Certainly! Let's create a complete example with both user authentication and theme management using separate contexts.

Step 1: Create User Context

Create a context for user authentication.

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

const UserContext = createContext();

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

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

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

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

export { UserProvider, UserContext };
```

Step 2: Create Theme Context

Create a context for managing themes.

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

const ThemeContext = createContext();

const ThemeProvider = ({ children }) => {
  const [isDarkMode, setDarkMode] = useState(false);

  const toggleTheme = () => {
    setDarkMode((prevMode) => !prevMode);
  };

  const theme = isDarkMode ? 'dark' : 'light';

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

export { ThemeProvider, ThemeContext };
```

Step 3: Wrap your App with Both Providers

Wrap your application with both the `UserProvider` and `ThemeProvider`.

```jsx
// App.js
import React from 'react';
import { UserProvider } from './UserContext';
import { ThemeProvider } from './ThemeContext';
import UserInfo from './UserInfo';
import LoginForm from './LoginForm';
import ThemedComponent from './ThemedComponent';

const App = () => {
  return (
    <UserProvider>
      <ThemeProvider>
        <div>
          <h1>User Authentication and Theme Example</h1>
          <UserInfo />
          <LoginForm />
          <ThemedComponent />
        </div>
      </ThemeProvider>
    </UserProvider>
  );
};

export default App;
```

Step 4: Use `useContext` in Components

Use `useContext` in components to access both user and theme contexts.

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

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

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

export default UserInfo;


// ThemedComponent.js
import React, { useContext } from 'react';
import { ThemeContext } from './ThemeContext';

const ThemedComponent = () => {
  const { theme, toggleTheme } = useContext(ThemeContext);

  return (
    <div style={{ background: theme === 'dark' ? '#333' : '#fff', color: theme === 'dark'
? '#fff' : '#333' }}>
      <h2>Themed Component</h2>
      <p>Current Theme: {theme}</p>
      <button onClick={toggleTheme}>Toggle Theme</button>
    </div>
  );
};

export default ThemedComponent;
```

Step 5: Render Components

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 complete example with both user authentication and theme management using separate contexts. Components can independently access the user and theme contexts using the `useContext` hook.