12. immer으로 쉽게 불변성 유지하기

- 값 하나를 수정하기 위해 전개연산자를 이용하여 다른 값은 유지하면서 새로운 값을 지정하기 위해서 이다.
- 복잡한 상태의 로직을 다룰 때마다 전개연산자를 여러번 사용하는 것은 번거로운 작업이다.
- 이러한 상황에서 `immer을 사용하면 구조가 복잡한 객체도 쉽게 불변성을 유지하면서 업데이트` 할 수가 있다.

12.1 immer 설치 및 사용

12.1.1 프로젝트 준비

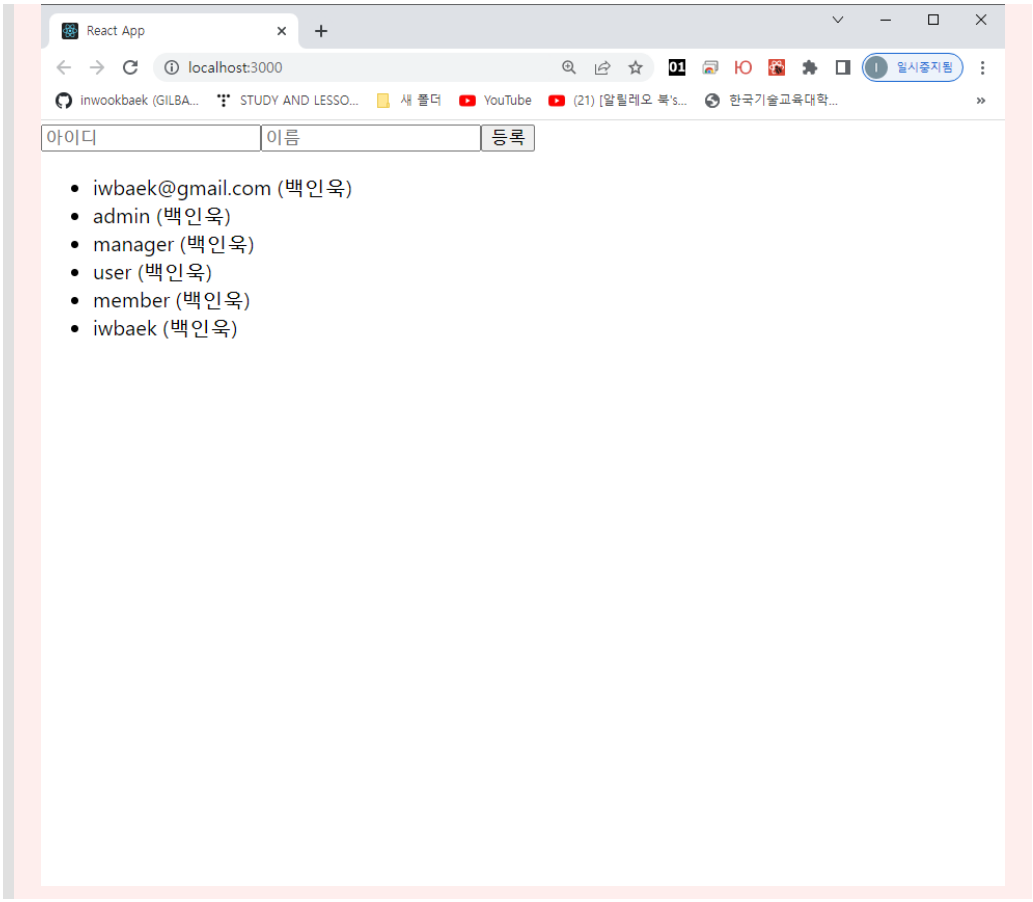
- 설치 : `yarn add immer`

12.1.2 immer을 사용하지 않고 불변성 유지하기

App.js - immer 적용 전

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

const App = () => {
  const nextId = useRef(1);
  const [form, setForm] = useState({ name: '', username: '' });
  const [data, setData] = useState({
    array: [],
    uselessValue: null
  });

  // input 수정을 위한 함수
  const onChange = useCallback(
    e => {
      const { name, value } = e.target;
      setForm({
      ...form,
      [name]: [value]
    });
  }, [form]);

  // form 등록을 위한 함수
  const onSubmit = useCallback(
    e => {
      e.preventDefault();
      const info = {
        id: nextId.current,
        name: form.name,
        username: form.username
      };

      // array 에 새 항목 등록
      setData({
```

```
        ...data,
        array: data.array.concat(info)
      });

      // form 초기화
      setForm({
        name: '',
        username: ''
      });
      nextId.current += 1;
    }, [data, form.name, form.username]
  );

  // 항목을 삭제하는 함수
  const onRemove = useCallback(
    id => {
      setData({
        ...data,
        array: data.array.filter(info => info.id !== id)
      });
    }, []
  );

  return (
    <div>
      <form onSubmit={onSubmit}>
        <input
          name="username"
          placeholder="아이디"
          value={form.username}
          onChange={onChange}
        />
        <input
          name="name"
          placeholder="이름"
          value={form.name}
          onChange={onChange}
        />
        <button type="submit">등록</button>
      </form>
      <div>
        <ul>
          {data.array.map(info => (
            <li key={info.id} onClick={() => onRemove(info.id)}>
              {info.username} ({info.name})
            </li>
          ))}
        </ul>
      </div>
    </div>
  );
};

export default App;
```



- 폼에서 id/name을 입력하면 하단 리스트에 추가되고 항목을 클릭하면 삭제되는 컴퍼넌트이다.
- 전개연산자와 배열내장함수를 사용하여 불변성을 유지하는 것은 어렵지 않지만 상태가 복잡해지면 번거운 작업이 될 수 있다.

12.1.3 immer 사용

예시코드

```
import produce from 'immer';
const nextState = produce(orginalState, draft => {
  // 변경할 값 바꾸기
  draft.somewhre.deep.inside = 5
})
```

procuce함수는 2가지 파라미터를 받는다

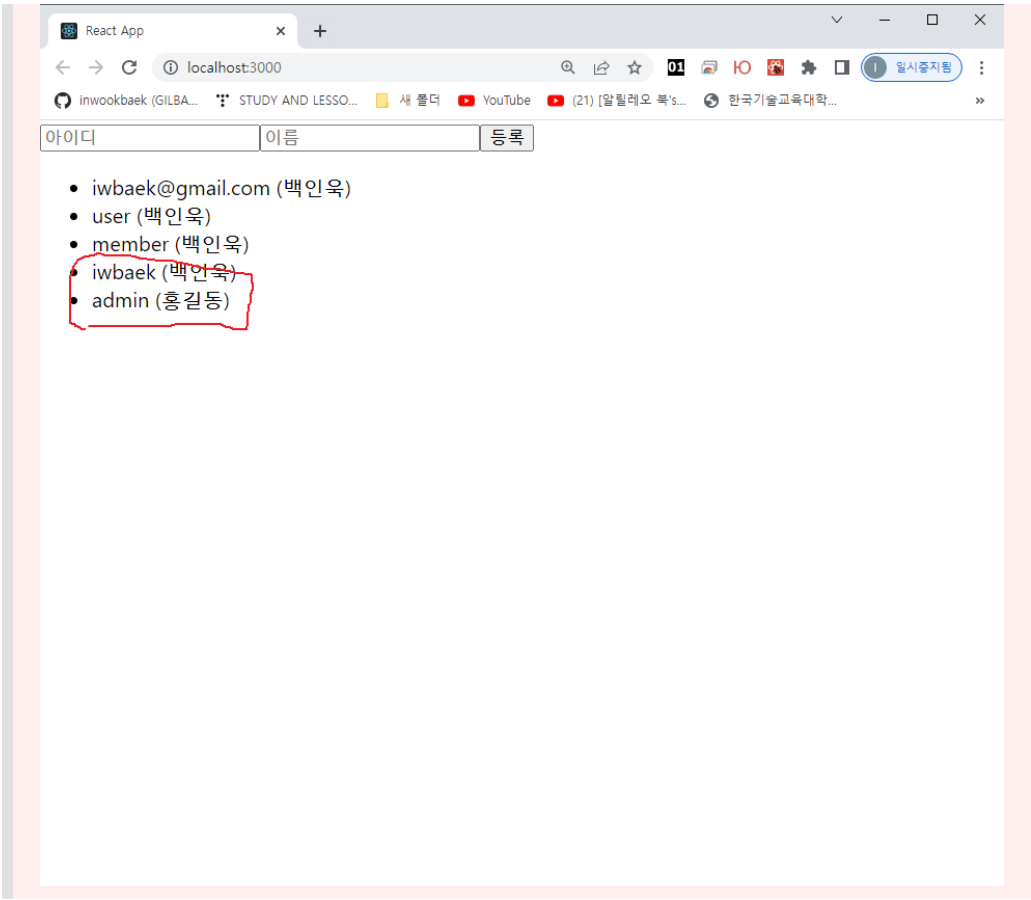
- 첫 번째 : 수정하고 싶은 상태
- 두 번째 : 상태를 업데이트를 정의하는 함수
  - 함수 내부에서 값을 변경하면 produce함수가 불변성 유지를 대신해 주면서 새로운 상태를 생성
- immr의 샐심은 불변성에 신경을 쓰지 않는 것처럼 코드를 작성하되 불변성관리는 제대로 해주는 것

App.js - immer 적용 후

```
import React, { useRef, useCallback, useState } from 'react';
import produce from 'immer';

const App = () => {
```

```
  const nextId = useRef(1);
  const [form, setForm] = useState({ name: '', username: '' });
  const [data, setData] = useState({
    array: [],
    uselessValue: null
  });

  // input 수정을 위한 함수
  const onChange = useCallback(e => {
    const { name, value } = e.target;
    setForm(
      produce(form, draft => {
        draft[name] = value;
      })
    );
  }, [form]);

  // form 등록을 위한 함수
  const onSubmit = useCallback(
    e => {
      e.preventDefault();
      const info = {
        id: nextId.current,
        name: form.name,
        username: form.username
      };

      // array 에 새 항목 등록
      setData(
        produce(data, draft => {
          draft.array.push(info);
        })
      );

      // form 초기화
      setForm({
        name: '',
        username: ''
      });
      nextId.current += 1;
    },
    [data, form.name, form.username]
  );

  // 항목을 삭제하는 함수
  const onRemove = useCallback(
    id => {
      setData(
        produce(data, draft => {
          draft.array.splice(draft.array.findIndex(info => info.id === id),
1);
        })
      );
    },
    [data]
  );
```

```
  return (
    <div>
      <form onSubmit={onSubmit}>
        <input
          name="username"
          placeholder="아이디"
          value={form.username}
          onChange={onChange}
        />
        <input
          name="name"
          placeholder="이름"
          value={form.name}
          onChange={onChange}
        />
        <button type="submit">등록</button>
      </form>
      <div>
        <ul>
          {data.array.map(info => (
            <li key={info.id} onClick={() => onRemove(info.id)}>
              {info.username} ({info.name})
            </li>
          ))}
        </ul>
      </div>
    </div>
  );
};

export default App;
```



- immer로 컴퍼넌트 상태를 작성할 때 객체안에 있는 값을 직접 수정하거나, 배열에 push, splice함수를 사용해도 무방하다.
- immer을 사용한다고 해도 무조건 코드가 간결해 지지는 않는다.
- onRemove인 경우 배열내장함수인 filter를 사용하는 것이 코드가 더 깔끔하다.
- `immer은 불변성을 유지하는 코드가 복잡할 때만 사용해도 충분` 하다

12.1.5 useState의 함수형 업데이트와 immer 함께쓰기

- immer의 `produce함수를 호출할 때 첫 번째 파라미터가 함수형태라면 업데이트함수를 반환` 한다.

예시코드

```
const update = prouce(draft => {
  draft.value = 2;
});

const originalState = {
  value: 1,
  foo: 'bar',
};
const nextState = update(originalState);
console.log(nextState) // {value:2, foo: 'bar'}
```

- 이런 immer의 속성과 useState의 함수형 업데이트를 함께 활용하면 코드를 더욱 깔끔하게 작성할 수 있다.

App.js 수정 - useState의 함수형 업데이트와 immer 함께쓰기

```
import React, { useRef, useCallback, useState } from 'react';
import produce from 'immer';

const App = () => {
  const nextId = useRef(1);
  const [form, setForm] = useState({ name: '', username: '' });
  const [data, setData] = useState({
    array: [],
    uselessValue: null
  });

  // input 수정을 위한 함수
  const onChange = useCallback(e => {
    const { name, value } = e.target;
    setForm(
      produce(draft => {
        draft[name] = value;
      })
    );
  }, []);

  // form 등록을 위한 함수
  const onSubmit = useCallback(
    e => {
      e.preventDefault();
      const info = {
        id: nextId.current,
```

```
        name: form.name,
        username: form.username
      };

      // array 에 새 항목 등록
      setData(
        produce(draft => {
          draft.array.push(info);
        })
      );

      // form 초기화
      setForm({
        name: '',
        username: ''
      });
      nextId.current += 1;
    },
    [form.name, form.username]
  );

  // 항목을 삭제하는 함수
  const onRemove = useCallback(
    id => {
      setData(
        produce(draft => {
          draft.array.splice(draft.array.findIndex(info => info.id === id),
1);
        })
      );
    },
    []
  );

  return (
    <div>
      <form onSubmit={onSubmit}>
        <input
          name="username"
          placeholder="아이디"
          value={form.username}
          onChange={onChange}
        />
        <input
          name="name"
          placeholder="이름"
          value={form.name}
          onChange={onChange}
        />
        <button type="submit">등록</button>
      </form>
      <div>
        <ul>
          {data.array.map(info => (
            <li key={info.id} onClick={() => onRemove(info.id)}>
              {info.username} ({info.name})
            </li>
```

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

export default App;
```