사이드 에펙트 처리 (7) ⭐context API

- **state, props의 한계**

: 컴포넌트가 트리 구조로 중첩되어 깊고 넓게 뻗어나가면서, 각기 다른 컴포넌트에 영향을 주어야 할 일들이 발생

React_Passing Data Deely with Context

```
//AuthContext.js
import React from 'react';

const AuthContext = React.createContext({
    isLoggedIn: false;
})

export default AuthContext;
```

- 컴포넌트 트리가 깊고 복잡해지면 state, props만 가지고 상태를 주고받는 방식의 복잡도가 급격히 늘어난다.
- 리액트에서는 context API 를 제공하는데, 전역 상태 관리를 할 수 있다.

- **커스텀 context 컴포넌트 만들기**

```
//AuthContext.js
import React from 'react';

const AuthContext = React.createContext({
    isLoggedIn: false,
    onLogin: (email, password) => {},
    onLogout : () => {}
});

export const AuthContextProvider = () => {
    return(
        <AuthContext.Provider value={{
                isLoggedIn: isLoggedIn,
                onLogout: logoutHandler,
                onLogin: loginHandler
            }}>
            {props.children}
        </AuthContext.Provider>
    )
};
```

- **Reducer와 Context로 스케일 업**

⭐ Scaling Up with Reducer and Context

> Reducers let you consolidate a component's state update logic. context lets you pass information deep down to other components. you can combince reducers adn context together to manage state of a complex screen.

- context의 한계

  : 잦은 변화가 일어나는 상태를 다루기에는 적합하지가 않다.

  : 모든 컴포넌트 간 통신을 다 context에서 다루려고 하면 안된다.

---

context에 상태와 로직을 모두 관리하면 컴포넌트에서는 뷰만 신경을 쓸 수 있다.

- context에도 한계가 분명히 존재한다.
  - 잦은 상태 변화를 핸들링하는 점에서 성능이 떨어지고,
  - 모든 커뮤니케이션을 다 하려고 하면 너무 무거워진다.