Guia Rápido sobre Context API no React 18

O que é o Context API?

A **Context API** no React é uma maneira de passar dados através da árvore de componentes sem precisar passar `props` manualmente em cada nível. Isso é útil para compartilhar informações "globais" que muitas partes da aplicação podem precisar, como o estado de autenticação, tema, preferências de idioma, etc.

Principais Características:

- **Criação de Contexto**: Você pode criar um contexto usando `React.createContext()`.
- **Provider**: Um **Provider** é usado para disponibilizar dados do contexto para os componentes filhos.
- **Consumer**: Componentes podem acessar os dados fornecidos pelo Provider usando `useContext()` ou o padrão antigo de `Context.Consumer`.
- **Performance**: Se não for usado corretamente, pode causar renderizações desnecessárias dos componentes que consomem o contexto.

Prós:

- **Simplicidade**: Fácil de implementar e entender, especialmente para gerenciamento de estados globais simples.
- **Sem Prop Drilling**: Evita a necessidade de passar props por múltiplos níveis de componentes, tornando o código mais limpo.
- **Bom para Estados Globais**: Perfeito para estados globais como tema, autenticação e configuração de idioma.

Contras:

- **Performance**: Pode levar a renderizações desnecessárias em componentes que consomem o contexto, mesmo que os dados que eles dependem não tenham mudado.

- **Escalabilidade**: Não é a melhor solução para aplicações maiores, onde o gerenciamento de estado é mais complexo. Para esses casos, bibliotecas como Redux ou Zustand podem ser mais apropriadas.
- **Reatividade**: Diferente de estados locais, mudanças no contexto não oferecem granularidade de atualização, ou seja, todos os componentes que consomem o contexto podem ser renderizados novamente.

Exemplo de `AuthContext`

Aqui está um exemplo de como criar um contexto para gerenciar a autenticação do usuário, chamado `AuthContext`:

Passo 1: Crie o `AuthContext`

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

// 1. Crie o Contexto
const AuthContext = createContext(null);

// 2. Provider para o AuthContext
export const AuthProvider = ({ children }) => {
  const [user, setUser] = useState(null);  // Estado para o
usuário autenticado

  const login = (userData) => {
    setUser(userData);
  };

  const logout = () => {
    setUser(null);
  };

  return (
    <AuthContext.Provider value={{ user, login, logout }}>
      {children}
    </AuthContext.Provider>
  );
};
```

```
// 3. Hook para usar o AuthContext
export const useAuth = () => {
  return useContext(AuthContext);
};
```

Passo 2: Consumindo o `AuthContext` em Componentes

```
import React from 'react';
import { useAuth } from './AuthContext';

const Navbar = () => {
  const { user, login, logout } = useAuth();

  return (
    <nav>
      {user ? (
        <>
          <p>Olá, {user.name}!</p>
          <button onClick={logout}>Logout</button>
        </>
      ) : (
        <button onClick={() => login({ name: 'Usuário Exemp
lo' })}>Login</button>
      )}
    </nav>
  );
};

export default Navbar;
```

Passo 3: Usando o `AuthProvider` no Componente Pai

```
import React from 'react';
import { AuthProvider } from './AuthContext';
import Navbar from './Navbar';
```

```jsx
const App = () => {
  return (
    <AuthProvider>
      <div className="App">
        <Navbar />
        {/* Outros componentes */}
      </div>
    </AuthProvider>
  );
};

export default App;
```

Explicação:

1. **Criação do Contexto**: O contexto é criado com `createContext()`, e o valor inicial é `null`.
2. **Provider**: O `AuthProvider` é um componente que envolve a aplicação e disponibiliza as funções `login` e `logout`, além do estado `user`, para os componentes que necessitarem.
3. **useContext**: Dentro dos componentes, o `useAuth` facilita o acesso ao contexto para verificar o estado de autenticação e chamar funções de login/logout.