Certainly! Let's create an example of managing user authentication and saving user information using React context and the `useContext` hook.

Step 1: Create an AuthContext

First, create a context for managing user authentication.

```
// AuthContext.js
import { createContext, useState } from 'react';

const AuthContext = createContext();

const AuthProvider = ({ children }) => {
  const [user, setUser] = useState(null);

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

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

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

export { AuthProvider, AuthContext };
```

Step 2: Wrap your App with the AuthProvider

Wrap your application (or part of it) with the `AuthProvider` .

```
// App.js
import React from 'react';
import { AuthProvider } from './AuthContext';
import UserInfo from './UserInfo';
import LoginForm from './LoginForm';

const App = () => {
  return (
    <AuthProvider>
      <div>
        <h1>User Authentication Example</h1>
        <UserInfo />
        <LoginForm />
      </div>
    </AuthProvider>
  );
};

export default App;
```

Step 3: Use `useContext` in a UserInfo Component

Create a component that uses the `useContext` hook to display user information and provide a logout button.

```
// UserInfo.js
import React, { useContext } from 'react';
import { AuthContext } from './AuthContext';

const UserInfo = () => {
  const { user, logout } = useContext(AuthContext);

  return (
    <div>
      {user ? (
        <div>
          <h2>Welcome, {user.username}!</h2>
          <button onClick={logout}>Logout</button>
        </div>
      ) : (
        <p>Please log in.</p>
      )}
    </div>
  );
};

export default UserInfo;
```

Step 4: Use `useContext` in a LoginForm Component

Create a component that uses the `useContext` hook to handle user login.

```
// LoginForm.js
import React, { useState, useContext } from 'react';
import { AuthContext } from './AuthContext';

const LoginForm = () => {
  const [username, setUsername] = useState('');
  const [password, setPassword] = useState('');

  const { login } = useContext(AuthContext);

  const handleLogin = () => {
    // Simulate user authentication (you might want to use a server and authentication
flow here)
    const userData = { username, password };
    login(userData);
  };

  return (
    <div>
      <h3>Login</h3>
      <label>
        Username:
        <input type="text" value={username} onChange={(e) => setUsername(e.target.value)}
/>
      </label>
      <br />
      <label>
        Password:
        <input type="password" value={password} onChange={(e) =>
setPassword(e.target.value)} />
      </label>
      <br />
      <button onClick={handleLogin}>Login</button>
    </div>
  );
};

export default LoginForm;
```

Step 5: Render Components

Finally, render your components in the main file.

```
// index.js
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);
```

Now, you have a simple example of managing user authentication using React context and the `useContext` hook. The `UserInfo` component displays user information if the user is logged in, and the `LoginForm` component allows the user to log in.