**1장. 리액트 입문 2/4: 배열**

**<<11. 배열 렌더링하기>>**

**\* 배열과 컴포넌트 재사용을 이용하여 렌더링**

- UserList.js

import React from 'react';

function User({ user }) { //컴포넌트 재사용

return (

<div>

<b>{user.username}</b> <span>({user.email})</span>

</div>

);

}

function UserList() {

const users = [ //배열 선언

{

id: 1,

username: 'velopert',

email: 'public.velopert@gmail.com'

},

{

id: 2,

username: 'tester',

email: 'tester@example.com'

},

{

id: 3,

username: 'liz',

email: 'liz@example.com'

}

];

return (

<div>

<User user={users[0]} /> //배열 사용

<User user={users[1]} />

<User user={users[2]} />

</div>

);

}

export default UserList;

텍스트이(가) 표시된 사진

자동 생성된 설명

- 배열이 고정적이라면 상관 없겠지만, 배열의 인덱스를 하나하나 조회해가면서 렌더링 하는 방법은 동적인 배열을 렌더링하지 못함

- 동적인 배열을 렌더링해야 할 때는 자바스크립트 배열의 내장함수 map()을 사용함

- **map()함수**는 배열 안에 있는 각 원소를 변환하여 새로운 배열을 만들어줌. 리액트에서 동적인 배열을 렌더링해야 할 때는 이 함수를 사용하여 일반 데이터 배열을 리액트 앨리먼트로 이루어진 배열로 변환해주면 됨

- (필수) 리액트에서 배열을 렌더링 할 때는 key라는 props를 설정해야 함. key값은 각 원소들마다 가지고 있는 고유값으로 설정해야 함. 지금의 경우는 id가 고유값임

- UserList.js

import React from 'react';

function User({ user }) {

return (

<div>

<b>{user.username}</b> <span>({user.email})</span>

</div>

);

}

function UserList() {

const users = [

{

id: 1,

username: 'velopert',

email: 'public.velopert@gmail.com'

},

{

id: 2,

username: 'tester',

email: 'tester@example.com'

},

{

id: 3,

username: 'liz',

email: 'liz@example.com'

}

];

return (

<div>

{users.map(user => (

<User user={user} key={user.id} /> //map() 사용, key 설정

))}

</div>

);

}

export default UserList;

**\* key의 존재 유무에 따른 업데이트 방식**

const array = ['a', 'b', 'c', 'd'];

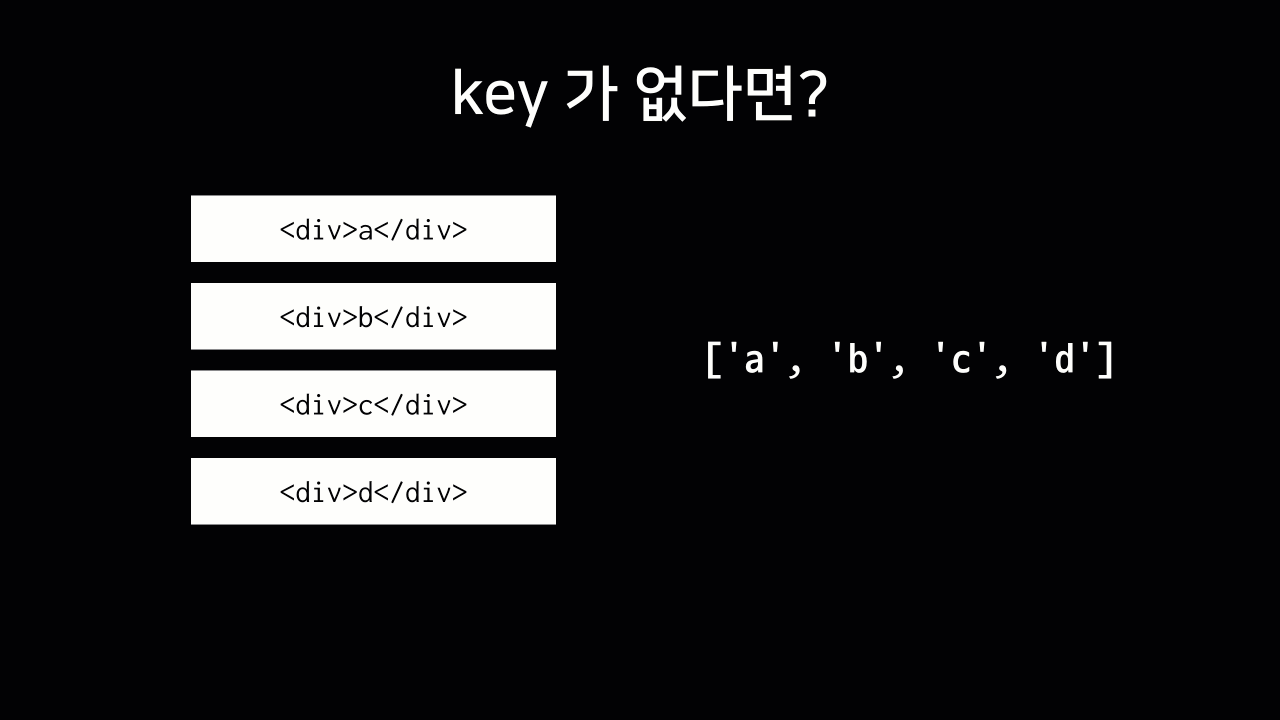
- 해당 배열이 존재한다고 가정

array.map(item => <div>{item}</div>);

- 다음과 같이 렌더링

- 위 배열의 b와 c 사이에 z를 삽입하게 된다면, 리렌더링을 할 때 <div>b</div>와 <div>c</div> 사이에 새 div 태그를 삽입을 하게 되는 것이 아니라 기존의 c가 z로 바뀌고 d는 c로 바뀌고 맨 마지막에는 d가 새로 삽입됨.

- 그 다음에 a 를 제거하게 된다면, 기존의 a 가 b 로 바뀌고, b 는 z 로 바뀌고, z는 c로 바뀌고, c는 d 로바뀌고, 맨 마지막에 있는 d 가 제거됨



- 비효울적임. 하지만 key가 있다면 개선할 수 있음

[

{

id: 0,

text: 'a'

},

{

id: 1,

text: 'b'

},

{

id: 2,

text: 'c'

},

{

id: 3,

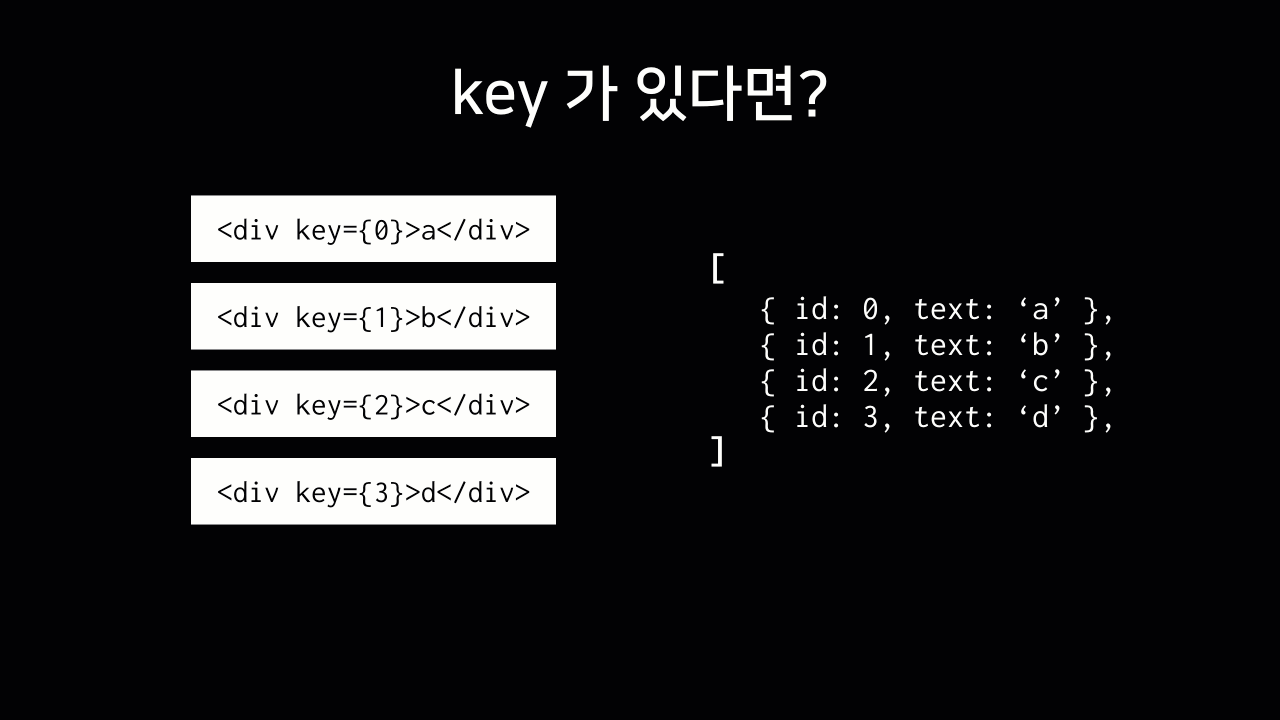
text: 'd'

}

];

array.map(item => <div key={item.id}>{item.text}</div>);

- 배열이 업데이트 도리 때 key가 없을 때처럼 비효율적으로 업데이트 하는 것이 아니라, 수정되지 않는 기존의 값은 그대로 두고 원하는 곳에 내용을 삽입하거나 삭제함



- 때문에, 배열을 렌더링 할 때에는 고유한 key 값이 있는것이 중요하며, 만약에 배열안에 중복되는 key 가 있을 때에는 렌더링시에 오류메시지가 콘솔에 나타나게 되며, 업데이트가 제대로 이루어지지 않게 됨

**<<12. useRef로 컴포넌트 안의 변수 만들기>>**

**\* 컴포넌트에서 특정 DOM을 선택:** ref를 사용

**\* 함수형 컴포넌트에서 이를 설정할 때:** useRef를 사용

**\* useRef Hook**은 DOM을 선택하는 용도 외에도, 다른 용도가 있음. 바로 **컴포넌트 안에서 조회 및 수정할 수 있는 변수를 관리하는 것**

**\* useRef로 관리하는 변수**는 값이 바뀐다고 해서 컴포넌트가 리렌더링되지 않음. 리액트 컴포넌트에서의 상태는 상태를 바꾸는 함수를 호출하고 나서 그 다음 렌더링 이후로 업데이트 된 상태를 조회할 수 있는 반면, useRef로 관리하고 있는 변수는 설정 후 바로 조회할 수 있음

**\* useRef로 관리하는 변수를 사용하여 다음과 같은 값을 관리할 수 있음**

* setTimeout, setInterval 을 통해서 만들어진 id
* 외부 라이브러리를 사용하여 생성된 인스턴스
* scroll 위치

**\* userRef를 사용하여 변수 관리:**

- App 컴포넌트에서 useRef를 사용하여 변수를 관리할 것임

- 용도는 배열에 새 항목을 추가할 때 새 항목에서 사용할 고유 id를 관리

- App.js

import React from 'react';

import UserList from './UserList';

function App() {

const users = [ //배열 선언

{

id: 1,

username: 'velopert',

email: 'public.velopert@gmail.com'

},

{

id: 2,

username: 'tester',

email: 'tester@example.com'

},

{

id: 3,

username: 'liz',

email: 'liz@example.com'

}

];

return <UserList users={users} />;

}

export default App;

**<<13. 배열에 항목 추가하기>>**

\* input 두 개와 button 하나로 이루어진 CreateUser.js 컴포넌트를 src 디렉터리에 만들자

- CreateUser.js

import React from 'react';

function CreateUser({ username, email, onChange, onCreate }) {

return (

<div>

<input

name="username"

placeholder="계정명"

onChange={onChange}

value={username}

/>

<input

name="email"

placeholder="이메일"

onChange={onChange}

value={email}

/>

<button onClick={onCreate}>등록</button>

</div>

);

}

export default CreateUser;

- 상태관리를 CreateUser에서 하지 않고 부모 컴포넌트인 App에서 하게 하고, input의 값 및 이벤트로 등록할 수 있는 함수들을 props로 넘겨 받아서 사용해주겠음

- App.js

import React, { useRef } from 'react';

import UserList from './UserList';

import CreateUser from './CreateUser';

function App() {

const users = [

{

id: 1,

username: 'velopert',

email: 'public.velopert@gmail.com'

},

{

id: 2,

username: 'tester',

email: 'tester@example.com'

},

{

id: 3,

username: 'liz',

email: 'liz@example.com'

}

];

const nextId = useRef(4);

const onCreate = () => {

// 나중에 구현 할 배열에 항목 추가하는 로직

// ...

nextId.current += 1;

};

return (

<>

<CreateUser />

<UserList users={users} />

</>

);

}

export default App;

텍스트이(가) 표시된 사진

자동 생성된 설명

- CreateUser 컴포넌트에게 필요한 props를 App에서 준비

- App.js

import React, { useRef, useState } from 'react';

import UserList from './UserList';

import CreateUser from './CreateUser';

function App() {

const [inputs, setInputs] = useState({

username: '',

email: ''

});

const { username, email } = inputs;

const onChange = e => {

const { name, value } = e.target;

setInputs({

...inputs,

[name]: value

});

};

const users = [

{

id: 1,

username: 'velopert',

email: 'public.velopert@gmail.com'

},

{

id: 2,

username: 'tester',

email: 'tester@example.com'

},

{

id: 3,

username: 'liz',

email: 'liz@example.com'

}

];

const nextId = useRef(4);

const onCreate = () => {

// 나중에 구현 할 배열에 항목 추가하는 로직

// ...

setInputs({

username: '',

email: ''

});

nextId.current += 1;

};

return (

<>

<CreateUser

username={username}

email={email}

onChange={onChange}

onCreate={onCreate}

/>

<UserList users={users} />

</>

);

}

export default App;

-user도 useState를 사용하여 컴포넌트의 상태로서 관리

- App.js

import React, { useRef, useState } from 'react';

import UserList from './UserList';

import CreateUser from './CreateUser';

function App() {

const [inputs, setInputs] = useState({

username: '',

email: ''

});

const { username, email } = inputs;

const onChange = e => {

const { name, value } = e.target;

setInputs({

...inputs,

[name]: value

});

};

const [users, setUsers] = useState([

{

id: 1,

username: 'velopert',

email: 'public.velopert@gmail.com'

},

{

id: 2,

username: 'tester',

email: 'tester@example.com'

},

{

id: 3,

username: 'liz',

email: 'liz@example.com'

}

]);

const nextId = useRef(4);

const onCreate = () => {

// 나중에 구현 할 배열에 항목 추가하는 로직

// ...

setInputs({

username: '',

email: ''

});

nextId.current += 1;

};

return (

<>

<CreateUser

username={username}

email={email}

onChange={onChange}

onCreate={onCreate}

/>

<UserList users={users} />

</>

);

}

export default App;

- 배열에 변화를 줄 때에는 객체와 마찬가지로, 불변성을 지켜주어야 함. 그렇지 때문에 배열의 push, splice, sort 등의 함수를 사용하면 안 됨. 만약 사용해야 한다면 기존의 배열을 한 번 복사하고 나서 사용해야 함

**\* 불변성을 지키면서 배열에 새 항목을 추가하는 방법**

- **1. Spread 연산자 사용**

- App.js

import React, { useRef, useState } from 'react';

import UserList from './UserList';

import CreateUser from './CreateUser';

function App() {

const [inputs, setInputs] = useState({

username: '',

email: ''

});

const { username, email } = inputs;

const onChange = e => {

const { name, value } = e.target;

setInputs({

...inputs,

[name]: value

});

};

const [users, setUsers] = useState([

{

id: 1,

username: 'velopert',

email: 'public.velopert@gmail.com'

},

{

id: 2,

username: 'tester',

email: 'tester@example.com'

},

{

id: 3,

username: 'liz',

email: 'liz@example.com'

}

]);

const nextId = useRef(4);

const onCreate = () => {

const user = {

id: nextId.current,

username,

email

};

setUsers([...users, user]); //spread 사용

setInputs({

username: '',

email: ''

});

nextId.current += 1;

};

return (

<>

<CreateUser

username={username}

email={email}

onChange={onChange}

onCreate={onCreate}

/>

<UserList users={users} />

</>

);

}

export default App;

**- 2. Concat 함수를 사용**

ㄴ concat 함수는 기존의 배열을 수정하지 않고 새로운 원소가 추가된 새로운 배열을 만들어줌

- App.js

import React, { useRef, useState } from 'react';

import UserList from './UserList';

import CreateUser from './CreateUser';

function App() {

const [inputs, setInputs] = useState({

username: '',

email: ''

});

const { username, email } = inputs;

const onChange = e => {

const { name, value } = e.target;

setInputs({

...inputs,

[name]: value

});

};

const [users, setUsers] = useState([

{

id: 1,

username: 'velopert',

email: 'public.velopert@gmail.com'

},

{

id: 2,

username: 'tester',

email: 'tester@example.com'

},

{

id: 3,

username: 'liz',

email: 'liz@example.com'

}

]);

const nextId = useRef(4);

const onCreate = () => {

const user = {

id: nextId.current,

username,

email

};

setUsers(users.concat(user)); //concat 사용

setInputs({

username: '',

email: ''

});

nextId.current += 1;

};

return (

<>

<CreateUser

username={username}

email={email}

onChange={onChange}

onCreate={onCreate}

/>

<UserList users={users} />

</>

);

}

export default App;

**<<14. 배열에 항목 제거하기>>**

- UserList.js에 삭제 버튼 렌더링

import React, { useRef, useState } from 'react';

import UserList from './UserList';

import CreateUser from './CreateUser';

function App() {

const [inputs, setInputs] = useState({

username: '',

email: ''

});

const { username, email } = inputs;

const onChange = e => {

const { name, value } = e.target;

setInputs({

...inputs,

[name]: value

});

};

const [users, setUsers] = useState([

{

id: 1,

username: 'velopert',

email: 'public.velopert@gmail.com'

},

{

id: 2,

username: 'tester',

email: 'tester@example.com'

},

{

id: 3,

username: 'liz',

email: 'liz@example.com'

}

]);

const nextId = useRef(4);

const onCreate = () => {

const user = {

id: nextId.current,

username,

email

};

setUsers(users.concat(user));

setInputs({

username: '',

email: ''

});

nextId.current += 1;

};

return (

<>

<CreateUser

username={username}

email={email}

onChange={onChange}

onCreate={onCreate}

/>

<UserList users={users} />

</>

);

}

export default App;

텍스트이(가) 표시된 사진

자동 생성된 설명

- User 컴포넌트의 삭제 버튼이 클릭될 때 user.id값을 앞으로 props로 받아올 onRemove함수의 파라미터로 넣어서 호출해주어야 함

- onRemove함수는 UserList에서도 전달 받을 것이며, 이를 그대로 User 컴포넌트에게 전달해줄 것임

- onRemove함수 구현

ㄴ 불변성을 지키면서 특정 원소를 배열에서 제거하기 위해서는 filter배열 내장 함수를 사용하는 것이 가장 편함. 이 함수는 배열에서 특정 조건이 만족하는 원소들만 추출하여 새로운 배열을 만들어줌.

- App.js

import React, { useRef, useState } from 'react';

import UserList from './UserList';

import CreateUser from './CreateUser';

function App() {

const [inputs, setInputs] = useState({

username: '',

email: ''

});

const { username, email } = inputs;

const onChange = e => {

const { name, value } = e.target;

setInputs({

...inputs,

[name]: value

});

};

const [users, setUsers] = useState([

{

id: 1,

username: 'velopert',

email: 'public.velopert@gmail.com'

},

{

id: 2,

username: 'tester',

email: 'tester@example.com'

},

{

id: 3,

username: 'liz',

email: 'liz@example.com'

}

]);

const nextId = useRef(4);

const onCreate = () => {

const user = {

id: nextId.current,

username,

email

};

setUsers(users.concat(user));

setInputs({

username: '',

email: ''

});

nextId.current += 1;

};

const onRemove = id => {

// user.id 가 파라미터로 일치하지 않는 원소만 추출해서 새로운 배열을 만듬

// = user.id 가 id 인 것을 제거함

setUsers(users.filter(user => user.id !== id));

};

return (

<>

<CreateUser

username={username}

email={email}

onChange={onChange}

onCreate={onCreate}

/>

<UserList users={users} onRemove={onRemove} />

</>

);

}

export default App;

텍스트이(가) 표시된 사진

자동 생성된 설명

**<<15. 배열에 항목 수정하기>>**

**\* User 컴포넌트에 계정명을 클릭했을 때 색상이 초록색으로 바뀌고 다시 누르면 검정색으로 바뀌도록 구현**

- App 컴포넌트의 users 배열 안에 active 속성 추가

- App.js

import React, { useRef, useState } from 'react';

import UserList from './UserList';

import CreateUser from './CreateUser';

function App() {

const [inputs, setInputs] = useState({

username: '',

email: ''

});

const { username, email } = inputs;

const onChange = e => {

const { name, value } = e.target;

setInputs({

...inputs,

[name]: value

});

};

const [users, setUsers] = useState([

{

id: 1,

username: 'velopert',

email: 'public.velopert@gmail.com',

active: true //active 속성 추가

},

{

id: 2,

username: 'tester',

email: 'tester@example.com',

active: false

},

{

id: 3,

username: 'liz',

email: 'liz@example.com',

active: false

}

]);

const nextId = useRef(4);

const onCreate = () => {

const user = {

id: nextId.current,

username,

email

};

setUsers(users.concat(user));

setInputs({

username: '',

email: ''

});

nextId.current += 1;

};

const onRemove = id => {

// user.id 가 파라미터로 일치하지 않는 원소만 추출해서 새로운 배열을 만듬

// = user.id 가 id 인 것을 제거함

setUsers(users.filter(user => user.id !== id));

};

return (

<>

<CreateUser

username={username}

email={email}

onChange={onChange}

onCreate={onCreate}

/>

<UserList users={users} onRemove={onRemove} onToggle={onToggle} />

</>

);

}

export default App;

**\* 추가적으로 cursor필드를 설정하여 마우스를 올렸을 때 커서가 손가락 모양으로 변하도록 수정**

- UserList.js

import React from 'react';

function User({ user, onRemove }) {

return (

<div>

<b

style={{

cursor: 'pointer',

color: user.active ? 'green' : 'black'

}}

>

{user.username}

</b>

<span>({user.email})</span>

<button onClick={() => onRemove(user.id)}>삭제</button>

</div>

);

}

function UserList({ users, onRemove }) {

return (

<div>

{users.map(user => (

<User user={user} key={user.id} onRemove={onRemove} />

))}

</div>

);

}

export default UserList;

텍스트이(가) 표시된 사진

자동 생성된 설명

**\* onToggle:**

- id값을 비교해서 id가 다르다면 그대로 두고, 같다면 active 값을 반전시키도록 구현

- App.js

import React, { useRef, useState } from 'react';

import UserList from './UserList';

import CreateUser from './CreateUser';

function App() {

const [inputs, setInputs] = useState({

username: '',

email: ''

});

const { username, email } = inputs;

const onChange = e => {

const { name, value } = e.target;

setInputs({

...inputs,

[name]: value

});

};

const [users, setUsers] = useState([

{

id: 1,

username: 'velopert',

email: 'public.velopert@gmail.com',

active: true

},

{

id: 2,

username: 'tester',

email: 'tester@example.com',

active: false

},

{

id: 3,

username: 'liz',

email: 'liz@example.com',

active: false

}

]);

const nextId = useRef(4);

const onCreate = () => {

const user = {

id: nextId.current,

username,

email

};

setUsers(users.concat(user));

setInputs({

username: '',

email: ''

});

nextId.current += 1;

};

const onRemove = id => {

// user.id 가 파라미터로 일치하지 않는 원소만 추출해서 새로운 배열을 만듬

// = user.id 가 id 인 것을 제거함

setUsers(users.filter(user => user.id !== id));

};

const onToggle = id => {

setUsers(

users.map(user =>

user.id === id ? { ...user, active: !user.active } : user

)

);

};

return (

<>

<CreateUser

username={username}

email={email}

onChange={onChange}

onCreate={onCreate}

/>

<UserList users={users} onRemove={onRemove} onToggle={onToggle} />

</>

);

}

export default App;

- UserList 컴포넌트에서 onToggle를 받아와서 User에게 전달해주고 onRemove를 구현했던 것처럼 onToggle에 id를 넣어서 호출

- UserList.js

import React from 'react';

function User({ user, onRemove, onToggle }) {

return (

<div>

<b

style={{

cursor: 'pointer',

color: user.active ? 'green' : 'black'

}}

onClick={() => onToggle(user.id)}

>

{user.username}

</b>

&nbsp;

<span>({user.email})</span>

<button onClick={() => onRemove(user.id)}>삭제</button>

</div>

);

}

function UserList({ users, onRemove, onToggle }) {

return (

<div>

{users.map(user => (

<User

user={user}

key={user.id}

onRemove={onRemove}

onToggle={onToggle}

/>

))}

</div>

);

}

export default UserList;

텍스트이(가) 표시된 사진

자동 생성된 설명