16. useEffect 를 사용하여 마운트/언마운트/업데이트시 할 작업 설정하기

이번에는 useEffect 라는 Hook 을 사용하여 컴포넌트가 마운트 됐을 때 (처음 나타났을 때), 언마운트 됐을 때 (사라질 때), 그리고 업데이트 될 때 (특정 props 가 바뀔 때) 특정 작업을 처리하는 방법에 대해서 알아보겠습니다.

마운트 / 언마운트

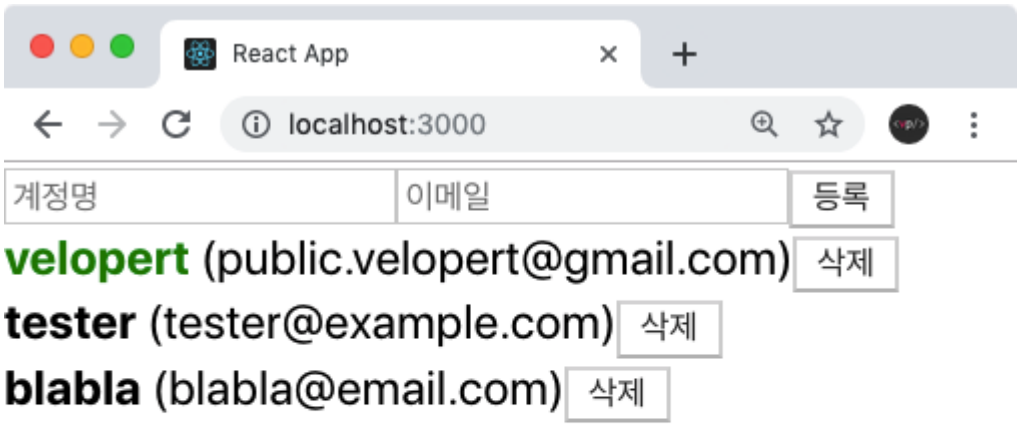
우선, 마운트/언마운트를 관리해보겠습니다.

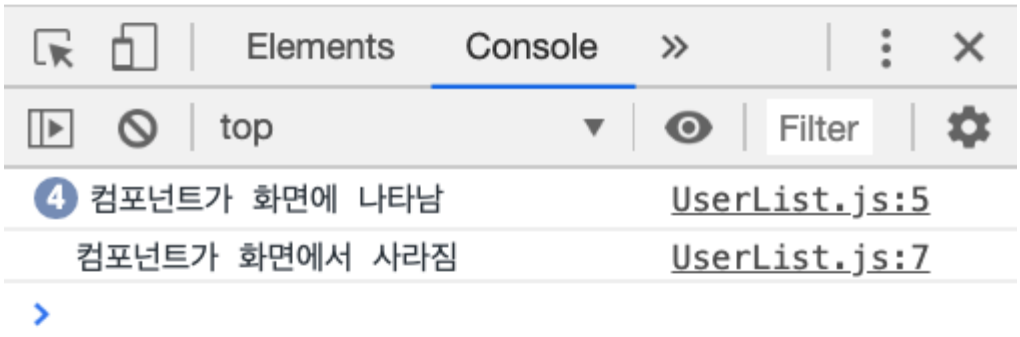
UserList.js

```
import React, { useEffect } from 'react';

function User({ user, onRemove, onToggle }) {
  useEffect(() => {
    console.log('컴포넌트가 화면에 나타남');
    return () => {
      console.log('컴포넌트가 화면에서 사라짐');
    };
  }, []);
  return (
    <div>
      <b
        style={{
          cursor: 'pointer',
          color: user.active ? 'green' : 'black'
        }}
        onClick={() => onToggle(user.id)}
      >
        {user.username}
      </b>
       
      <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;
```

useEffect 를 사용 할 때에는 첫번째 파라미터에는 함수, 두번째 파라미터에는 의존값이 들어있는 배열 (deps)을 넣습니다. 만약에 deps 배열을 비우게 된다면, 컴포넌트가 처음 나타날때에만 useEffect 에 등록한 함수가 호출됩니다.
그리고, useEffect 에서는 함수를 반환 할 수 있는데 이를 cleanup 함수라고 부릅니다. cleanup 함수는 useEffect 에 대한 뒷정리를 해준다고 이해하시면 되는데요, deps 가 비어있는 경우에는 컴포넌트가 사라질 때 cleanup 함수가 호출됩니다.
코드를 작성하고 나서 콘솔을 확인해보고, 새로운 항목을 추가도 해보고 제거도 해보세요.





주로, 마운트 시에 하는 작업들은 다음과 같은 사항들이 있습니다.

- `props` 로 받은 값을 컴포넌트의 로컬 상태로 설정
- 외부 API 요청 (REST API 등)
- 라이브러리 사용 (D3, Video.js 등...)
- setInterval 을 통한 반복작업 혹은 setTimeout 을 통한 작업 예약

그리고 언마운트 시에 하는 작업들은 다음과 같은 사항이 있습니다.

- setInterval, setTimeout 을 사용하여 등록한 작업들 clear 하기 (clearInterval, clearTimeout)
- 라이브러리 인스턴스 제거

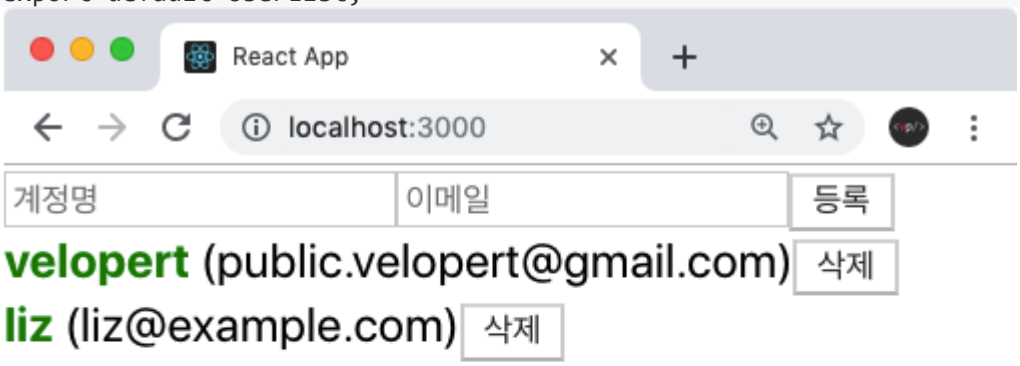
deps 에 특정 값 넣기

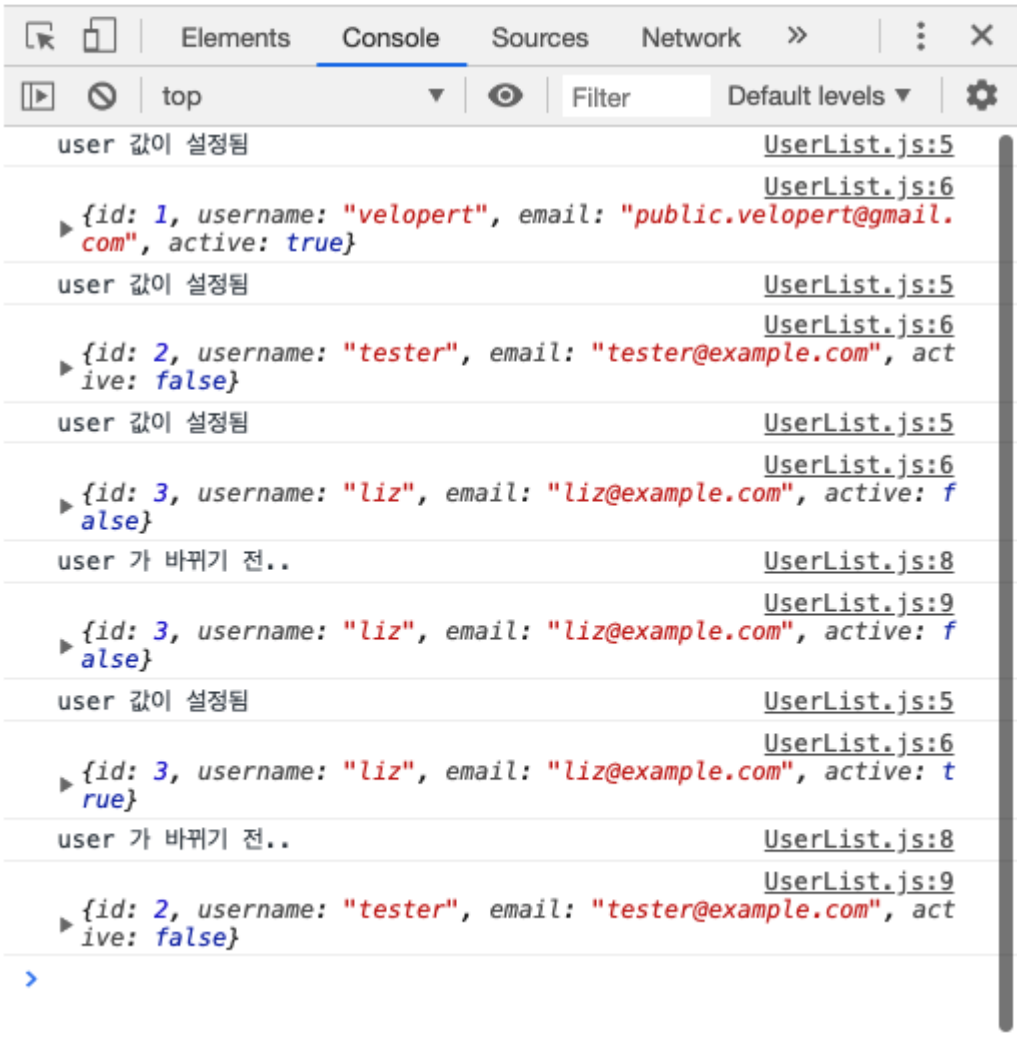
이번에는 deps 에 특정 값을 넣어보도록 하겠습니다. deps 에 특정 값을 넣게 된다면, 컴포넌트가 처음 마운트 될 때에도 호출이 되고, 지정한 값이 바뀔 때에도 호출이 됩니다. 그리고, deps 안에 특정 값이 있다면 언마운트시에도 호출이되고, 값이 바뀌기 직전에도 호출이 됩니다.

한번 코드를 이렇게 작성해보세요.

```
import React, { useEffect } from 'react';

function User({ user, onRemove, onToggle }) {
  useEffect(() => {
    console.log('user 값이 설정됨');
    console.log(user);
    return () => {
      console.log('user 가 바뀌기 전..');
      console.log(user);
    };
  }, [user]);
  return (
    <div>
      <b
        style={{
          cursor: 'pointer',
          color: user.active ? 'green' : 'black'
        }}
        onClick={() => onToggle(user.id)}
      >
        {user.username}
      </b>
       
      <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;
```



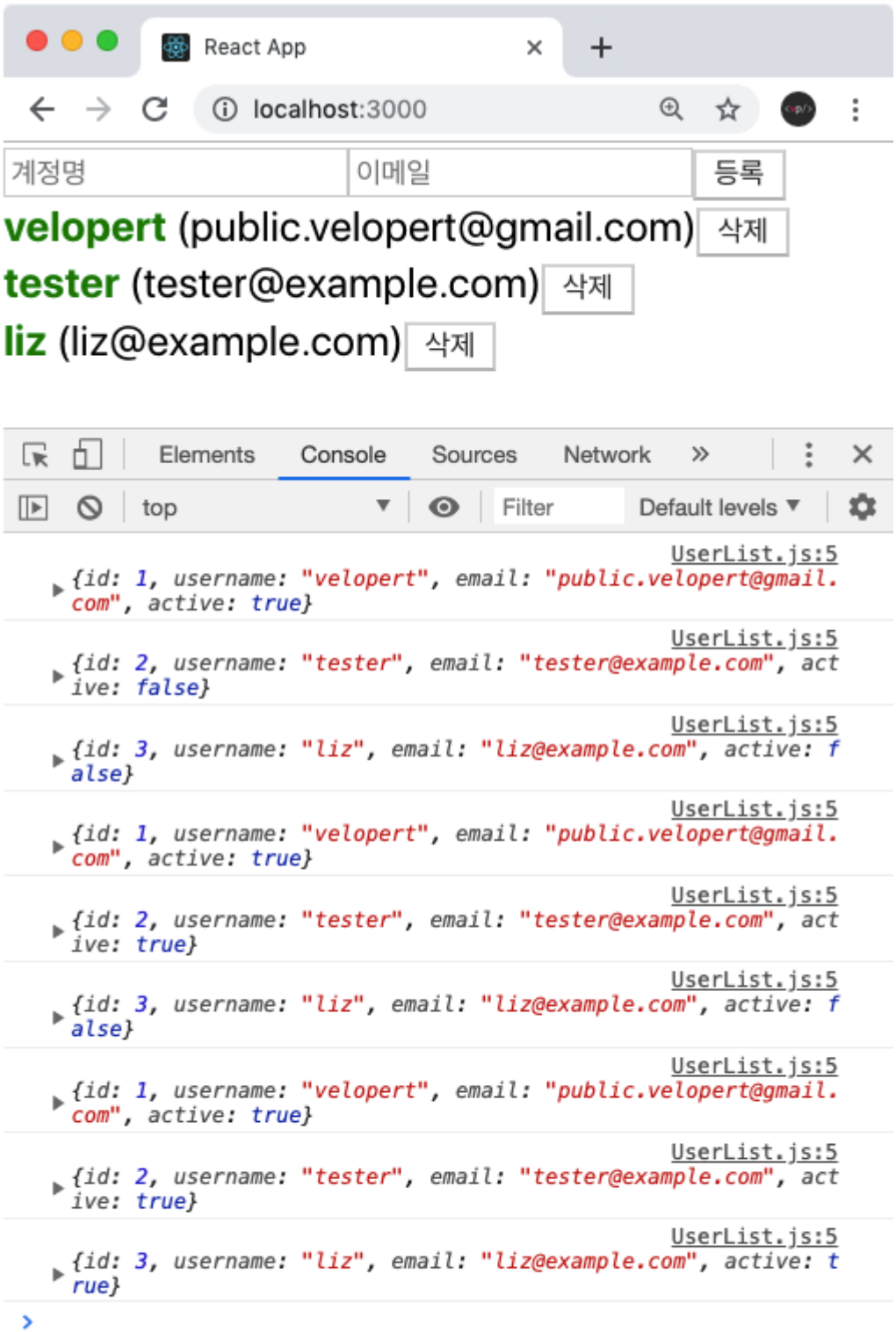


`useEffect` 안에서 사용하는 상태나, props 가 있다면, `useEffect` 의 `deps` 에 넣어주어야 합니다. 그렇게 하는게, 규칙입니다.
만약 `useEffect` 안에서 사용하는 상태나 props 를 `deps` 에 넣지 않게 된다면 `useEffect` 에 등록한 함수가 실행 될 때 최신 props / 상태를 가르키지 않게 됩니다.

deps 파라미터를 생략하기

`deps` 파라미터를 생략한다면, 컴포넌트가 리렌더링 될 때마다 호출이 됩니다. 한번 다음과 같이 코드를 작성해보세요.

```
import React, { useEffect } from 'react';

function User({ user, onRemove, onToggle }) {
  useEffect(() => {
    console.log(user);
  });
  return (
    <div>
      <b
        style={{
          cursor: 'pointer',
          color: user.active ? 'green' : 'black'
        }}
        onClick={() => onToggle(user.id)}
      >
        {user.username}
      </b>
       
      <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;
```



참고로 리액트 컴포넌트는 기본적으로 부모컴포넌트가 리렌더링되면 자식 컴포넌트 또한 리렌더링이 됩니다. 바뀐 내용이 없다 할지라도요.

물론, 실제 DOM 에 변화가 반영되는 것은 바뀐 내용이 있는 컴포넌트에만 해당합니다. 하지만, Virtual DOM 에는 모든걸 다 렌더링하고 있다는 겁니다.

나중에는, 컴포넌트를 최적화 하는 과정에서 기존의 내용을 그대로 사용하면서 Virtual DOM 에 렌더링 하는 리소스를 아낄 수도 있습니다. 이것은 다음번에 알아볼게요.