Esto puede conducir a un código complejo y difícil de mantener, ya que cada componente intermedio necesita manejar los datos que se le pasan.

A continuación, veremos un ejemplo:

```jsx
const App = () => {
  const [data, setData] = useState({
    name: "John Doe",
  });

  return (
    <div>
      <MyComponent data={data} />
    </div>
  );
};

const MyComponent = ({ data }) => {
  return (
    <div>
      <h1>{data.name}</h1>
    </div>
  );
};
```

El prop drilling puede causar los siguientes problemas:

Código complejo y difícil de mantener: A medida que una aplicación crece, el prop drilling puede conducir a un código complejo y difícil de mantener. Esto se debe a que cada componente intermedio necesita manejar los datos que se le pasan.

Ineficiente: El prop drilling puede ser ineficiente, ya que React debe actualizar todos los componentes que dependen de un componente con prop drilling.

Reducción de la legibilidad: El prop drilling puede reducir la legibilidad del código, ya que puede ser difícil seguir la ruta que siguen los datos a través de la jerarquía de componentes.

Hay varias formas de evitar el prop drilling:

Usar el context: El context es una forma de compartir datos entre componentes de una manera eficiente. Esto puede ayudar a evitar pasar datos a través de varios componentes anidados.

Usar un biblioteca de gestión de estado: Una biblioteca de gestión de estado (Redux, Recoil)puede ayudar a organizar el estado de una aplicación. Esto puede ayudar a evitar el

prop drilling al permitir que los datos se compartan entre componentes de una manera más eficiente.

En general, el prop drilling es una práctica que debe evitarse siempre que sea posible.

Contex Api

La API de context de React es una forma de compartir datos entre componentes de una manera eficiente. Los datos del context se pueden usar para cosas como el idioma, la configuración de la aplicación o el estado de la sesión del usuario.
La API de context de React consta de dos componentes:

1. **El componente proveedor**: El componente proveedor proporciona el valor del context.

2. **El componente consumidor**: El componente consumidor accede al valor del context.

Componente proveedor

El componente proveedor es un componente que proporciona el valor del context. Para crear un componente proveedor, debe envolver su componente en un componente Provider(). El argumento del componente Provider() es el objeto de context que desea proporcionar.

Componente consumidor

El componente consumidor es un componente que accede al valor del context. Para acceder al valor del context, debe usar el hook useContext(). El hook useContext() toma un objeto de context como argumento y devuelve el valor actual del context.

Cómo funciona la API de context de React

La API de context de React funciona creando un árbol de contextos. El componente proveedor crea el nodo raíz del árbol de contextos. Los componentes consumidores que están envueltos en el componente proveedor heredan el context del componente proveedor.

Cuando cambia el valor del context, React actualiza todos los componentes consumidores que están envueltos en el componente proveedor.

La API de context de React tiene varias ventajas:

1. **Eficiencia**: La API de context de React es eficiente porque solo actualiza los componentes consumidores que necesitan actualizarse.

2. **Organización**: La API de context de React ayuda a organizar el código al mantener el estado de la aplicación en un solo lugar.

3. **Flexibilidad**: La API de context de React es flexible y se puede usar para compartir cualquier tipo de datos.

Desventajas de la API de context de React:

1. **Complejidad**: La API de context de React puede ser compleja de entender al principio.

2. **Depuración**: La API de context de React puede ser difícil de depurar si algo sale mal.

Usos comunes de la API de context de React:

1. Compartir el idioma de la aplicación

2. Compartir la configuración de la aplicación

3. Compartir el estado de la sesión del usuario

4. Compartir datos de una API

5. Compartir datos de un caché

En conclusión, la API de context de React es una herramienta poderosa que puede ayudar a crear aplicaciones React más eficientes y organizadas.

Creando nuestro primer context

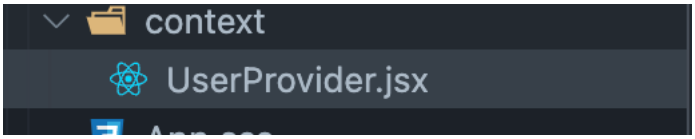
A continuación vamos a crear paso a paso nuestro primero context

Paso 1

Generaremos una carpeta llamada **context** en la cual almacenaremos todos nuestros contextos

Paso 2

En la carpeta context vamos a crear el archivo que contendrá nuestro primer context, en este caso vamos a manejar datos de un usuario y consumiremos esos datos desde distintos componentes. Para esto crearemos el archivo UserProvider.



Paso 3

- Importamos las funciones createContext() y useState() de React.
- Creamos un context llamado UserContext usando la función createContext(). Esto crea un objeto de contexto. El objeto de contexto se puede usar para compartir datos entre componentes.
- Creamos un componente proveedor llamado UserProvider. El componente proveedor es el componente que proporciona el valor del context.
- En el componente UserProvider, usamos el hook useState() para crear un estado llamado user. El estado user contiene el nombre y la edad del usuario.
- En el componente UserProvider, envolvemos nuestros componentes hijos en un componente UserContext.Provider. El argumento del componente UserContext.Provider es el valor del context que queremos proporcionar.
- Exportamos el componente UserProvider por defecto.

Nuestro context debería verse de la siguiente manera.

```jsx
import { createContext, useState } from 'react'  4.3k (gzipped: 1.9k)

export const UserContext = createContext()

const UserProvider = ({ children }) => {
  const [user, setUser] = useState({ name: 'John Doe', age: 25 })
  return (
    <UserContext.Provider value={{ user, setUser }}>
      {children}
    </UserContext.Provider>
  )
}

export default UserProvider
```

Paso 4 cómo usar el context:

Para usar el context, primero debemos envolver nuestra aplicación en el componente UserProvider.

Nuestro envoltorio debería quedar de la siguiente forma:

```jsx
import './App.css'
import ChangeUser from './components/ChangeUser'
import Edit from './components/Edit'
import Home from './components/Home'
import UserProvider from './context/UserContext'

function App () {
  return (
    <>
      <UserProvider>
        <Home />
        <Edit />
        <ChangeUser />
      </UserProvider>
    </>
  )
}

export default App
```

Como podemos ver se importa el provider, y se envuelven los componentes que van a usar ese context, para usar los valores que proporciona el context usamos el hook useContext, nuestros componentes se deberían ver de la siguiente forma:

```jsx
import { useContext } from "react";  4.1k (gzipped: 1.8k)
import { UserContext } from "../context/UserContext";
const Home = () => {
  const { user } = useContext(UserContext);
  return (
    <div>
      Home
      <p>{user.name}</p>
    </div>
  );
};

export default Home;
```

Ya hemos creado y consumido nuestro primer context ahora solo nos queda ver en el navegador los resultados.