8강

useEffect Hook

➢ useEffect 란?

- useEffect는 컴포넌트가 렌더링될 때 실행되는 부수효과 함수이다.
- 여기서 "부수효과(Side Effect)"란, 렌더링 이외의 작업을 의미한다.

예)

1. 서버에서 데이터 불러오기 (fetch)
2. 브라우저 API 사용 (localStorage, window 이벤트 등)
3. 타이머 설정 (setTimeout, setInterval)
4. 콘솔 로그나 DOM 조작 등

- 이런 작업은 화면을 그리는(rendering) 행위와는 직접 관련이 없기 때문에
- React는 useEffect라는 별도 Hook을 통해 처리하도록 한다.

➢ useEffect 기본 형식

useState

```
import { useEffect } from "react";

useEffect(() => {
  // 실행할 코드 (부수효과)
  console.log("컴포넌트가 렌더링될 때 실행됩니다.");

  return () => {
    // 정리(clean-up) 코드 (선택사항)
    console.log("컴포넌트가 사라질 때 실행됩니다.");
  };
}, [의존성배열] );
```

구성요소	설명
() => { ... }	실제로 실행할 함수
return () => { ... }	정리(clean-up) 함수 (선택)
[의존성배열]	언제 useEffect가 실행될지를 결정하는 배열

➢ 의존성 배열(Dependency Array)에 따른 동작 차이

(1) 의존성 배열 없음

```
useEffect(() => {
    console.log("매 렌더링마다 실행됨");
});
```

✔ 컴포넌트가 렌더링 될 때마다 실행된다.

✔ 성능에 부담이 될 수 있으므로 일반적으로 잘 사용하지 않는다.

(2) 빈 배열 []

```
useEffect(() => {
    console.log("처음 한 번만 실행됨 (Mount)");
}, []);
```

✔ 컴포넌트가 처음 화면에 나타날 때 1번만 실행된다.

✔ 주로 데이터 로드(fetch), 초기 설정 등에 사용한다.

➢ 의존성 배열(Dependency Array)에 따른 동작 차이

(3) 특정 상태(state)나 props를 넣은 경우

```
useEffect(() => {
    console.log("count가 바뀔 때마다 실행됨");
}, [count] );
```

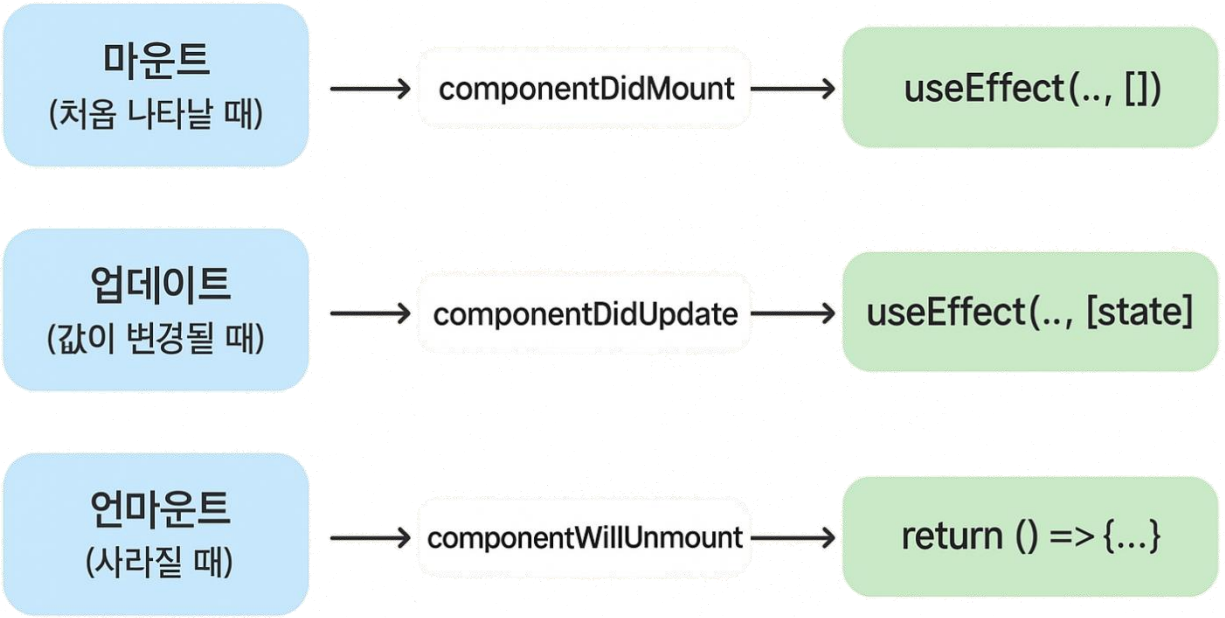
✔의존성 배열 안의 값이 변경될 때마다 실행된다.

✔상태 변경에 따른 후속 작업을 처리할 때 사용한다.

➢ 생명주기(Lifecycle)와의 관계

- React의 컴포넌트 생명주기(Lifecycle) 흐름과 useEffect의 관계를 다음과 같이 볼 수 있다

생명주기	클래스형 컴포넌트 메서드	useEffect에서의 대응
마운트 (처음 나타날 때)	componentDidMount	useEffect(..., [])
업데이트 (값이 변경될 때)	componentDidUpdate	useEffect(..., [state])
언마운트 (사라질 때)	componentWillUnmount	return () => {...} (정리 함수)



➢ 여기서 잠깐

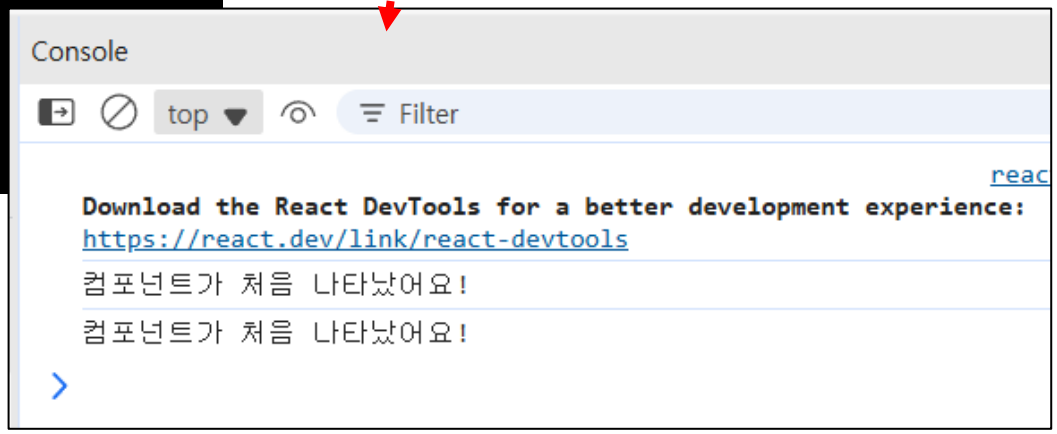
조건

① 처음 렌더링 될 때 실행

② 컴포넌트가 화면에 처음 나타날 때만 useEffect 실행

```
export default function Eff02() {
  useEffect(() => {
    console.log('컴포넌트가 처음 나타났어요!');
  }, []);


  return <h2>useEffect 기본 실행</h2>;
}
```



➤ 여기서 잠깐

조건

① 랜더링마다 실행 하기

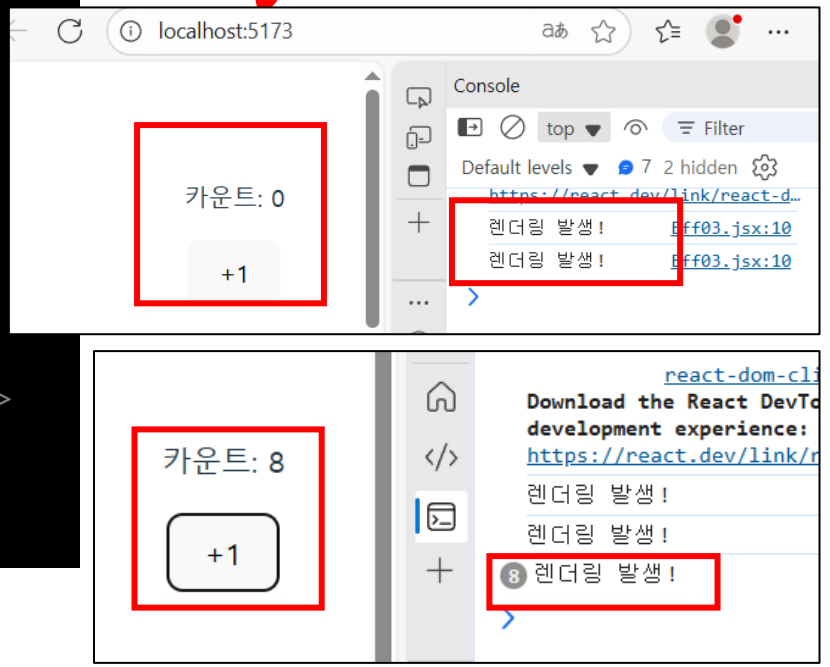
② 의존성 배열이 없을 때 실행 시점은 **컴포넌트가 랜더링 될 때 마다 useEffect가** 실행 되고 있다.

effect가 어떤 값에 의존하는 지 명시되지 않았으니, 매 렌더링마다 실행되고 있다.

```
export default function Eff03() {
  const [count, setCount] = useState(0);

  useEffect(() => {
    console.log('렌더링 발생!');
  });

  return (
    <div>
      <p>카운트: {count}</p>
      <button onClick={() => setCount(count + 1)}>+1</button>
    </div>
  );
}
```



➤ 여기서 잠깐

조건

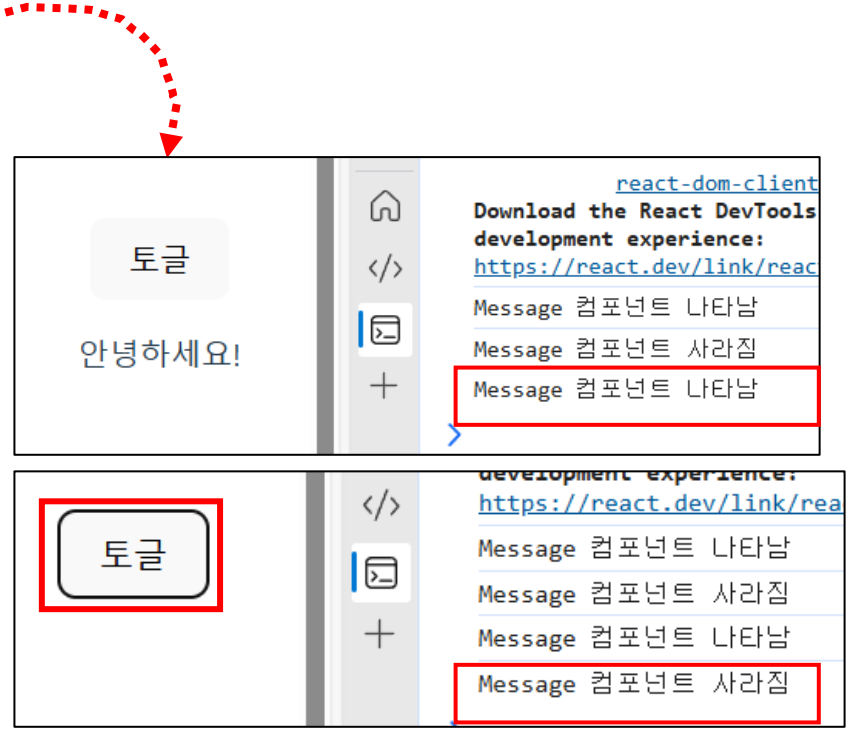
① cleanup 함수

② **컴포넌트가 사라질 때 실행되는 Cleanup 함수**

```
function Message() {
  useEffect(() => {
    // 컴포넌트가 나타날때 마운트(mount)
    console.log('Message 컴포넌트 나타남');
    return () => console.log('Message 컴포넌트 사라짐');
    // cleanUp함수 => 컴포넌트가 사라질 때 언마운트(unmount)
  }, []);

  return <p>안녕하세요!</p>;
}

export default function Eff04() {
  const [show, setShow] = useState(true);
  return (
    <div>
      <button onClick={() => setShow(!show)}>토글</button>
      {show && <Message />}
    </div>
  );
}
```



문제] 다음의 조건에 만족하도록 React를 작성 하시오.

조건

① 자동으로 1초씩 초가 경과하도록 작성하시오.

② 함수이름은 Eff05.jsx로 작성하시오.

③ useState(), useEffect()를 이용해 작성하시오.

④ setInterval(), clearInteravl()함수를 이용해 작성하시오.

⑤ src 폴더안에 -> Effect 폴더 생성 -> Eff05.jsx 파일을 생성하여 작성하시오.

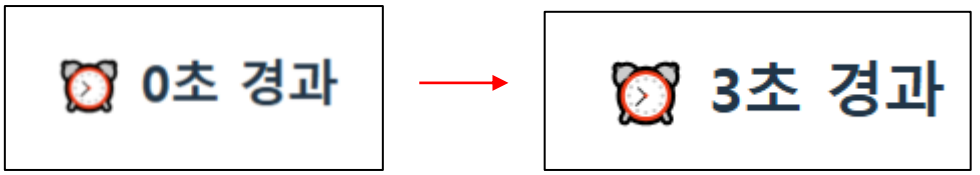
⑥ src 폴더안에 -> App.jsx에 작성한 Eff05.jsx를 import하여 작성하시오

```
setInterval(실행할함수, 시간간격(ms));

setInterval(() => {
  console.log('1초마다 실행');
}, 1000);
```

setInterval( ) 사용 방법

<출력 결과>



➢ useEffect로 데이터 불러오기 – fetch ~ then() 비동기 함수 사용

```
import { useEffect, useState } from "react";


export default function UserList() {
  // 사용자 목록을 담을 state
  const [users, setUsers] = useState([]);

  useEffect(() => {
    // 1. 컴포넌트가 처음 렌더링 될 때 실행
    fetch("https://jsonplaceholder.typicode.com/users")
      // 2. 서버에서 응답(Response)을 받으면 JSON으로 변환
      .then(res => res.json())
      // 3. 변환된 데이터를 users 상태에 저장
      .then(data => setUsers(data));
  }, []); // 4. [] → 마운트 시 1회만 실행


  return (
    <ul>
      {users.map(user => (
        <li key={user.id}>{user.name}</li>
      ))}
    </ul>
  );
}
```

요약

- 컴포넌트가 처음 화면에 나타날 때(mount) API 요청이 실행됨
- setUsers로 상태가 갱신되면 자동으로 다시 렌더링 됨

사용자 목록

- Leanne Graham
- Ervin Howell
- Clementine Bauch
- Patricia Lebsack
- Chelsey Dietrich
- Mrs. Dennis Schulist
- Kurtis Weissnat
- Nicholas Runolfsdottir V
- Glenna Reichert
- Clementina DuBuque

➢ useEffect로 데이터 불러오기 – async ~ await 비동기 함수 사용

```
export default function Eff07() {
  const [users, setUsers] = useState([]);

  useEffect(() => {
    // 1. 비동기 함수 정의 (async)
    // fetch는 비동기 함수이기 때문에 await를 쓰려면 async 함수 안에서 실행해야 함
    const load = async () => {
      // 2. 서버(API)에서 데이터 가져오기 (fetch는 Promise 반환)
      const res = await fetch('https://jsonplaceholder.typicode.com/users');
      // 3. 응답(response)을 JSON 형태로 변환 (이것도 비동기)
      const data = await res.json();
      // 4. 받아온 데이터를 state에 저장 (렌더링 갱신)
      setUsers(data);
    };
    // 함수 호출
    load();
  }, []);

  return (
    <ul>
      {users.map((u) => (
        <li key={u.id}>{u.email}</li>
      ))}
    </ul>
  );
}
```

요약

처음엔 fetch ~ then 짧고 편하지만, 큰 프로젝트에서의 오류 문제와, 유지 보수 문제로 실무에서는 async ~ await 많이 사용한다.

- Sincere@april.biz
- Shanna@melissa.tv
- Nathan@yesenia.net
- Julianne.OConner@kory.org
- Lucio_Hettinger@annie.ca
- Karley_Dach@jasper.info
- Telly.Hoeger@billy.biz
- Sherwood@rosamond.me
- Chaim_McDermott@dana.io
- Rey.Padberg@karina.biz

➢ useEffect 로 데이터 불러오기

https://jsonplaceholder.typicode.com/users


```
[
  {
    "id": 1,
    "name": "Leanne Graham",
    "username": "Bret",
    "email": "Sincere@april.biz",
    "address": {
      "street": "Kulas Light",
      "suite": "Apt. 556",
      "city": "Gwenborough",
      "zipcode": "92998-3874",
      "geo": {
        "lat": "-37.3159",
        "lng": "81.1496"
      }
    },
    "phone": "1-770-736-8031 x56442",
    "website": "hildegard.org",
    "company": {
      "name": "Romaguera-Crona",
      "catchPhrase": "Multi-layered client-server neural-net",
      "bs": "harness real-time e-markets"
    }
  },
  {
    "id": 2,
    "name": "Ervin Howell",
    "username": "Antonette",
    "email": "Shanna@melissa.tv",
    "address": {
      "street": "Victor Plains",
      "suite": "Suite 879",
      "city": "Wisokyburgh",
      "zipcode": "90566-7771",
      "geo": {
        "lat": "-43.9509",
        "lng": "-34.4618"
      }
    },
    "phone": "010-692-6593 x09125",
    "website": "anastasia.net",
    "company": {
      "name": "Deckow-Crist",
```


- JSONPlaceholder는 개발·테스트·교육용으로 만든 무료 가짜(샘플) REST API이다.
- 실제 서비스처럼 GET, POST, PUT, DELETE 같은 요청을 해볼 수 있지만, 실제 데이터베이스에 영구 저장되지는 않는다.

- **만들어진 목적:**
  프론트엔드 개발자들이 API가 아직 준비되지 않았거나 백엔드가 없을 때도 클라이언트 쪽 로직(페칭, 에러 처리, 로딩 표시 등)을 연습하도록 돕기 위해.
- **장점:**
  간단하고 바로 호출 가능, CORS 허용(브라우저에서 바로 호출 가능), 별도 인증 필요 없음, 무료.

➢ JSON(JavaScript Object Notation) 이란

"**자바스크립트 객체 표기법**"

- 데이터를 주고받기 위한 형식(문법)이다.
- 자바스크립트의 객체 문법을 본떠 만든 데이터 교환 포맷이다.

Json 예시

```
{
  "name": "홍길동",
  "age": 25,
  "email": "gildong@example.com"
}
```

위처럼 { } 안에 "키": "값" 쌍으로 이루어진 구조

JSON의 특징

특징	설명
언어 독립적	JS에서 만들었지만, Python·Java·C# 등 모든 언어에서 읽고 쓸 수 있음
가볍다	단순한 텍스트 구조이기 때문에 빠르고 용량이 작음
데이터 전송용	서버와 클라이언트가 정보를 주고받을 때 표준처럼 사용

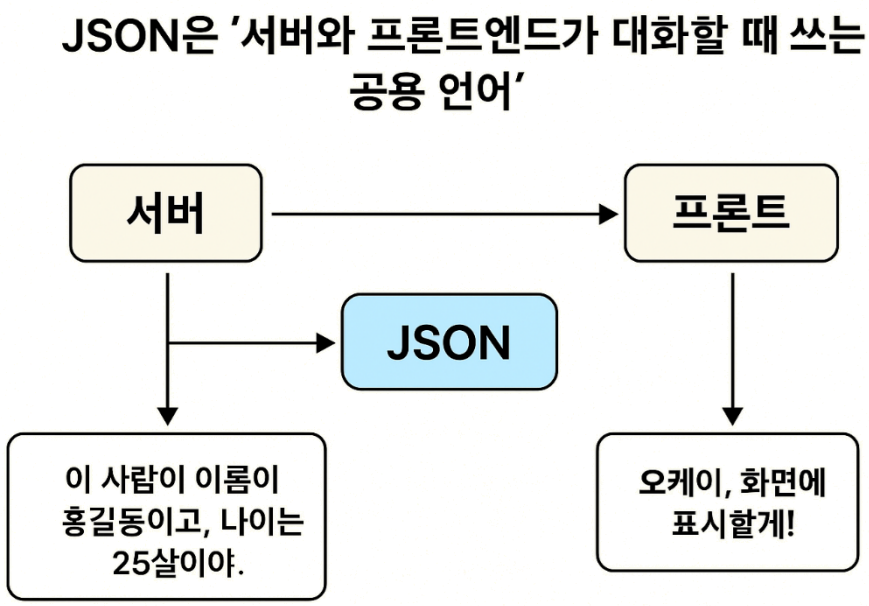
➢ JSON(JavaScript Object Notation) 이란

JSON은 "서버와 프론트엔드가 대화할 때 쓰는 공용 언어"

서버: "이 사람이 이름이 홍길동이고, 나이는 25살이야."

프론트: "오케이, 화면에 표시할게!"

이 대화가 실제로는 JSON 데이터로 이뤄진다.



➢ REST = Representational State Transfer 란

"자원의 상태(데이터)를 표현(전달)하는 방법"

- 주소(URL)와 HTTP 요청 방식(GET, POST 등)으로 서버의 데이터를 주고받는 규칙이다.

REST의 핵심 구성

구성 요소	예시	설명
자원(Resource)	/users, /posts	서버에 있는 데이터(목록, 글, 회원 등)
행동(Method)	GET / POST / PUT / DELETE	자원에 대한 동작 (조회, 등록, 수정, 삭제)
표현(Representation)	JSON, XML 등	데이터를 주고받는 형식 (요즘은 대부분 JSON 사용)

REST의 예시

동작	요청 방식	설명
GET /users	사용자 목록 조회	
GET /users/1	id가 1인 사용자 조회	
POST /users	새 사용자 등록	
PUT /users/1	id가 1인 사용자 정보 수정	
DELETE /users/1	id가 1인 사용자 삭제	

CRUD

➢ 비동기(Asynchronous)

```jsx
import { useState, useEffect } from "react";
export default function UserList() {
  const [users, setUsers] = useState([]);

  useEffect(() => {
    console.log("데이터 로드 시작");
    fetch("https://jsonplaceholder.typicode.com/users")
      .then((res) => res.json())
      .then((data) => setUsers(data));
  }, []); // 최초 1번만 실행

  return (
    <div>
      <h2>👥 사용자 목록</h2>
      <ul>
        {users.map((user) => (
          <li key={user.id}>{user.name}</li>
        ))}
      </ul>
    </div>
  );
}
```

요약

- **브라우저(사용자 인터페이스)**를 멈추지 않고 외부 서버와 데이터를 주고받기 위해서 비동기 방식을 사용해야 한다.

흐름 정리

순서	동작	설명
①	컴포넌트 처음 렌더링	useEffect가 실행됨
②	fetch() 요청 보냄	서버(JSONPlaceholder)에 요청 전송
③	서버가 응답 보냄	JSON 형식의 데이터 수신
④	res.json() 으로 파싱	문자열 → JS 객체로 변환
⑤	setUsers(data) 실행	상태 업데이트 → 화면 다시 렌더링

➢ 동기(Synchronous) & 비동기(Asynchronous)

- 동기(Synchronous): 작업 A가 끝나야 작업 B가 시작됨.
  예) 전화 통화: 통화가 끝나야 다른 일 가능.
  → 브라우저에서 동기 처리로 네트워크 요청을 하면 UI가 멈춤(사용자 입력 불가).

- 비동기(Asynchronous): 작업 A를 요청하고, A가 끝날 때까지 기다리지 않고 다음 작업을 계속함.
  예) 택배 주문: 주문하고 다른 일 함 → 택배 도착하면 알림.
  → 네트워크 요청을 비동기로 하면 UI가 반응 상태(버튼 클릭, 애니메이션) 유지.

  - 비동기는 네트워크처럼 느린 작업을 '백그라운드'에서 처리해서, 웹 페이지(UI)가 멈추지 않고 사용자와 계속 상호작용하게 해 주는 기법이다.
  - fetch는 이런 비동기 통신을 쉽게 해주는 도구이며, 응답이 오면 then이나 await으로 결과를 받는다.

- 웹에서는 네트워크 통신이 느릴 수 있으므로 항상 비동기로 처리해야 사용자 경험(UX)이 유지된다.

➢ 동기(Synchronous) & 비동기(Asynchronous)

➢ 왜 꼭 비동기(Asynchronous)여야 할까?

- 학생이 버튼을 눌러 페이지를 로드했는데, 네트워크가 느려서 10초 동안 화면이 멈춘다
  → 사용자는 이 페이지를 닫아버림.
- 동기 방식이면 브라우저가 완전히 멈추고 다른 버튼도 못 누름.
- 비동기면 로딩 스피너만 보여주고 사용자는 다른 인터랙션 가능.
- 따라서 사용자 경험(UX)을 위해 필수이다.

```
useEffect(() => {
  fetch("https://jsonplaceholder.typicode.com/users")
   .then(res => res.json())
   .then(data => setUsers(data));
}, []);
```

1번만

fetch()는 비동기 네트워크 요청 함수

- 괄호 안의 주소(**URL**)은 REST API의 엔드포인트(즉, 데이터를 주는 서버 주소).
- 이 함수는 Promise 객체를 반환 한다.
  → 즉시 결과를 주는 게 아니라 "데이터가 도착하면 알려줄게"라는 약속을 반환.

fetch의 첫 번째 응답(res)은 Response(응답)객체

- 하지만 그 안에는 실제 데이터(**JSON**)가 문자열로 들어있기 때문에 .json() 메서드로 파싱(해석)해야 실제 자바스크립트 객체(Array/Object) 형태로 쓸 수 있다.
- .json()도 비동기이기 때문에 또 Promise를 반환한다. 그래서 또 .then()을 이어서 써야 한다..

.then(data => setUsers(data))

data는 users 정보가 들어 있는 실제 배열(JSON 파싱 결과)이다.

```
[
  { id: 1, name: "Leanne Graham", ... },
  { id: 2, name: "Ervin Howell", ... },
  ...
]
```

setUsers(data)는 useState로 만든 상태 업데이트 함수
→ users 상태를 새 데이터로 바꾸면서 컴포넌트를 다시 렌더링하게 만든다.
→ 화면(UI)에 사용자 목록이 출력된다.

기초 연습 문제

문제] 다음의 조건에 만족하도록 React를 작성 하시오.

조건

① https://jsonplaceholder.typicode.com/users 주소의 데이터를 불러오시오.

② 불러온 데이터의 사용자 이름(name)을 화면에 목록<ul>으로 출력하시오.

③ useState()와 useEffect() 훅을 이용하여 작성하시오.

④ 버튼을 클릭할 때마다 데이터를 다시 불러올 수 있도록 하시오.

⑤ fetch() 함수를 사용하여 데이터를 요청하시오.

⑥ setTimeOut() 함수를 이용해 약 3초간 <로딩중...>이 화현에 출력되도록 작성하시오.

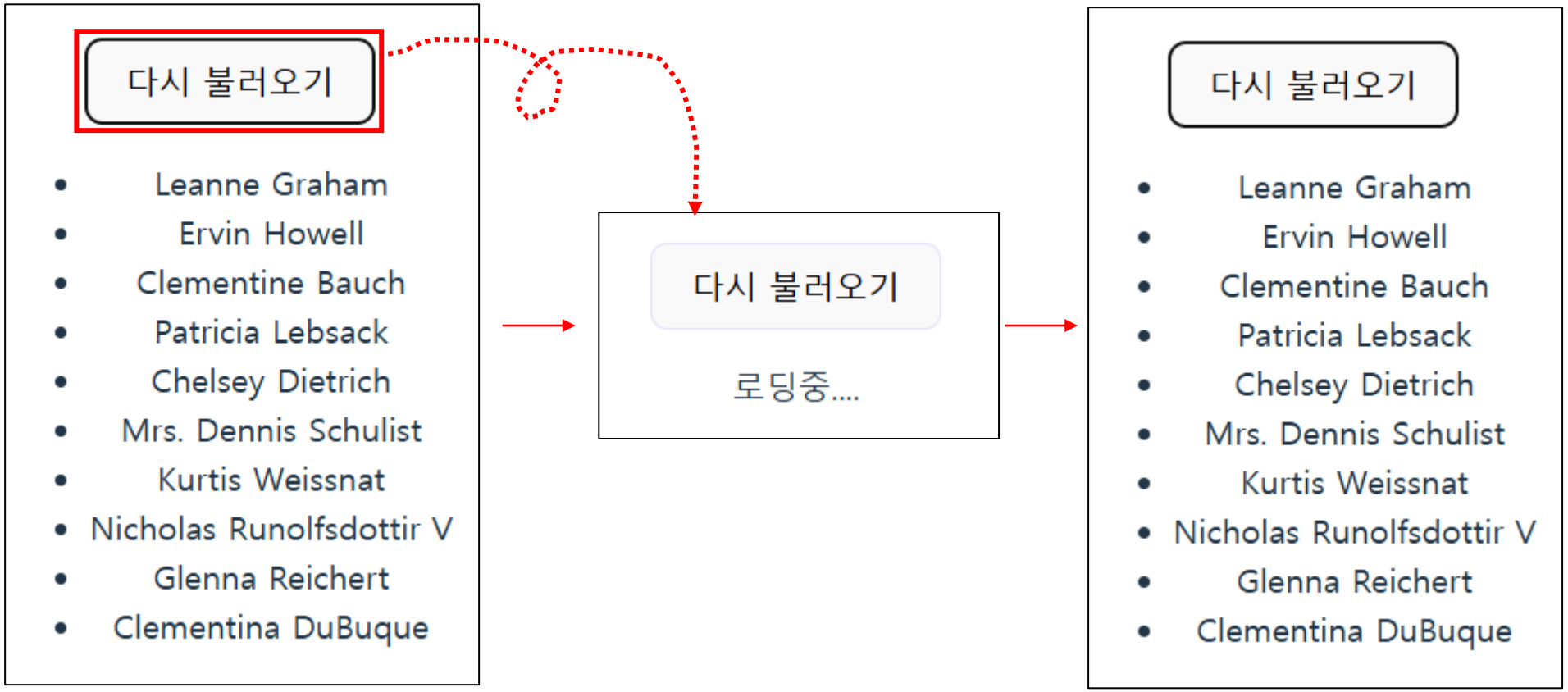
⑦ 함수 이름은 Eff08.jsx로 작성하시오.

⑧ src 폴더 안에 Effect 폴더를 생성하여 Eff08.jsx 파일을 작성하시오.

⑨ src/App.jsx에서 Eff08을 import하여 실행 결과를 확인하시오.

문제] 다음의 조건에 만족하도록 React를 작성 하시오.

<출력 결과>



문제] 다음의 조건에 만족하도록 React를 작성 하시오.

조건

① https://jsonplaceholder.typicode.com/posts 주소의 데이터를 불러오시오.

② 불러온 데이터의 제목(title)을 화면에 목록<ul>으로 출력하시오.

③ useState()와 useEffect() 훅을 이용하여 작성하시오.

④ fetch() 함수를 사용하여 데이터를 요청하시오.

⑤ Slice()함수를 이용해 위에서 부터 5번째까지의 제목만 출력되도록 작성하시오.

⑥ 함수 이름은 Eff09.jsx로 작성하시오.

⑦ src 폴더 안에 Effect 폴더를 생성하여 Eff09.jsx 파일을 작성하시오.

⑧ src/App.jsx에서 Eff09을 import하여 실행 결과를 확인하시오.

문제] 다음의 조건에 만족하도록 React를 작성 하시오.

<출력 결과>

데이터 불러오기

- sunt aut facere repellat provident occaecati excepturi optio reprehenderit
- qui est esse
- ea molestias quasi exercitationem repellat qui ipsa sit aut
- eum et est occaecati
- nesciunt quas odio