Context api

Context API

React의 Context API에 대해 알아보겠습니다.

Context API는 React 컴포넌트 트리 전체에서 데이터를 공유할 수 있는 방법을 제공합니다. 이를 통해 중첩된 구조에서 데이터를 전달하는 데 있어서 불필요한 props drilling을 방지할 수 있습니다.

> 💡 props drilling은 React 컴포넌트 구조에서 하위 컴포넌트로 데이터를 전달하기 위해 상위 컴포넌트를 거치는 것을 의미합니다. 이는 컴포넌트 구조가 깊어지면서 코드가 복잡해지고 유지보수가 어려워지는 문제를 유발합니다. Context API는 이러한 문제점을 해결하기 위한 방법 중 하나입니다.

Context를 사용하기 위해서는 createContext() 함수를 사용하여 Context 객체를 생성해야 합니다. 이후 Context.Provider 컴포넌트를 이용해 하위 컴포넌트에게 데이터를 전달할 수 있습니다. Provider는 value prop을 통해 데이터를 전달합니다.

Consumer 컴포넌트를 이용하면 Provider에서 전달한 데이터를 사용할 수 있습니다. Consumer는 함수형 컴포넌트 또는 클래스 컴포넌트로 작성할 수 있습니다. Context를 사용하는 컴포넌트는 Consumer를 이용해 데이터를 사용하거나, useContext Hook을 사용하여 데이터를 가져올 수 있습니다.

Context API는 Redux와 같은 상태 관리 라이브러리를 대체할 수는 없지만, 단순한 상태 관리에는 편리하게 사용할 수 있습니다.

다음은 Context API를 사용한 예제입니다.

src > Exam9.js에 작성합니다.

```
// Exam9.js
import React, { createContext, useContext } from 'react';

export const UserContext = createContext();
```

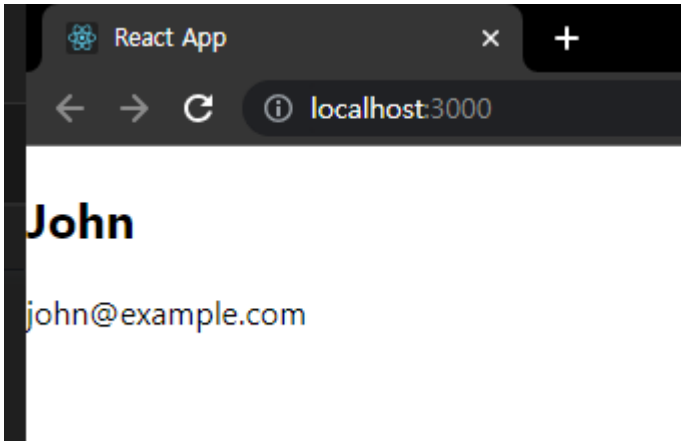
```
export default function Exam9() {
  const user = useContext(UserContext);

  return (
    <div>
      <h2>{user.name}</h2>
      <p>{user.email}</p>
    </div>
  );
}
```

위 코드에서, UserContext 객체를 createContext() 함수를 사용하여 생성하고, UserContext.Provider를 사용하여 user 객체를 하위 컴포넌트에게 전달합니다. Profile 컴포넌트에서 useContext Hook을 사용하여 UserContext에서 데이터를 가져와 사용합니다.

```
import Exam9, { UserContext } from './Exam9'

function App() {

  const user = {
    name: 'John',
    email: 'john@example.com',
  };

  return (
    <div className="App">
      <UserContext.Provider value={user}>
        <Exam9 />
      </UserContext.Provider>
    </div>
  );
}

export default App;
```

이 예제에서는 App 컴포넌트에서 생성한 user 객체를 Profile 컴포넌트에서 사용하고 있습니다.

Context API에서 Provider의 역할은 상위 컴포넌트에서 하위 컴포넌트로 데이터를 전달해주는 역할을 합니다. Provider 컴포넌트는 value prop을 통해 하위 컴포넌트에게 데이터를 제공합니다. useContext Hook이나 Consumer 컴포넌트를 이용하여 Provider에서 전달한 데이터를 사용할 수 있습니다.



다음은 Context API를 사용한 심화 예제입니다.

이번에는 바로 App.js 에 작성하겠습니다.

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

function App() {
  const UserContext = createContext();

  function Profile() {
    const { user } = useContext(UserContext);

    return (
      <div>
        <h2>{user.name}</h2>
        <p>{user.email}</p>
      </div>
    );
  }

  const [user, setUser] = useState(null);

  const handleLogin = () => {
    setUser({
      name: 'John',
      email: 'john@example.com',
    });
  };

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

  return (
    <div>
      <UserContext.Provider value={{ user, handleLogin, handleLogout }}>
```

```
      <div>
        {user ? (
          <div>
            <Profile />
            <button onClick={handleLogout}>Logout</button>
          </div>
        ) : (
          <button onClick={handleLogin}>Login</button>
        )}
      </div>
    </UserContext.Provider>
    </div>
  );
}

export default App;
```

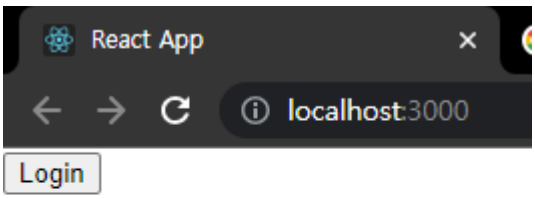
위 코드에서, App 컴포넌트에서는 useState Hook을 사용하여 user state와 handleLogin, handleLogout 함수를 정의합니다.

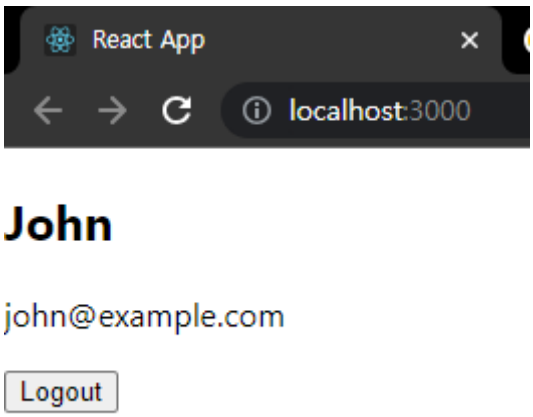
이후 UserContext.Provider를 사용하여 user state와 handleLogin, handleLogout 함수를 하위 컴포넌트에게 전달합니다.

Profile 컴포넌트에서는 useContext Hook을 사용하여 UserContext에서 user state를 가져와 사용합니다.

위 예제에서는 Login 버튼을 클릭하면 handleLogin 함수가 실행되어 user state가 변경되고, Logout 버튼을 클릭하면 handleLogout 함수가 실행되어 user state가 null로 변경됩니다.

Profile 컴포넌트에서는 user state가 null이 아닐 경우, 사용자 정보를 출력하고 Logout 버튼을 렌더링합니다.





이 예제에서는 user state와 handleLogin, handleLogout 함수를 Context API를 이용하여 하위 컴포넌트에게 전달하고, Profile 컴포넌트에서 useContext Hook을 사용하여 user

state를 가져옴으로써, props drilling을 방지하고, 코드의 가독성과 유지보수성을 향상시킬 수 있습니다.