17. useMemo 를 사용하여 연산한 값 재사용하기

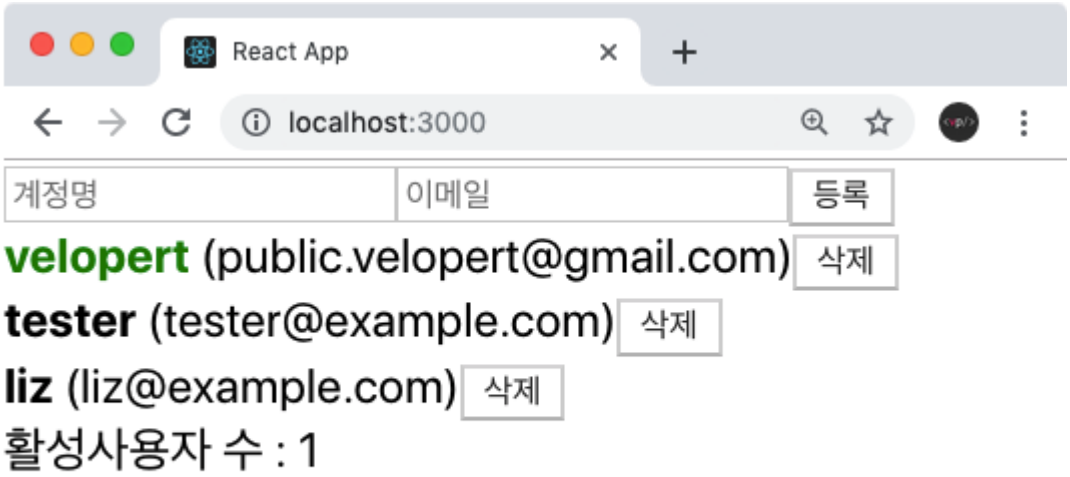
이번에는 성능 최적화를 위하여 연산된 값을 `useMemo` 라는 Hook 을 사용하여 재사용하는 방법을 알아보도록 하겠습니다.
App 컴포넌트에서 다음과 같이 `countActiveUsers` 라는 함수를 만들어서, `active` 값이 `true` 인 사용자의 수를 세어서 화면에 렌더링을 해보세요.

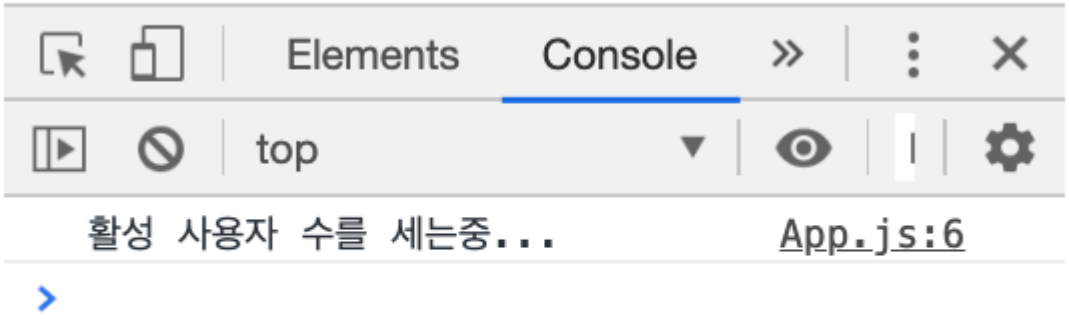
App.js

```
import React, { useRef, useState } from 'react';
import UserList from './UserList';
import CreateUser from './CreateUser';

function countActiveUsers(users) {
  console.log('활성 사용자 수를 세는중...');
  return users.filter(user => user.active).length;
}

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
    }
  ]);
```

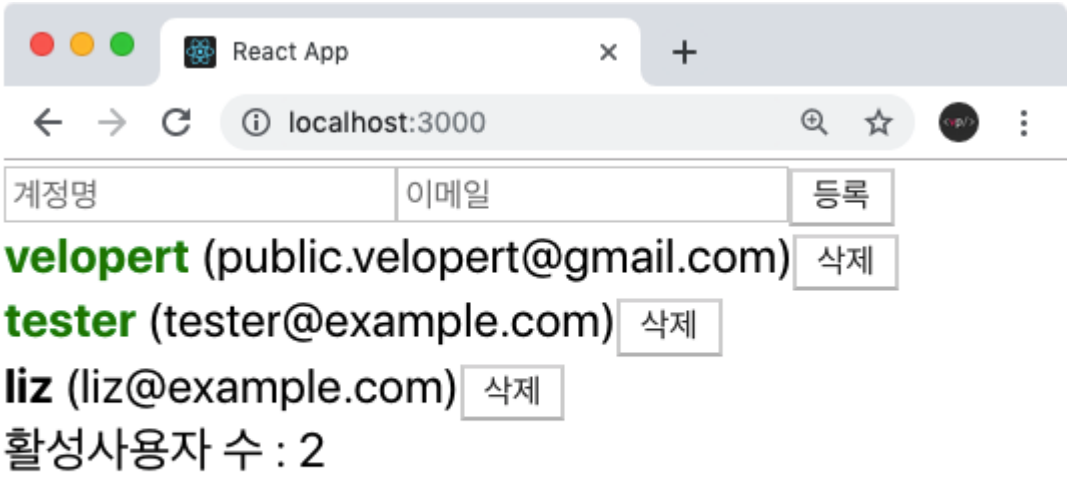
```
  const nextId = useRef(4);
  const onCreate = () => {
    const user = {
      id: nextId.current,
      username,
      email
    };
    setUsers(users.concat(user));

    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
      )
    );
  };
  const count = countActiveUsers(users);
  return (
    <>
      <CreateUser
        username={username}
        email={email}
        onChange={onChange}
        onCreate={onCreate}
      />
      <UserList users={users} onRemove={onRemove} onToggle={onToggle} />
      <div>활성사용자 수 : {count}</div>
    </>
  );
}

export default App;
```

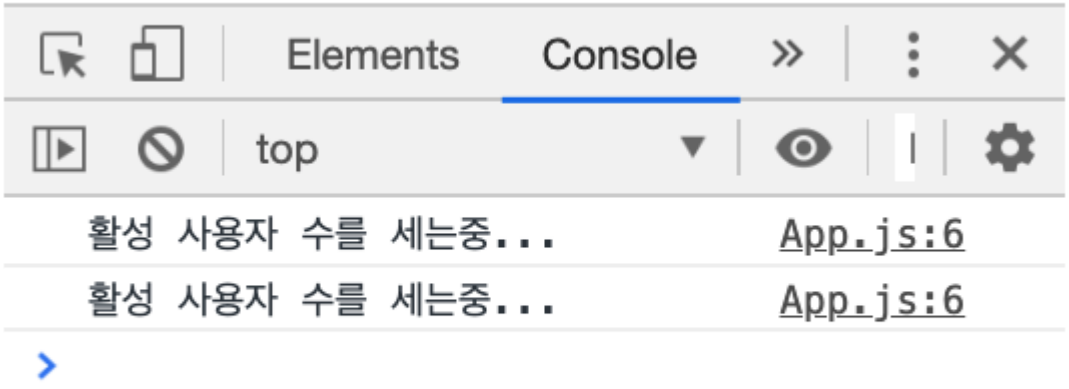
`countActiveUsers` 함수에서 콘솔에 메시지를 출력하도록 한 이유는, 이 함수가 호출될때마다 우리가 알수있게 하기 위함입니다.
구현을 마치면 다음과 같이 나타날텐데요.



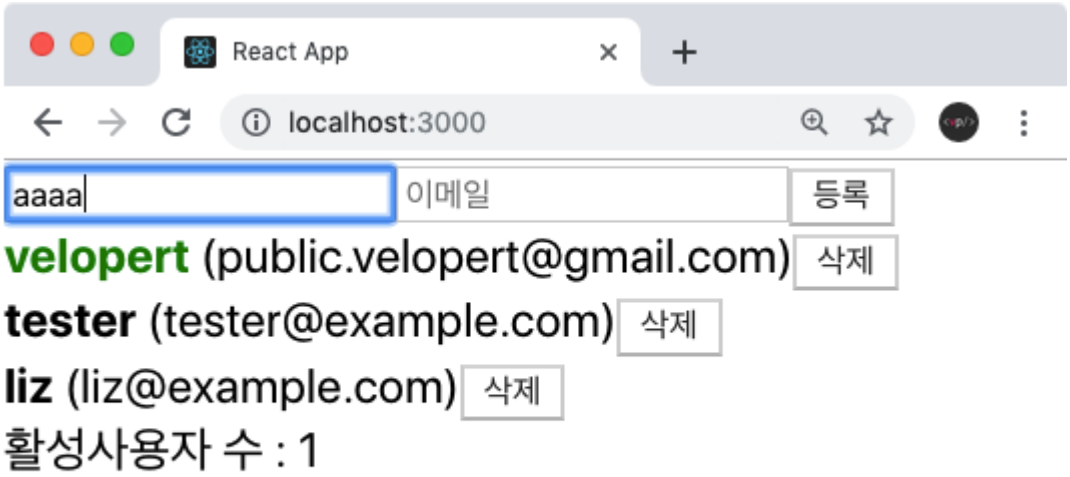


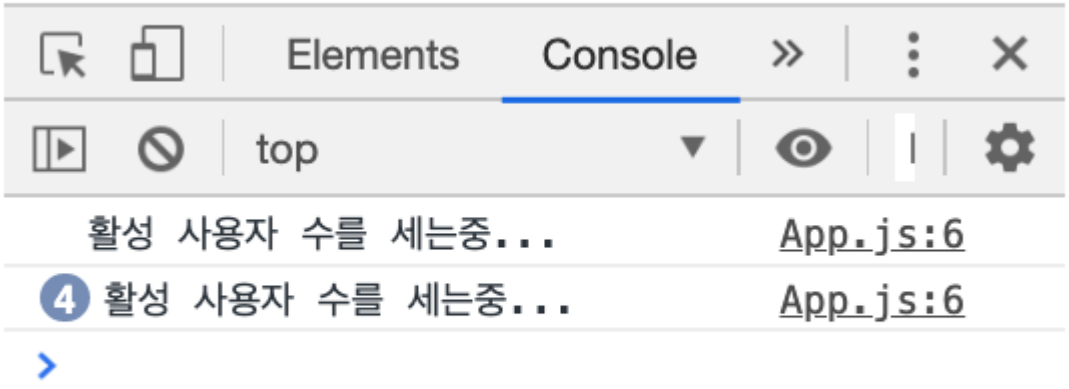
다른 계정명을 눌러서 초록색으로 만들면 활성 사용자 수 또한 업데이트 될 것입니다.





그런데, 여기서 발생하는 성능적 문제가 한가지 있습니다. 바로, input 의 값을 바꿀때에도 `countActiveUsers` 함수가 호출된다는 것 입니다.





활성 사용자 수를 세는건, users 에 변화가 있을때만 세야되는건데, input 값이 바뀔 때에도 컴포넌트가 리렌더링 되므로 이렇게 불필요할때에도 호출하여서 자원이 낭비되고 있습니다.

이러한 상황에는 `useMemo` 라는 Hook 함수를 사용하면 성능을 최적화 할 수 있습니다.
Memo 는 "memoized" 를 의미하는데, 이는, 이전에 계산 한 값을 재사용한다는 의미를 가지고 있습니다.

한번 사용해볼까요?

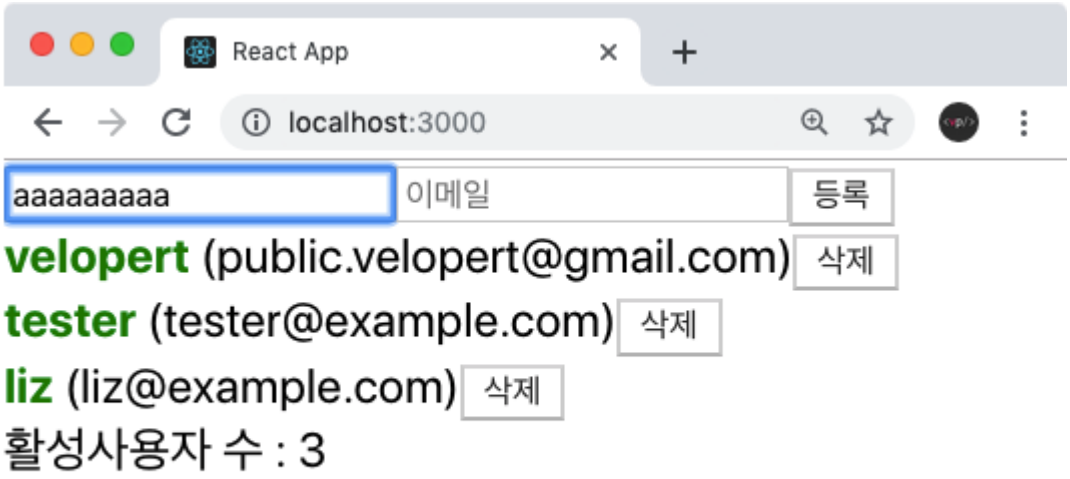
App.js

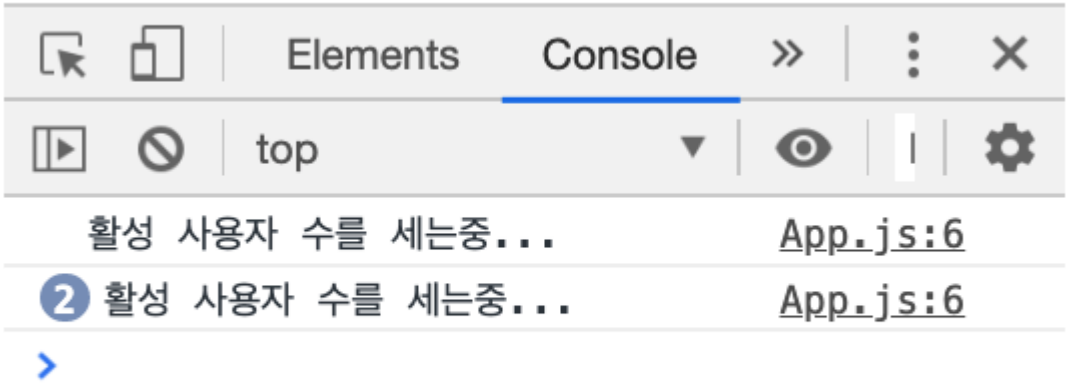
```
import React, { useRef, useState, useMemo } from 'react';
import UserList from './UserList';
```

```
import CreateUser from './CreateUser';

function countActiveUsers(users) {
  console.log('활성 사용자 수를 세는중...');
  return users.filter(user => user.active).length;
}

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
      )
    );
  };
  const count = useMemo(() => countActiveUsers(users), [users]);
  return (
    <>
      <CreateUser
        username={username}
        email={email}
        onChange={onChange}
        onCreate={onCreate}
      />
      <UserList users={users} onRemove={onRemove} onToggle={onToggle} />
      <div>활성사용자 수 : {count}</div>
    </>
  );
}

export default App;
```

`useMemo` 의 첫번째 파라미터에는 어떻게 연산할지 정의하는 함수를 넣어주면 되고 두번째 파라미터에는 deps 배열을 넣어주면 되는데, 이 배열 안에 넣은 내용이 바뀌면, 우리가 등록한 함수를 호출해서 값을 연산해주고, 만약에 내용이 바뀌지 않았다면 이전에 연산한 값을 재사용하게 됩니다.
한번 계정명들을 클릭도 해보고, input 을 수정도 해보세요.





성능

최적화가 잘 이루어졌나요?