11. 배열 렌더링하기

이번에는 리액트에서 배열을 렌더링하는 방법을 알아보겠습니다.

이러 배열이 있다고 가정해봅시다.

```
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'
  }
];
```

만약에 이 내용을 컴포넌트로 렌더링한다면 어떻게 해야 할까요?

일단, 가장 기본적인 방법으론 비효율적이지만, 그냥 그대로 코드를 작성하는 것 입니다.

src 디렉터리에 UserList.js 컴포넌트를 다음과 같이 만들어보세요.

UserList.js

```
import React from 'react';

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>
      <div>
```

```
        <b>{users[0].username}</b> <span>({users[0].email})</span>
      </div>
      <div>
        <b>{users[1].username}</b> <span>({users[1].email})</span>
      </div>
      <div>
        <b>{users[2].username}</b> <span>({users[1].email})</span>
      </div>
    </div>
  );
}

export default UserList;
```

그런데, 재사용되는 코드를 일일히 넣는게 별로 좋지 않으니, 컴포넌트를 재사용 할 수 있도록 새로 만들어주겠습니다.

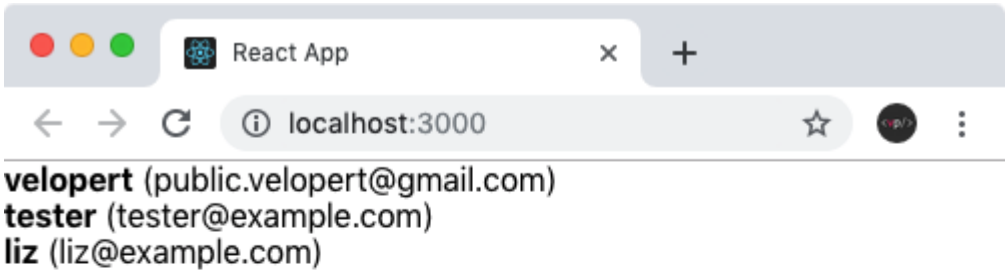
참고로, 한 파일에 여러개의 컴포넌트를 선언해도 괜찮습니다.

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>
      <User user={users[0]} />
      <User user={users[1]} />
      <User user={users[2]} />
    </div>
  );
```

```
}

export default UserList;
```

한번 컴포넌트를 App 에서 렌더링 해봅시다.



배열이 고정적이라면 상관없겟지만, 배열의 인덱스를 하나하나 조회해가면서 렌더링하는 방법은 동적인 배열을 렌더링하지 못합니다.

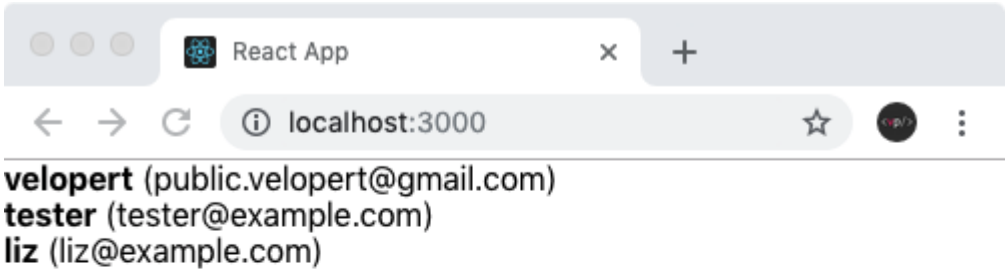
동적인 배열을 렌더링해야 할 때에는 자바스크립트 배열의 내장함수 `map()` 을 사용합니다. 이 함수를 잘 모르신다면 이 링크 를 참고하세요.
`map()` 함수는 배열안에 있는 각 원소를 변환하여 새로운 배열을 만들어줍니다. 리액트에서 동적인 배열을 렌더링해야 할 때는 이 함수를 사용하여 일반 데이터 배열을 리액트 엘리먼트로 이루어진 배열로 변환해주면 됩니다.
UserList 컴포넌트를 다음과 같이 수정해보세요.

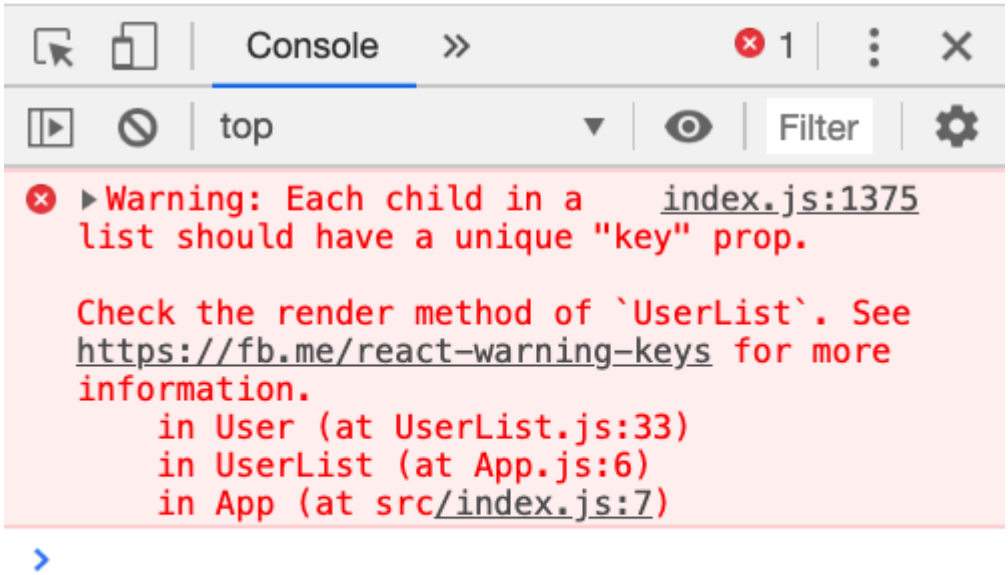
UserList.js

```
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} />
      ))}
    </div>
  );
}

export default UserList;
```

간단하지요? 이렇게 하면 배열의 모든 원소가 렌더링됩니다. 하지만, 여기서 끝이 아닙니다. 브라우저에서 콘솔을 열어보면 다음과 같은 에러가 보여질것입니다.





리액트에서 배열을 렌더링 할 때에는 `key` 라는 props 를 설정해야합니다. `key` 값은 각 원소들마다 가지고 있는 고유값으로 설정을 해야합니다. 지금의 경우엔 `id` 가 고유 값이지요.

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} />
      ))}
    </div>
  );
}

export default UserList;
```

만약 배열 안의 원소가 가지고 있는 고유한 값이 없다면 `map()` 함수를 사용 할 때 설정하는 콜백함수의 두번째 파라미터 `index` 를 `key` 로 사용하시면 됩니다.

```
<div>
  {users.map((user, index) => (
    <User user={user} key={index} />
  ))}
</div>
```

만약에 배열을 렌더링 할 때 `key` 설정을 하지 않게된다면 기본적으로 배열의 `index` 값을 `key` 로 사용하게 되고, 아까 봤었던 경고메시지가 뜨게 됩니다. 이렇게 경고 메시지가 뜨는 이유는, 각 고유 원소에 `key` 가 있어야만 배열이 업데이트 될 때 효율적으로 렌더링 될 수 있기 때문입니다.

key 의 존재유무에 따른 업데이트 방식

예를 들어서 다음과 같은 배열이 있다고 가정해봅시다.

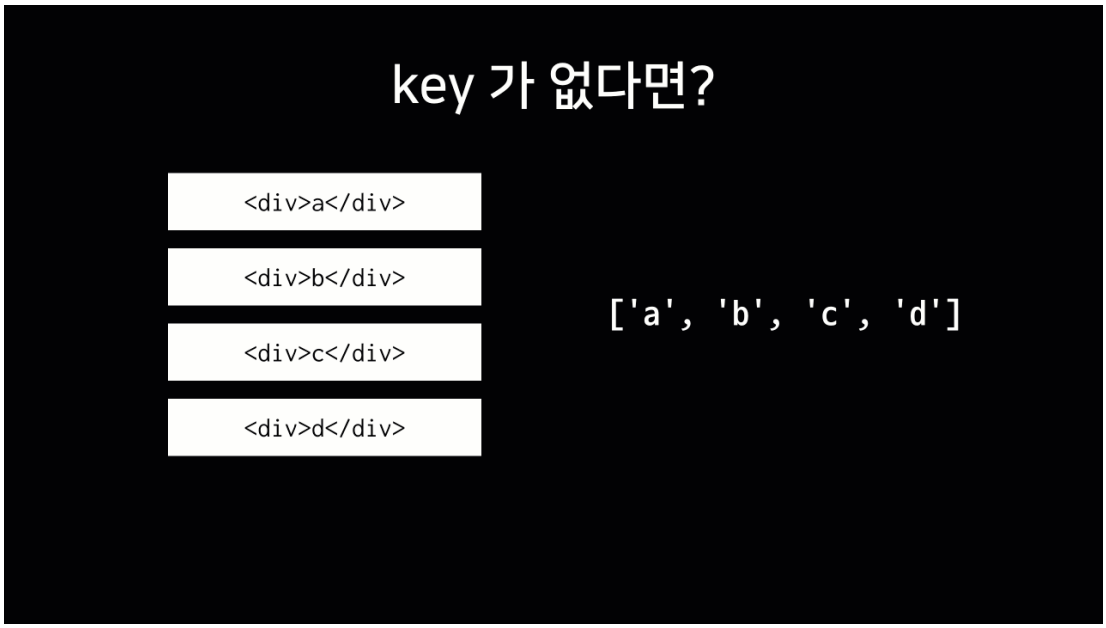
```
const array = ['a', 'b', 'c', 'd'];
```

그리고 위 배열을 다음과 같이 렌더링한다고 가정해보겠습니다.

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

위 배열의 b 와 c 사이에 z 를 삽입하게 된다면, 리렌더링을 하게 될 때 `<div>b</div>` 와 `<div>c</div>` 사이에 새 `div` 태그를 삽입을 하게 되는 것이

아니라, 기존의 c 가 z 로바뀌고, d 는 c 로 바뀌고, 맨 마지막에 d 가 새로 삽입됩니다.
그 다음에 a 를 제거하게 된다면, 기존의 a 가 b 로 바뀌고, b 는 z 로 바뀌고, z 는 c 로 바뀌고, c 는 d 로바뀌고, 맨 마지막에 있는 d 가 제거됩니다.



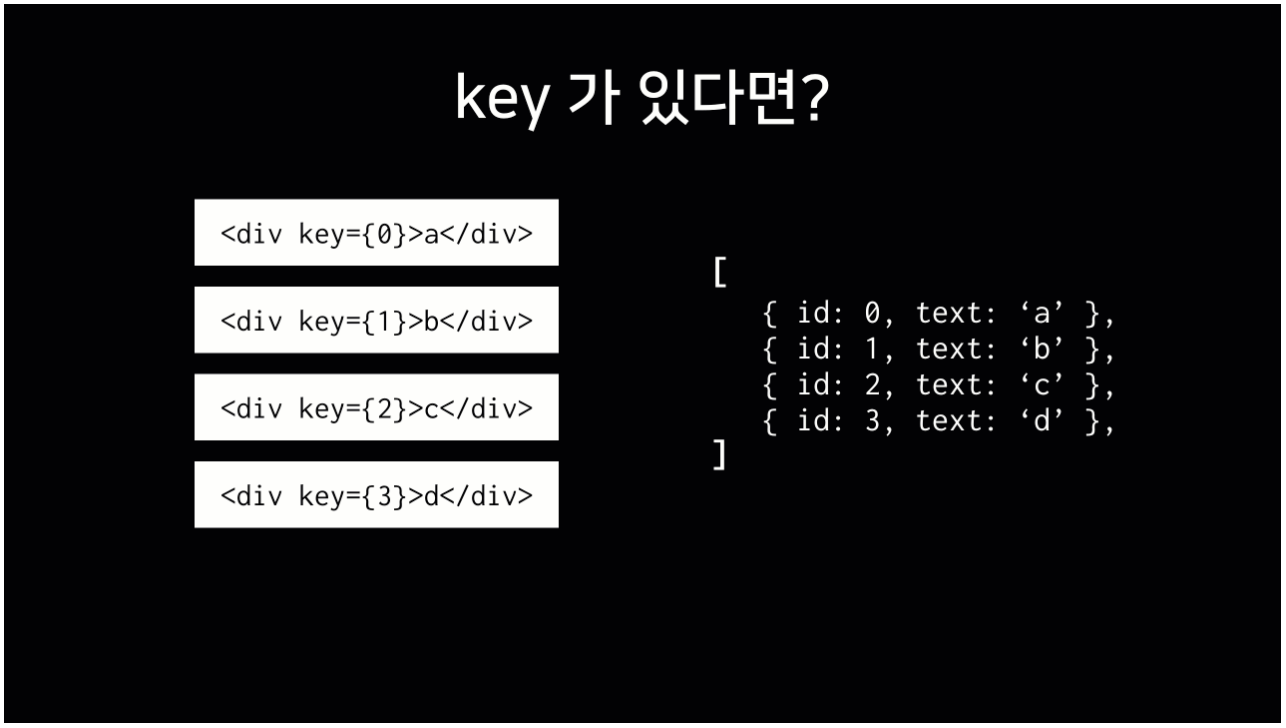
비효율적이지요? 하지만, `key` 가 있다면 이 작업은 개선됩니다.
객체에 다음과 같이 `key` 로 사용 할 수 있는 고유 값이 있고

```
[
  {
    id: 0,
    text: 'a'
  },
  {
    id: 1,
    text: 'b'
  },
  {
    id: 2,
    text: 'c'
  },
  {
    id: 3,
    text: 'd'
  }
];
```

다음과 같이 렌더링이 된다면

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

배열이 업데이트 될 때 `key` 가 없을 때 처럼 비효율적으로 업데이트 하는 것이 아니라, 수정되지 않는 기존의 값은 그대로 두고 원하는 곳에 내용을 삽입하거나 삭제합니다.



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 됩니다.