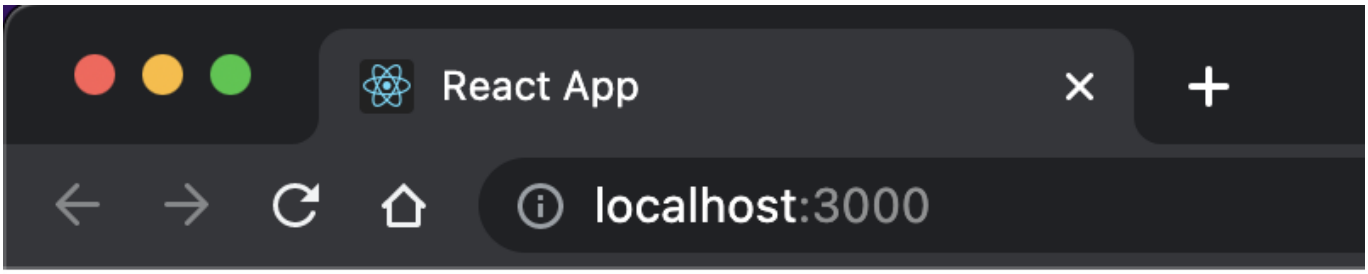
유다현 노트정리

22.05.09 ~

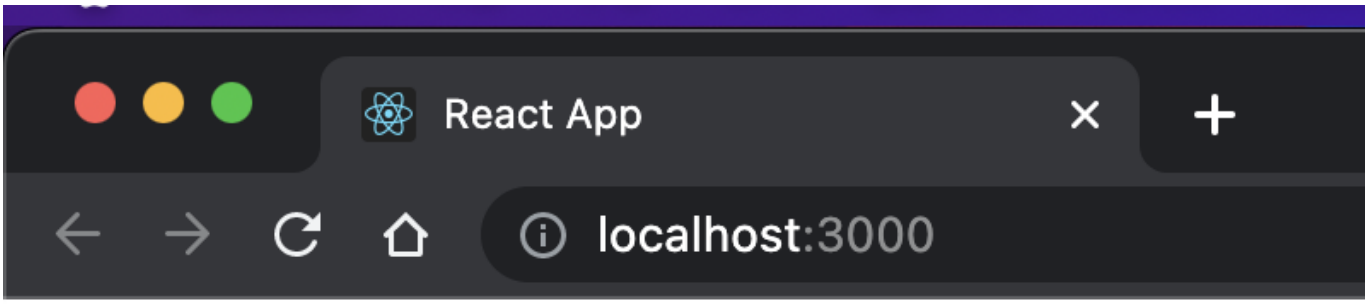
01. useRef

리액트 프로젝트 내부에서 Dom을 직접적으로 건드려야 할 때 ref로 이름을 달 수 있다. id를 사용하는건 권장되지 않는다.

```
import React from "react";
import "./style.css";
const MyRef = () => {
  const input = React.useRef();
  const [mypassword, setpassword] = React.useState();
  const [click, setclick] = React.useState(false);
  const [validate, setvalidate] = React.useState(false);

  const handleChange = (e) => {
    setpassword(e.target.value);
  };

  const handleButtonClick = () => {
    setclick(true);
    setvalidate(mypassword === "0000");
  };

  return (
    <div>
      <input
        type="text"
        ref={input}
        value={mypassword}
        onChange={handleChange}
        className={click ? (validate ? "success" : "failure") : ""}
        placeholder="비밀번호를 입력해주세요."
      />
      <button type="button" onClick={handleButtonClick}>
        확인
      </button>
    </div>
  );
};

export default MyRef;
```



원래 비밀번호라면 type="password"로 하는것이 맞는데 그러면 입력값이 안보여서 text로 변경하였다.



02. 컴포넌트에 ref 달기

컴포넌트 내부에 있는 DOM을 컴포넌트 외부에서 사용한다. 컴포넌트에 ref 를 다는 방법은 DOM에 쓰는 방법과 같다.

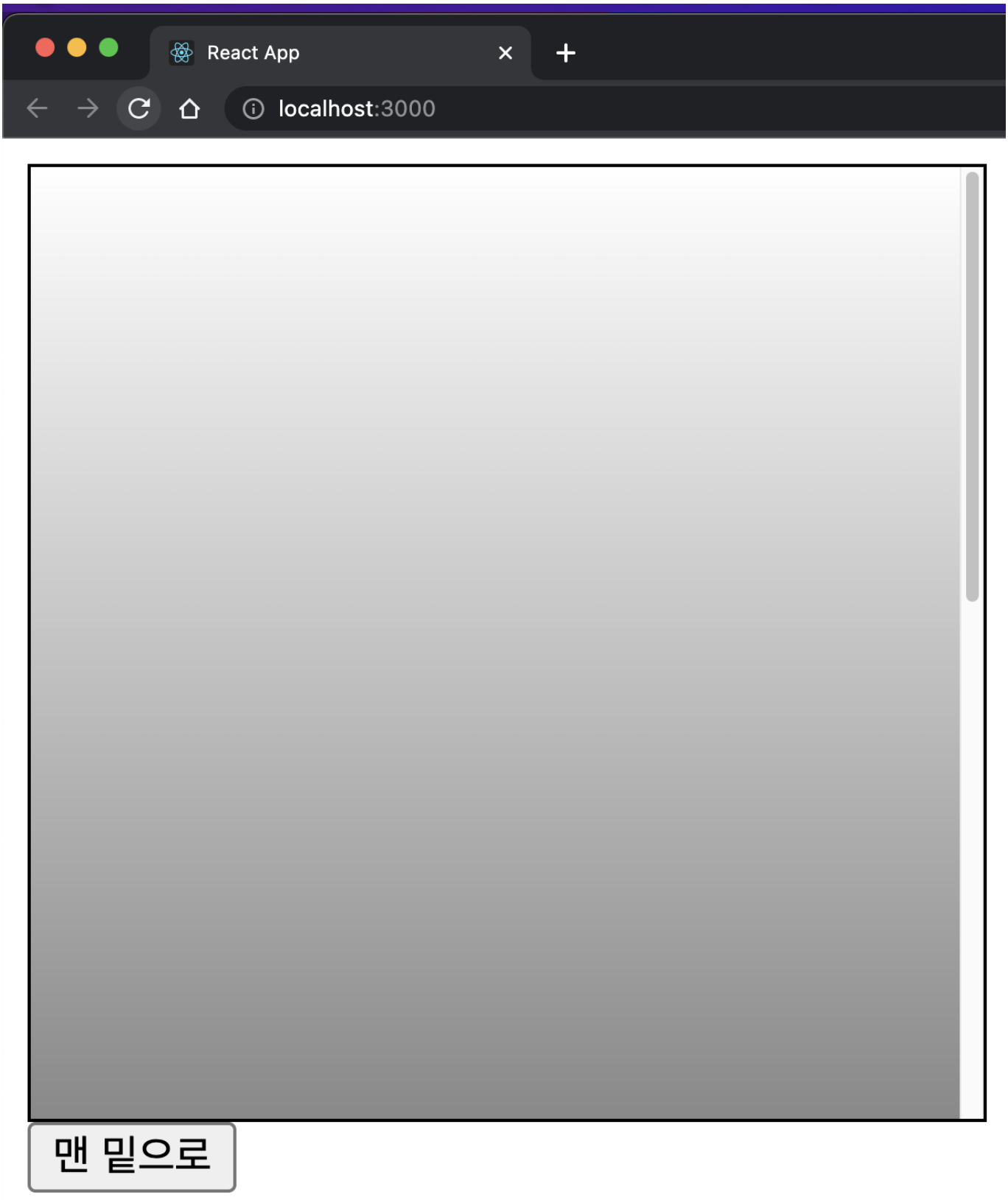
React 컴포넌트에 ref prop을 넘길 수 있는 forwordRef 함수 사용법

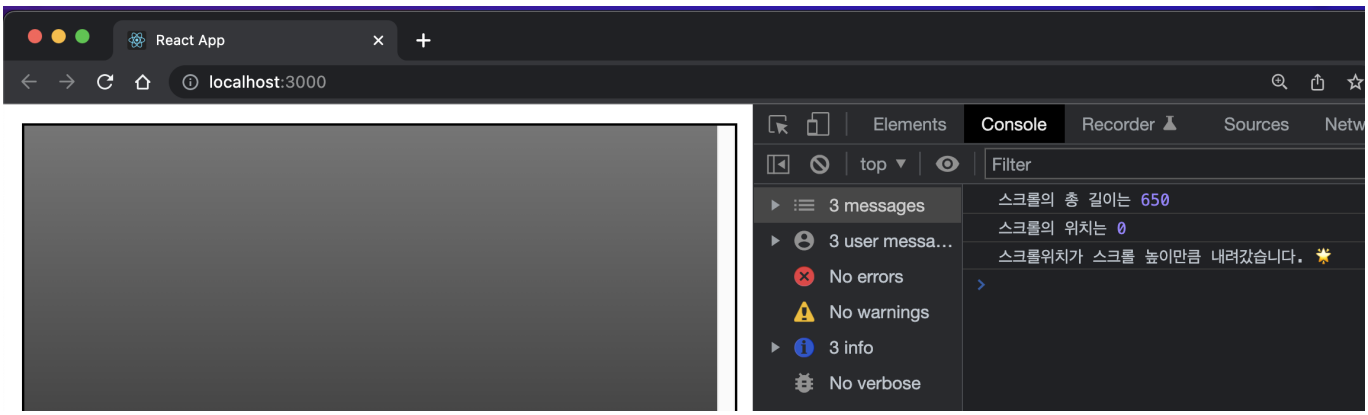
03. forwordRef 사용법

```
import React from "react";
import ScrollBox from "./ScrollBox";
function App() {

  const scrollBoxRef = React.useRef();
  const scrollBottom = () => {
    console.log("스크롤의 총 길이는",scrollBoxRef.current.scrollHeight);
    console.log("스크롤의 위치는",scrollBoxRef.current.scrollTop);
    scrollBoxRef.current.scrollTop=scrollBoxRef.current.scrollHeight;
    console.log("스크롤위치가 스크롤 높이만큼 내려갔습니다. 🌟")
```

```
  };
  return (
    <div>
      <ScrollBox ref={scrollBoxRef} />
      <button onClick={scrollBottom}>맨 밑으로</button>
    </div>
   );
}



export default App;
```

```
import React from "react";

const ScrollBox = React.forwardRef((props, ref) => {
  const style = {
    border: "1px solid black",
    height: "300px",
    width: "300px",
    overflow: "auto",
    position: "relative",
  };
  const innerStyle = {
    width: "100%",
    height: "650px",
    background: "linear-gradient(#fff,#000)",
  };

  return (
    <>
      <div ref={ref} style={style}>
        <div style={innerStyle}></div>
      </div>
    </>
  );
});

export default ScrollBox;
```


React App
localhost:3000
맨 밑으로


React App
localhost:3000
Elements
Console
Recorder
Sources
top
Filter
3 messages
3 user messa...
No errors
No warnings
3 info
No verbose
스크롤의 총 길이는 650
스크롤의 위치는 0
스크롤위치가 스크롤 높이만큼 내려갔습니다.

컴포넌트 끼리 데이터를 공유할 때는 부모 <-> 자식 흐름으로 교류해야 해야 한다.

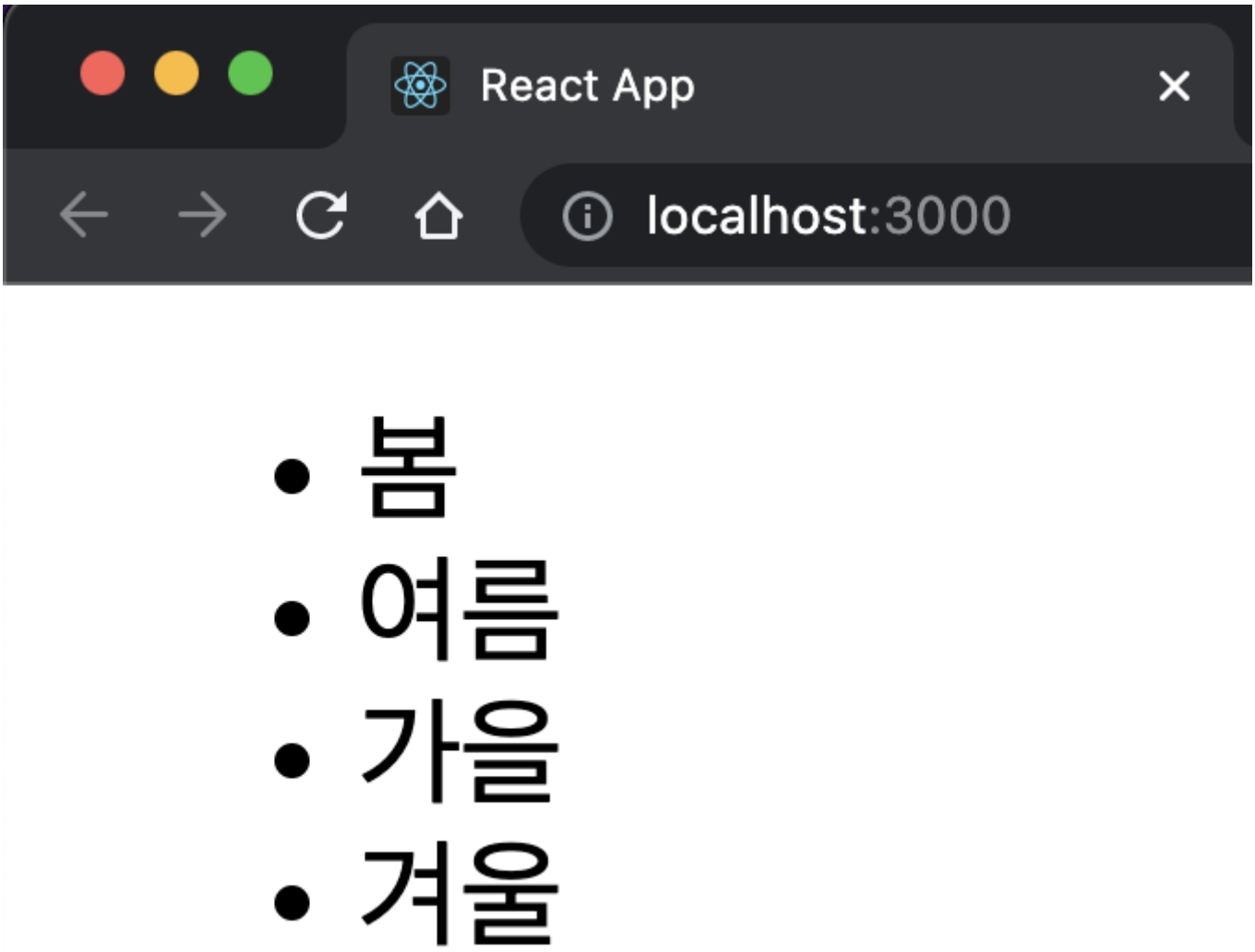
> 일반적으로 forwardRef() 함수는 HTML 엘리먼트 대신에 사용되는 최말단 컴포넌트(ex. <Input/>, <Button/>)를 대상으로 주로 사용되며, 그 보다 상위 컴포넌트에서는 forwardRef() 함수를 사용하는 것이 권장되지 않습니다. 왜냐하면 어떤 컴포넌트의 내부에 있는 HTML 엘리먼트의 레퍼런스를 외부에 있는 다른 컴포넌트에서 접근하도록 하는 것은 컴포넌트 간의 결합도(coupling)을 증가시켜 애플리케이션의 유지보수를 어렵게 만들기 때문입니다.

@ 출저 : https://www.daleseo.com/react-forward-ref/

04. 컴포넌트 반복

데이터 배열을 컴포넌트 배열로 변환하기

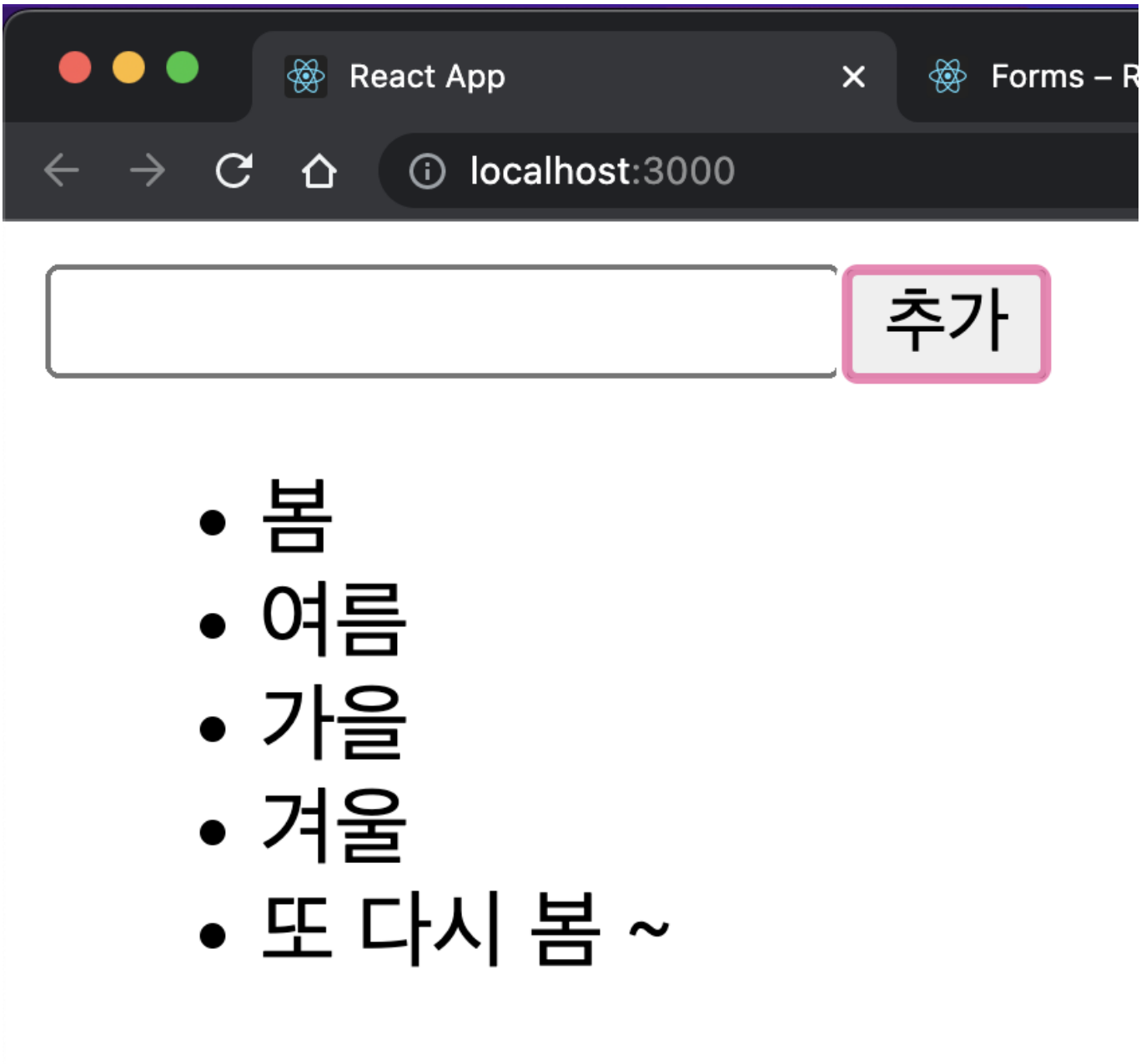
```
import React from 'react';

const Sample = () => {
    const name = ["봄","여름","가을","겨울"];
    const nameList = name.map((name,i)=><li key={i}>{name}</li>);
    return (
        <div>
            <ul>{nameList}</ul>
        </div>
    );
};

export default Sample;
```



배열로 반환하는 경우 데이터가 가진 고윳값을 key 값으로 설정해야 한다.

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

const IterationSample = () => {
    const [names,setNames]= useState([
        {id:1,text:"봄"},
        {id:2,text:"여름"},
        {id:3,text:"가을"},
        {id:4,text:"겨울"},
    ]);
    const [inputText,setInputText] = useState();
    const [nextId , setNextID] = useState(5);
    const onChange=(e)=>{setInputText(e.target.value)}
    const onClick=()=>{
        const nextNames = names.concat({id:nextId,text:inputText})
        setNextID(nextId+1)
        setNames(nextNames);
        setInputText("")
    }
     const namesList = names.map(name=><li key={name.id}>{name.text}</li>)
```

```
    return (
        <div>
            <input type="text" value={inputText} onChange={onChange}/>
            <button onClick={onClick}>추가</button>
            <ul>{namesList}</ul>
        </div>
    );
};

export default IterationSample;
```



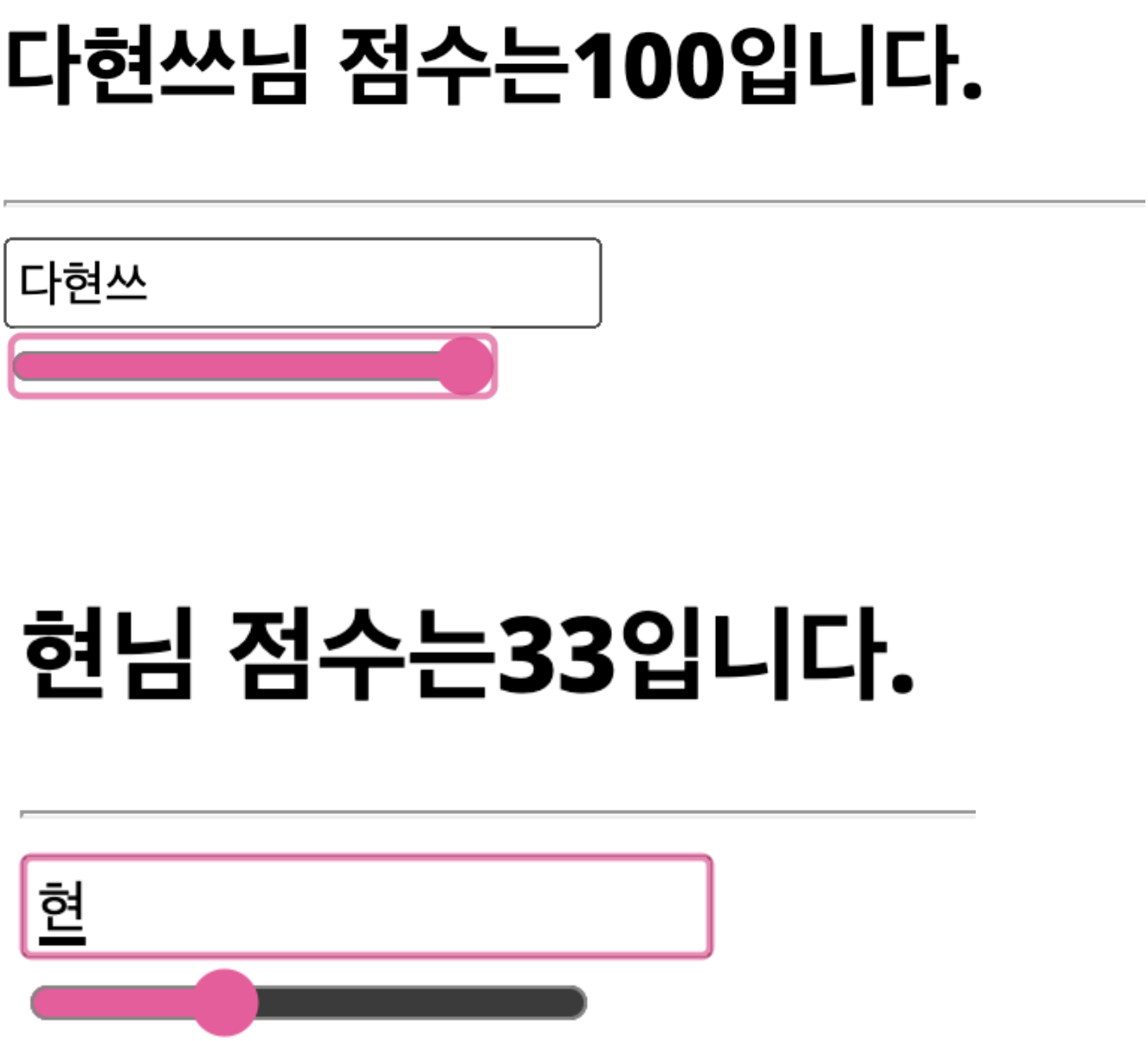
배열의 새 항목을 추가할 때는 배열의 push 함수를 사용하지 않고 concat을 사용한다. push 함수는 기존 배열 자체를 변경해주는 반면 concat은 새로운 배열을 만들어 기존 상태를 그대로 두면서 불변성을 유지해주기 때문이다. filter도 똑같이 사용 가능하다.

05. HOOK

useState

리액트 컴포넌트가 가변적인 상태를 지닐 수 있게 해준다. 기본값을 설정하고 함수가 호출될 때 샅애를 설정하는 함수를 이용해 상태값을 변경해준다.

```
import React from 'react';
import {useState} from "react";

const MyState = () => {
const [name,setname] = useState("");
const [grade,setgrade] =useState(50);

const nameState=(e)=>{
    setname(e.currentTarget.value);
}
const gradeState=(e)=>{
    setgrade(e.currentTarget.value);
}
    return (
        <div>
            <h2>{name}님 점수는{grade}입니다. </h2>
            <hr/>
            <input type="text" onChange={nameState} placeholder="이름을 적으세
요."/><br/>
            <input type="range" onChange={gradeState}/>
        </div>
    );
};

export default MyState;
```



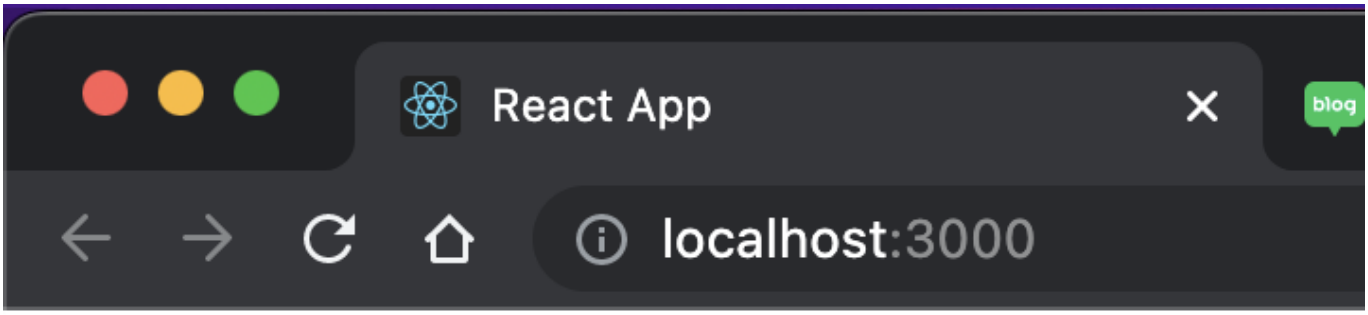
useEffect

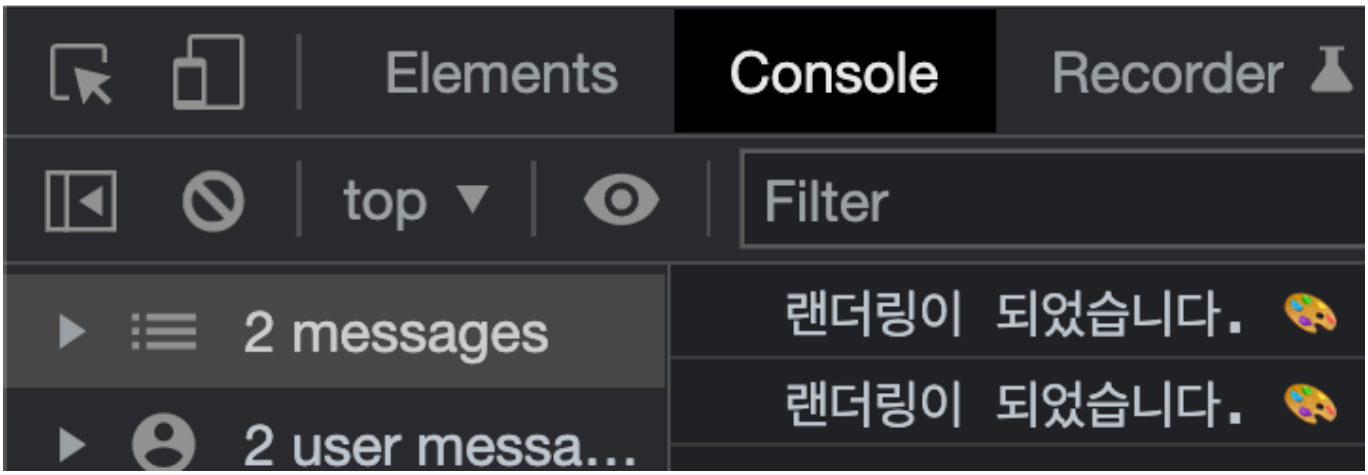
리액트 컴포넌트가 랜더링 될 때마다 특정 작업을 수행 할 수 있도록 설정 할 수 있는 HOOK

가장 기본 적인 사용 방법

```
import React from 'react';

const UseEffect = () => {
    React.useEffect(() => {
        console.log("랜더링이 되었습니다. 🎨 ");
    })
    return (
        <div>

        </div>
```

```
    );
};

export default UseEffect;
```





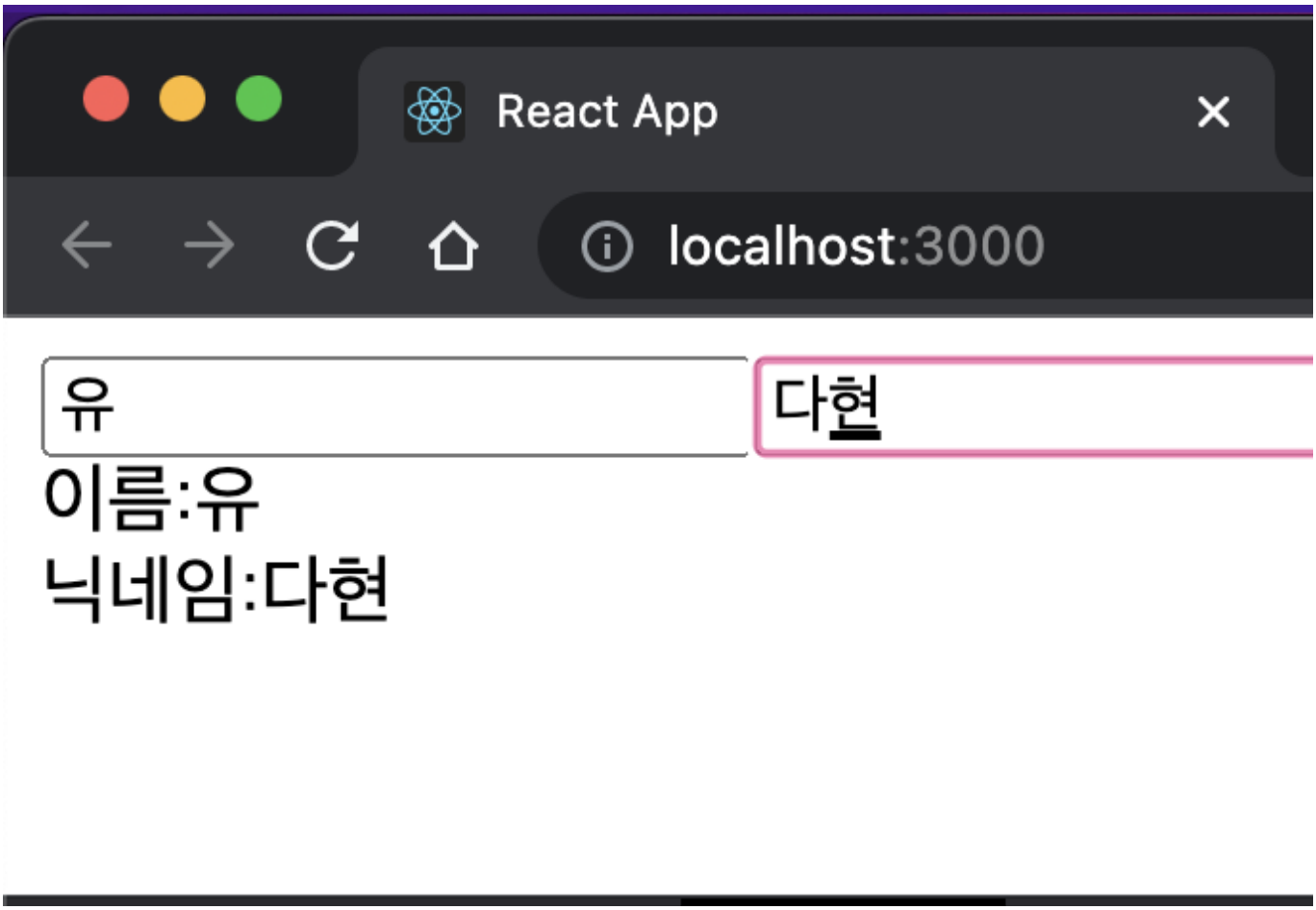
두번 나오는 이유는 <React.StrictMode> 때문

---

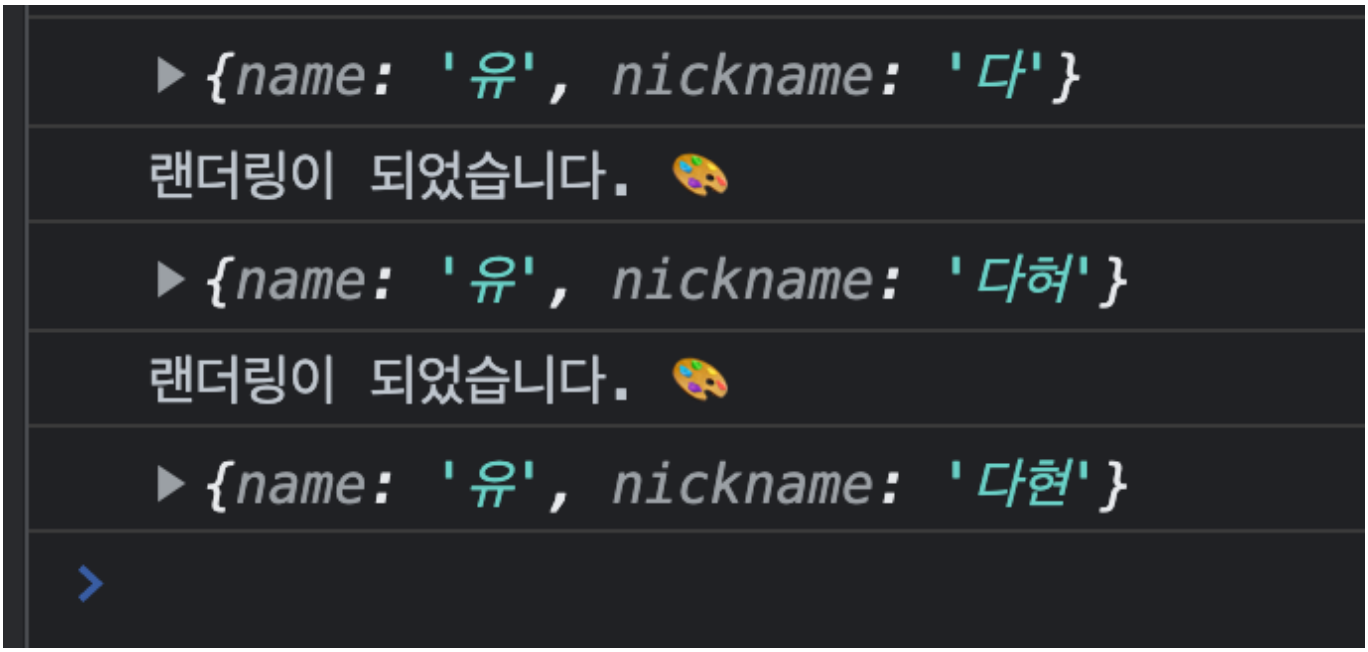
state의 상태가 바뀔 때 랜더링 될 수 있다.

```
import React from 'react';

const UseEffect = () => {
    const [name,setNames] = React.useState();
    const [nickname,setNickNames] = React.useState();
    React.useEffect(() => {
```

```
        console.log("랜더링이 되었습니다. 🎨 ");
        console.log({
            name,nickname
        })
    })
    const onChange =(e)=>{
        setNames(e.target.value);
    }
    const onChangeNickname =(e)=>{
        setNickNames(e.target.value);
    }
    return (
        <div>
            <input type="text" value={name} onChange={onChange}/>
            <input type="text" value={nickname} onChange=
{onChangeNickname}/><br/>
            이름:{name}<br/>
            닉네임:{nickname}
        </div>
    );
};

export default UseEffect;
```


React App
localhost:3000
유
다현
이름:유
닉네임:다현

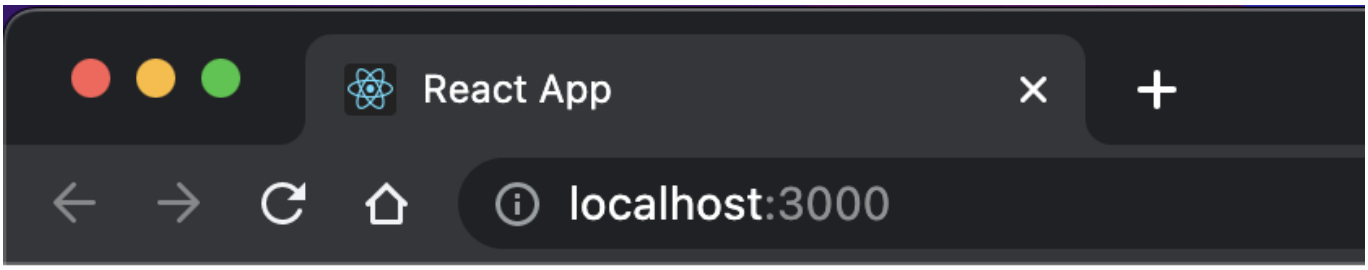

랜더링이 되었습니다.
{name: 'ㅇ', nickname: undefined}
랜더링이 되었습니다.
{name: '유', nickname: undefined}
랜더링이 되었습니다.
{name: '유', nickname: 'ㄷ'}
랜더링이 되었습니다.
{name: '유', nickname: '다'}
랜더링이 되었습니다.
{name: '유', nickname: '닭'}
랜더링이 되었습니다.




```
import React from "react";
import UseEffect from './UseEffect';

function App() {
   const [visible,setVisible] = React.useState(false);
  return (
    <div>
      <button onClick={
        ()=>{
          setVisible(!visible);
        }
      }>{visible ? "숨기기":"보이기"}</button>
    {visible && <UseEffect/>}
    </div>
   );
}

export default App;
```

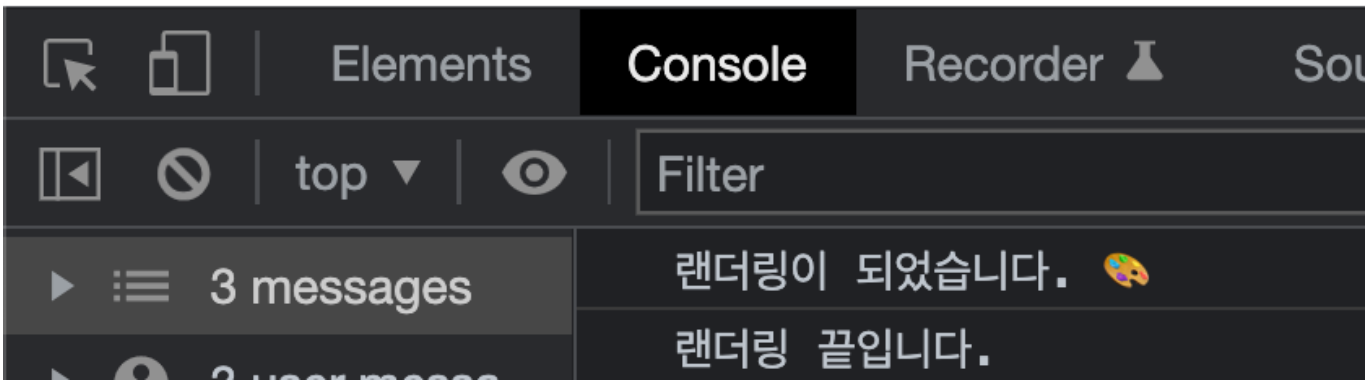
```
import React from 'react';

const UseEffect = () => {
    const [name,setNames] = React.useState();
    const [nickname,setNickNames] = React.useState();
    React.useEffect(() => {
        console.log("랜더링이 되었습니다. 🎨 ");

        return ()=>{
            console.log("랜더링 끝입니다. ")
        }
    },[name])
    const onChange =(e)=>{
```

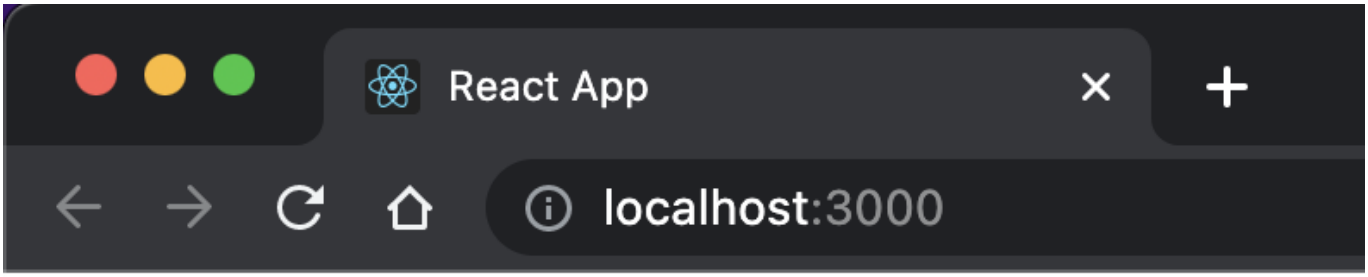
```
        setNames(e.target.value);
    }
    const onChangeNickname =(e)=>{
        setNickNames(e.target.value);
    }
    return (
        <div>
            <input type="text" value={name} onChange={onChange}/>
            <input type="text" value={nickname} onChange=
{onChangeNickname}/><br/>
            이름:{name}<br/>
            닉네임:{nickname}
        </div>
    );
};

export default UseEffect;
```



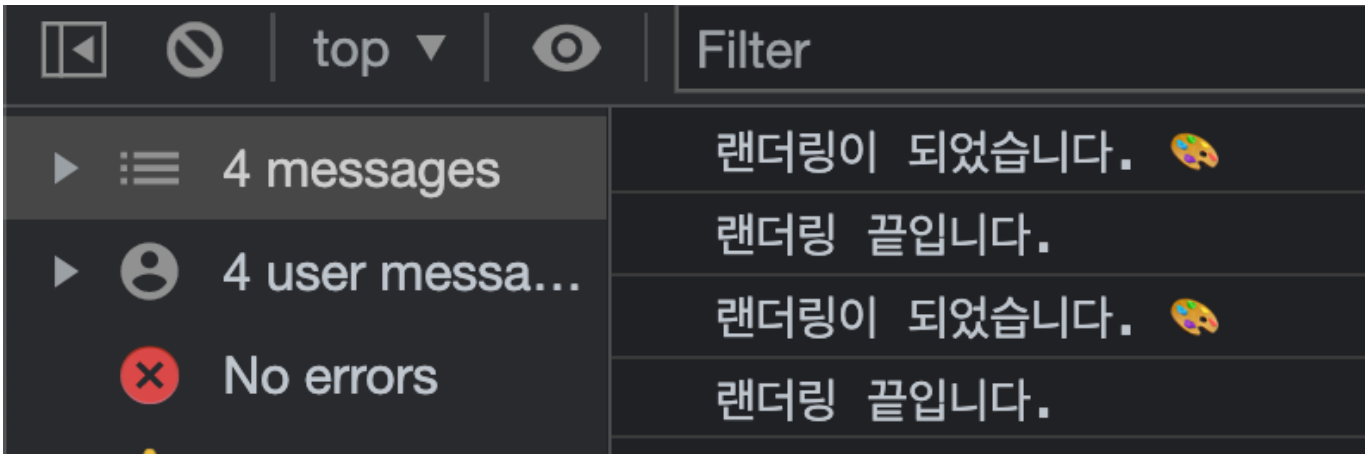
숨기기

이름:
닉네임:





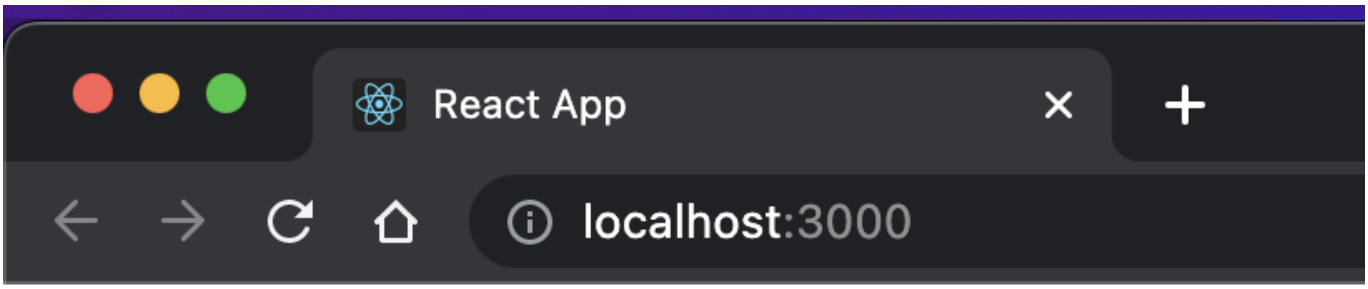
보이기



useReduce

리듀스는 현재 상태 , 업데이트를 위해 필요한 정보를 담은 action 값을 전달 받아 새로운 상태를 번환하는 함수

```
import React from 'react';

function reducer(state,action){
    return{
        ...state,
        [action.name]:action.value
    }
}
const UseReducer2 = () => {
     const [state,dispatch] = React.useReducer(reducer,{
         name:"",
         nickname:""
     })

     const {name,nickname} = state;
     const onChange =e=>{
         dispatch(e.target);
     }
    return (
        <div>
            <input name="name" value={name} onChange={onChange}/>
            <input name="nickname" value={nickname} onChange={onChange}/>
<br/>
            이름:{name}<br/>
            닉넴:{nickname}
        </div>
    );
};

export default UseReducer2;
```



유 다현

이름:유

닉넴: 다현

useMemo

함수형 컴포넌트내부에서 발생하는 연산을 최적화 할 수 있다. 메모리얼에 저장하는 방식

```
import React from 'react';

const getAVG = numbers=>{
    console.log("계산중");
    if(numbers.length === 0) return 0;
    const sum = numbers.reduce((a,b)=>a+b);
    return sum / numbers.length;
};
const UseMemo = () => {
    const [list,setList] =React.useState([]);
    const [number,setNumber]= React.useState("");
    const onChange = e=>{
        setNumber(e.target.value);
    }
    const onInsert =e=>{
        const nextList = list.concat(parseInt(number));
        setList(nextList);
        setNumber("");
    }
    return (
        <div>
            <input value={number} onChange={onChange}/>
            <button onClick={onInsert}>등록</button>
            <ul>
                {list.map((v,i)=>{
                   return  <li key={i}>{v}</li>
                })}
            </ul>
            <p>평균값 : {getAVG(list)}</p>
        </div>

    );
};
```

```
export default UseMemo;
```

useCallback

useMemo와 비슷하지만 렌더링 성능을 최적화 할 때 쓰인다.

```
import React from 'react';

const getAVG = numbers=>{
    console.log("계산중");
    if(numbers.length === 0) return 0;
    const sum = numbers.reduce((a,b)=>a+b);
    return sum / numbers.length;
};
const UseCallback = () => {
    const [list,setList] =React.useState([]);
    const [number,setNumber]= React.useState("");

    const onChange =React.useCallback(e=>{
        setNumber(e.target.value);
    },[]); // 컴포넌트가 처음으로 렌더링 될 때만 함수 생성

    const onInsert =React.useCallback(()=>{
        const nextList = list.concat(parseInt(number));
        setList(nextList);
        setNumber("");
    },[number,list])//number 혹은 list 가 변경 되었을 때만 함수 생성

    return (
        <div>
            <input value={number} onChange={onChange}/>
            <button onClick={onInsert}>등록</button>
            <ul>
                {list.map((v,i)=>{
                   return  <li key={i}>{v}</li>
                })}
            </ul>
            <p>평균값 : {getAVG(list)}</p>
        </div>

    );
};

export default UseCallback;
```