




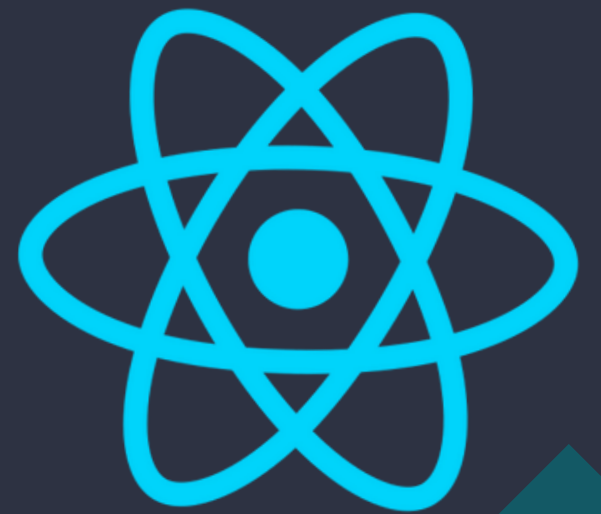
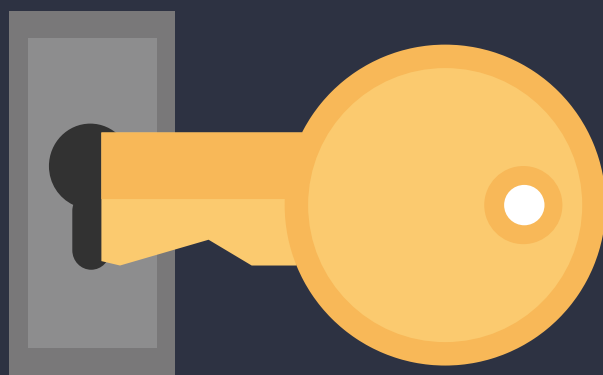
# User Authentication in React using Context API: A Simple Guide!



 Username

 Password

LOGIN



# Why Context API for Authentication?

Avoids **prop drilling** when managing authentication state.

Enables a **global**, easily accessible user authentication system.



# Steps to Implement

1. Create a **User Context**.
2. Wrap your app in a **Provider**.
3. **Consume** the context in components.



## Step 1: Create a Context

Define a Context for authentication:



```
1 import { createContext } from  
  "react";  
2  
3 export const AuthContext =  
  createContext();
```



## Step 2 : Create a Provider

Wrap your app with a Provider to share auth state:

```
1 import { useState } from "react";
2 import { AuthContext } from "../AuthContext";
3
4 export const AuthProvider = ({ children }) => {
5   const [user, setUser] = useState(null); // User state
6
7   const login = (userData) => setUser(userData);
8   const logout = () => setUser(null);
9
10  return (
11    <AuthContext.Provider value={{ user, login, logout }}>
12      {children}
13    </AuthContext.Provider>
14  );
15 };
16
```



## Step 3 : Consume the Context

Use the context in child components:

```
1 import { useContext } from "react";
2 import { AuthContext } from "../AuthContext";
3
4 const Dashboard = () => {
5   const { user, logout } = useContext(AuthContext);
6
7   return (
8     <div>
9       {user ? (
10         <>
11           <h1>Welcome, {user.name}!</h1>
12           <button onClick={logout}>Logout</button>
13         </>
14       ) : (
15         <p>Please log in.</p>
16       )}
17     </div>
18   );
19 };
20
```



## Adding Authentication Logic

Implement login/logout logic, e.g., using APIs:

```
1 const loginUser = async (credentials) => {  
2   const response = await fetch('/login', {  
3     method: 'POST',  
4     body: JSON.stringify(credentials),  
5   });  
6  
7   const userData = await response.json();  
8   login(userData); // Update the context  
9 };  
10
```



# Protecting Routes

Create a reusable ProtectedRoute:

```
1 const ProtectedRoute = ({ children }) => {  
2   const { user } = useContext(AuthContext);  
3  
4   return user ? children : <Navigate to="/login" />;  
5 };  
6
```

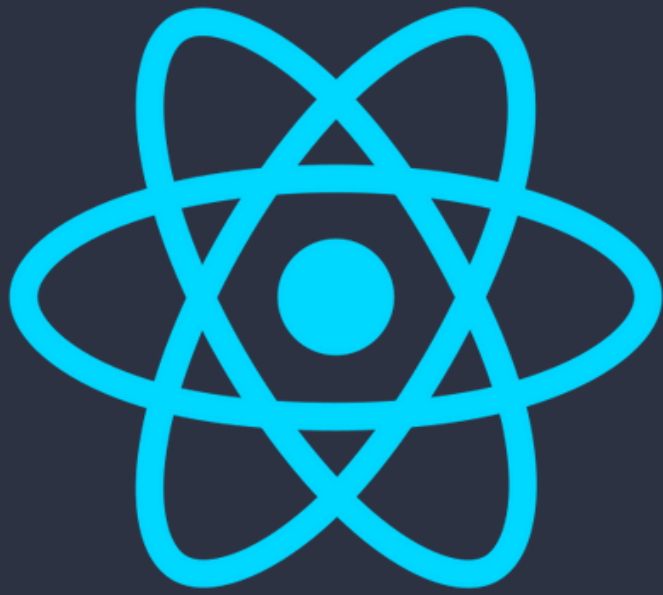




## Conclusion

With Context API, managing user authentication becomes seamless and scalable. It's perfect for **small-to-medium** projects. For larger apps, consider state libraries like **Redux** or tools like **Firebase**.





10

# FOLLOW ME FOR MORE



**PRAVEEN RAJ V**

Praveen-raj-v1211

