[스파르타코딩클럽] 리액트 심화반 5주차

[수업 목표]

1. 프로젝트 구조 관리에 대해 생각해본다.
2. 최적화 기법에 대해 알아본다.
3. 자주 쓰는 기능을 전역화 해본다.
4. 서버 컴포넌트, 서스펜스 등 리액트 18에서 요긴한 기능을 써본다.

[목차]

---

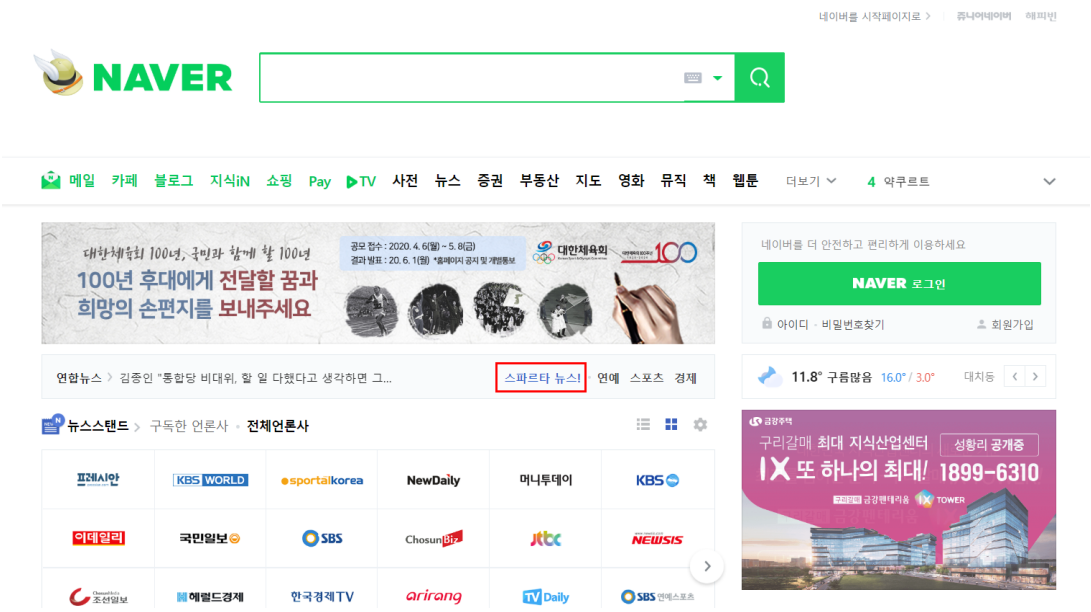
01. 오늘 배울 것

▼ 1) 프로젝트 구조 관리, 최적화, 서스펜스

▼ 1. 프로젝트 구조 관리

👉 쓰레드에 대해 들어보셨나요? 쓰레드는 쉽게 말해 일꾼이에요.
자바스크립트는 싱글 쓰레드 기반으로 동작하는 언어입니다.
다른 언어를 배워봤다면 싱글 쓰레드와 비동기를 함께 듣자마자 오잉? 하셨을 수 있어요.
일꾼이 하나인데 어떻게 비동기가 가능한 지 알아봅시다. 😄

👉 개념을 이해했다면 좀 더 단어를 정확히 해볼까요?
프로그램의 실행부터 종료까지 한 작업을 프로세스라고 해요.

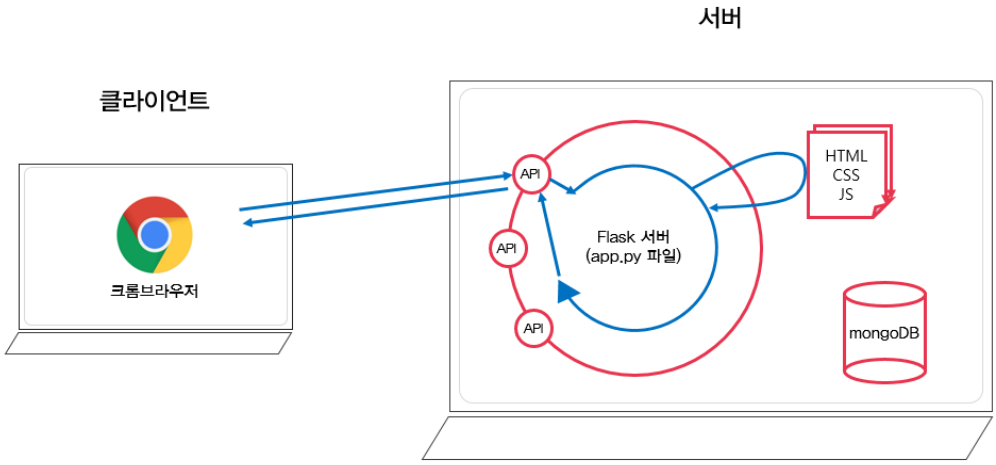


▼ 2. 최적화

👉 네! 우리가 보는 웹페이지는 모두 서버에서 미리 준비해두었던 것을 "받아서",
"그려주는" 것입니다. 즉, 브라우저가 하는 일은 1) 요청을 보내고, 2) 받은 HTML 파일을 그려주는 일 뿐이죠.

👉 근데, 1)은 어디에 요청을 보내냐구요? 좋은 질문입니다. 서버가 만들어 놓은 "API"라는 창구에 미리 정해진 약속대로 요청을 보내는 것이랍니다.

예) https://naver.com/
→ 이것은 "naver.com"이라는 이름의 서버에 있는, "/" 창구에 요청을 보낸 것!



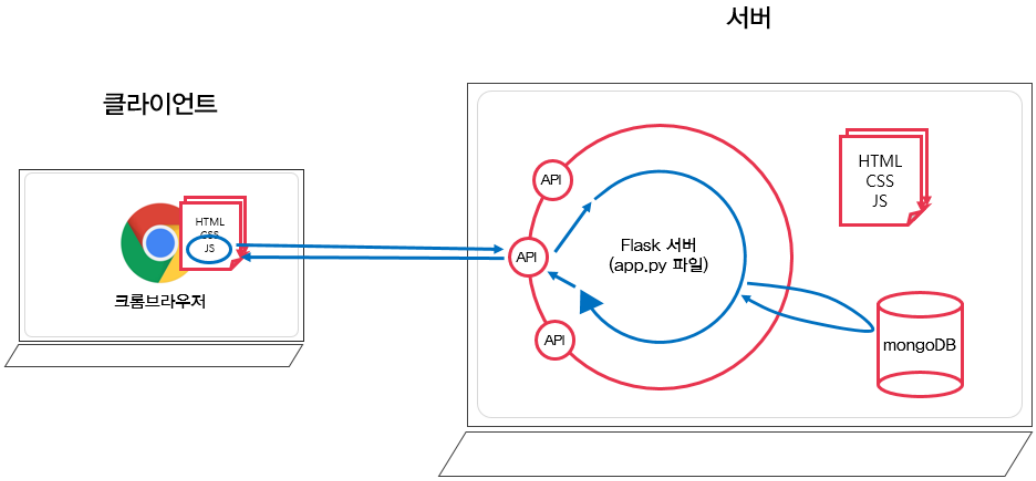
▼ 3. 전역화

👉 앗, 그럼 항상 이렇게 HTML만 내려주냐구요?
아뇨! 데이터만 내려 줄 때가 더~ 많아요.

사실 HTML도 줄글로 쓰면 이게 다 '데이터'아닌가요?

👉 자, 공연 티켓을 예매하고 있는 상황을 상상해봅시다!
좌석이 차고 꺼질때마다 보던 페이지가 리프레시 되면 난감하겠죠ㅠ?

이럴 때! 데이터만 받아서 받아 끼우게 된답니다.



👉 데이터만 내려올 경우는, 이렇게 생겼어요!
(소곤소곤) 이런 생김새를 JSON 형식이라고 한답니다.

openapi.seoul.go.kr:8088/6d4d7

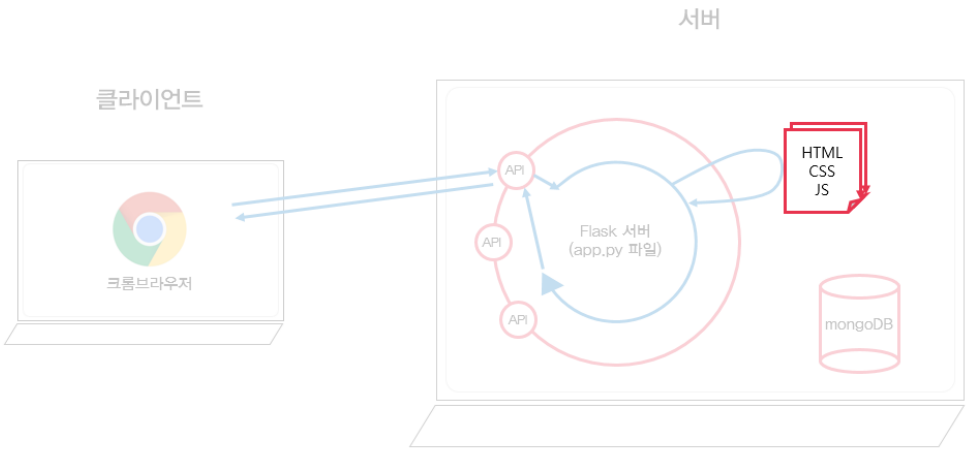
주의 요함 | openapi.seoul.go.kr:8088/6d4d776b466c656533356a4b4b5872/json/RealtimeCityAir/1/99


```
{
  - RealtimeCityAir: {
        list_total_count: 25,
      - RESULT: {
            CODE: "INFO-000",
            MESSAGE: "정상 처리되었습니다"
        },
      - row: [
          - {
                MSRDT: "202004241900",
                MSRRGN_NM: "도심권",
                MSRSTE_NM: "중구",
                PM10: 44,
                PM25: 20,
                O3: 0.039,
                NO2: 0.02,
                CO: 0.4,
                SO2: 0.003,
                IDEX_NM: "보통",
                IDEX_MVL: 59,
                ARPLT_MAIN: "PM10"
            },
          - {
                MSRDT: "202004241900",
```


▼ 4. 서버 컴포넌트

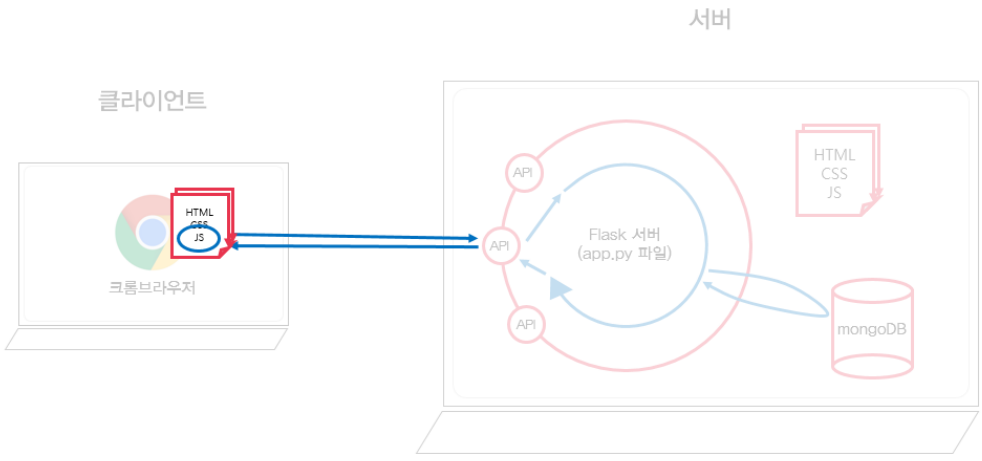
▼ 1주차: HTML, CSS, Javascript

👉 오늘은 HTML과 CSS를 배우는 날! 즉, 4주차에 내려줄 HTML파일을 미리 만들어 두는 과정입니다. + 또, 2주차에 자바스크립트를 능숙하게 다루기 위해서, 오늘 문법을 먼저 조금 배워둘게요!



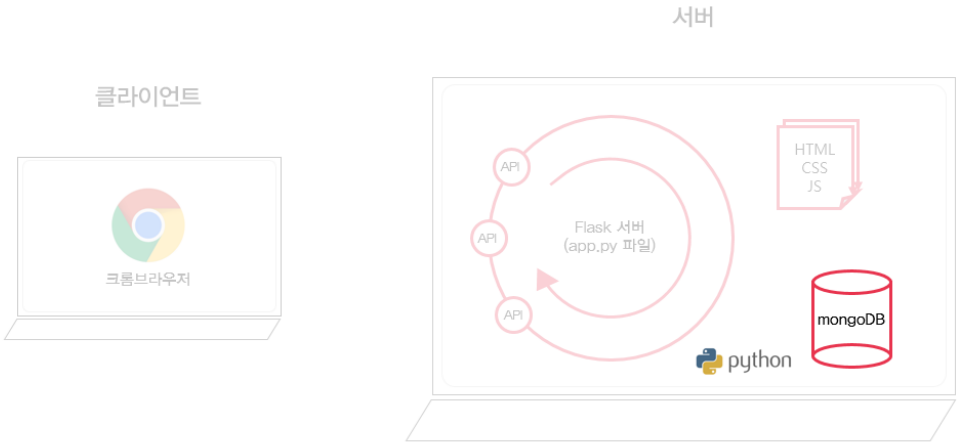
▼ 2주차: jQuery, Ajax, API

👉 오늘은 HTML파일을 받았다고 가정하고, Javascript로 서버에 데이터를 요청하고 받는 방법을 배워볼거예요



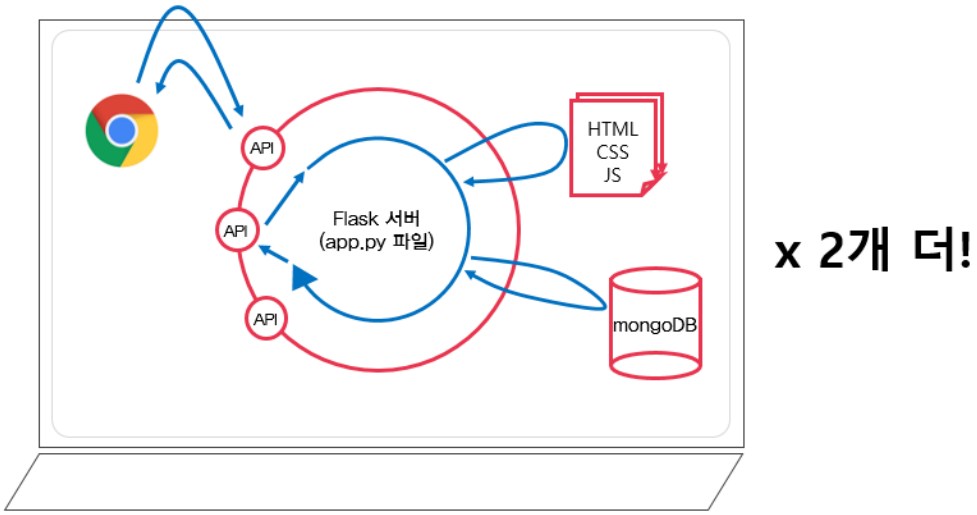
▼ 3주차: Python, 크롤링, mongoDB

👉 오늘은 드디어 '파이썬'을 배울거예요. 먼저 문법을 연습하고, 라이브러리를 활용하여 네이버 영화목록을 쫙 가져와보겠습니다. (기대되죠!)
+
그리고, 우리의 인생 첫 데이터베이스. mongoDB를 다뤄볼게요!



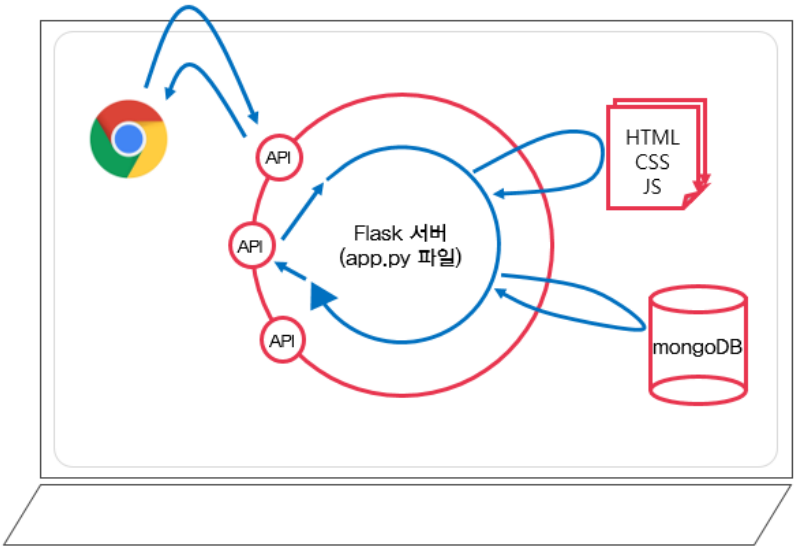
▼ 4주차: 미니프로젝트1, 미니프로젝트2

👉 오늘은 서버를 만들어봅니다! HTML과 mongoDB까지 연동하여, 미니프로젝트1, 2를 완성해보죠! 굉장히 재미있을 거예요!



👉 나중에 또 이야기하겠지만 헷갈리면 안되는 것!
우리는 컴퓨터가 한 대 잖아요... 그래서 같은 컴퓨터에다 서버도 만들고, 요청도 할 거예요. 즉, 클라이언트 = 서버가 되는 것이죠.

이것을 바로 "로컬 개발환경"이라고 한답니다! 그림으로 보면, 대략 이렇습니다.

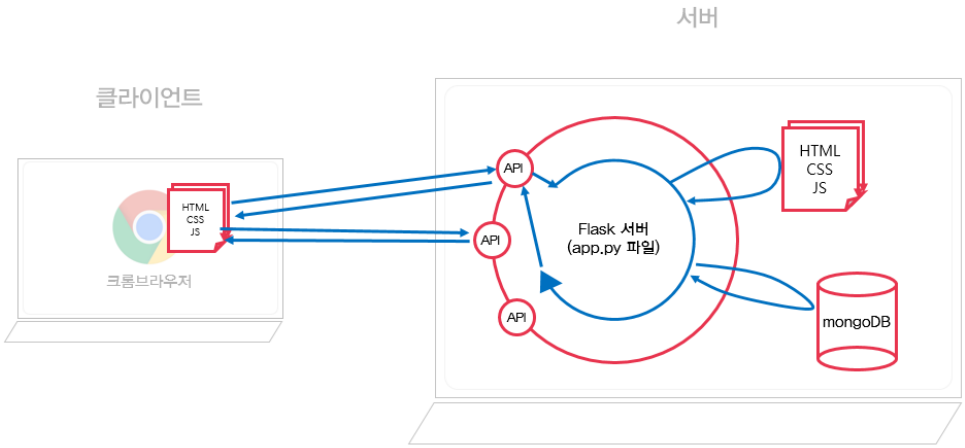


▼ 5주차: 미니프로젝트3, AWS

👉 오늘은 아직 익숙해지지 않았을 당신을 위해! 같은 난이도의 유사한 한 개의 프로젝트를 더 진행합니다.

그치만 우리 컴퓨터를 24시간 돌려둘수는 없잖아요!

그래서 두구두구.. 인생 첫 배포!를 해볼 예정입니다! 클라우드 환경에서 컴퓨터를 사고, 거기에 파일을 올려 실행해보겠습니다.



▼ 5. 서스펜스

02. 컴포넌트 관리

👉 **프로젝트 시작 전! 꼭 생각해야 하는 것 :**

1. 컴포넌트, 어떻게 해야 여러번 써먹을 수 있을까? (=재사용 가능할까?)
2. API, 어떻게 해야 같은 후처리를 반복하지 않을 수 있을까?

▼ 2) 합성과 상속

👉 상속(Inheritance)
상속은 말 그대로 물려받는 것입니다. 부모에서 자식으로 어떤 데이터나 함수를 받아서 사용하는 거예요.

👉 합성(Composition)
합성은 서로 다른 컴포넌트를 합쳐서 사용하는 걸 말해요.
모달 창을 만든다고 생각해봅시다. 어떤 모달은 제목과 예/아니오 버튼이 필요할 수 있고, 또 어떤 모달은 버튼없이 제목과 내용만 필요할 수 있어요.
이렇게 같은 틀을 가지고는 있지만 내용물 차이가 나는 경우 부모에서 자식으로 값을 내려주다보면 없는 정보를 너무 많이 넘겨줘야합니다.
이럴 때에 각각 제목, 내용, 버튼 컴포넌트를 만들고 합성을 통해 필요한 내용만 넘기는 식으로 컴포넌트를 꾸리면 좀 더 깔끔하게 관리할 수 있습니다.

- 리액트는 children prop을 사용해서 자식 엘리먼트를 받아와 쓸 수 있습니다. 아래처럼요!

```
const A = () => {
  return <p>hello!</p>;
}

const B = (props) => {
  return (
    <div>
      {props.children}
    </div>
  );
}

const Greeting = (props) => {
  return (
    <B>
      <A/>
      <p>welcome!</p>
    </B>
  );
}
```

▼ 3) IoC

👉 **Inversion of Control(IoC): 제어 역전**
컴포넌트를 사용하는 유저에게 주어지는 유연성(flexibility)와 제어(control)의 정도를 말합니다.
일반적으로 상위 → 하위로 명령을 전달한다면, 하위 → 상위로 명령을 전달하게 되는 게 IoC입니다.

- 다양한 원시값으로 이뤄진 배열 중, 특정 자료형만 골라서 다 더해주는 함수 A가 있다면?
  - 특정 자료형만 다 더해주기 위해 필요한 것 : 1) 파라미터로 받아온 값의 자료형이 뭔 지 확인하는 로직 2) 값끼리 더해주는 로직
  - 이 함수를 쓸 때는 다양한 원시값이 든 배열과, 어떤 자료형일 때 더해줄 건 지에 대한 값을 넘겨주어야 합니다.
  - 함수 A를 만든 개발자가 빼먹은 자료형이 있다면 오류가 날 수 있어요.
  - 함수 A는 배열에 없는 자료형까지 모두 확인하는 로직이 있어야 해요.
- 함수 A를 호출하기 전에 미리 특정 자료형만 골라둔다면?
  - 함수 A의 1)로직은 없어져도 괜찮습니다!
  - 함수 A에서 해야할 일을 미리 해두고 함수 A를 호출한다면 함수 A가 할 일은 더하기 뿐이니, 더한다는 임무에만 충실하면 됩니다.

→ 이런 식으로 함수 A가 제어하던 것을 다른 위치에서 제어하는 것이 IoC입니다.

▼ 4) 잘만들어진 패키지는 어떻게 컴포넌트를 관리하고 있나?

> 👉 리액트를 사용할 때 자주 쓰는 패키지를 뜯어보면서 알아봅시다!

▼ React Bootstrap

> 👉 Compound Components Pattern을 씁니다.
> 불필요한 props drilling을 없애주는 패턴이에요.
> (문서 보러 가기→)

- 하나의 거대한 부모가 있고 그 부모에게 모든 props를 넘겨 자식 컴포넌트한테 내려주는 방식이 아닙니다! → props가 각각 서브 컴포넌트에 붙어 있어요.
- 모든 props를 외울 필요가 없어요. (API 복잡도가 낮아요!)
- 마크업 구조가 유연합니다.
- UI 자유도가 높은 편이지만, 컴포넌트 간 구조를 모른다면 사용이 어려울 수 있습니다.

▼ material-ui

> 👉 Control Props Pattern을 씁니다.
> 컴포넌트의 데이터를 컴포넌트 바깥에서 조절할 수 있게 해주는 패턴이에요.
> (문서 보러 가기→)

- 컴포넌트를 불러와서 어떤 데이터를 직접 넣고 관리하며 사용할 수 있어요. → 이런 걸 두고 state가 컴포넌트 바깥에 드러났다고 합니다!
- 컴포넌트를 불러다 쓸 때 그 컴포넌트에 넣을 데이터와 데이터 수정 방식을 직접 지정할 수 있으니 컴포넌트 데이터 제어가 쉬워요.
- 컴포넌트 바깥에 useState등을 만들어두고 데이터를 제어해야 하니, 데이터 관리하는 부분과 데이터를 보여주는 부분이 분리되어 코드가 지저분해지기 쉽습니다.

▼ react-hook-form

> 👉 Custom Hook Pattern을 씁니다.
> 데이터 관리 등 메인 로직은 커스텀 훅으로 관리하는 패턴이에요.
> (문서 보러 가기→)

- 여기서부터는 사용법이 정말 까다롭습니다! ㅠㅠ
- 커스텀 훅이 어떤 동작을 하는 지 명확히 이해하지 못한다면 같은 팀원이 아예 손도 못댈 수 있거든요.
- 메인 로직이 뷰와 분리되어 있어서 사용할 때 둘을 이어주는 작업을 해주어야 해요. 컴포넌트 동작을 정확히 알아야만 제대로 쓸 수 있게 됩니다.
- 훅으로 로직을 묶어두어서 기능과 뷰가 깔끔하게 나뉩니다.

03. Meta tag

> 👉 메타태그는 웹페이지의 제목이나 이미지, 간단한 설명을 검색엔진에 알려주는 역할을 해요. 🙂

▼ 5) react-helmet 설치하기

```
yarn add react-helmet
```

▼ 6) helmat으로 메타 태그 바꾸기

```
//One.js
import React from "react";
import {Helmet} from "react-helmet";
const One = (props) => {
    return (
      <div>
        <Helmet>
          <title>page one</title>
          <meta property="og:title" content="page one" />
          <meta property="og:description" content="hi there :) page one" />
          <meta property="og:image" content="%PUBLIC_URL%/logo192.png" />
        </Helmet>
        <h2>Hi, there :) ! page one</h2>
        <button
          onClick={() => {
            props.history.push("/");
          }}
        >
          page one
        </button>
        <button
          onClick={() => {
            props.history.push("/two");
          }}
        >
          page two
        </button>
      </div>
    );
}

export default One;
```

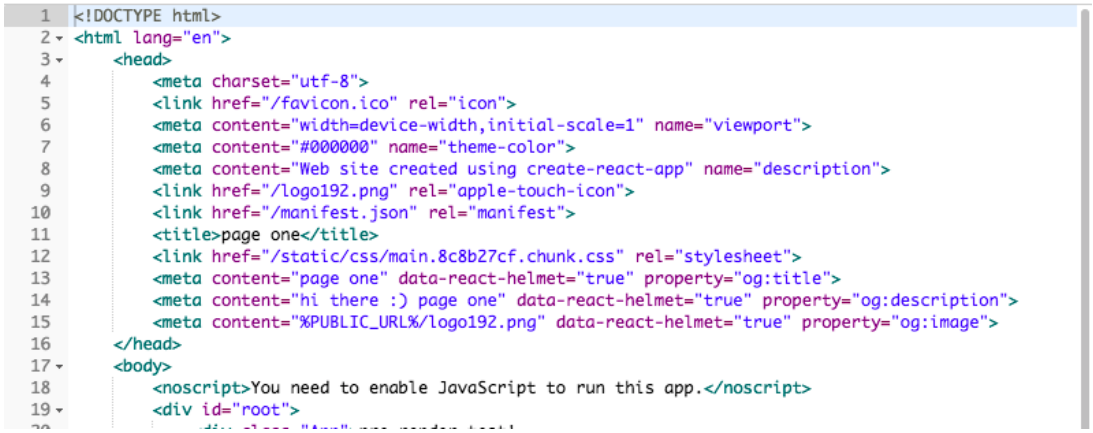
▼ 7) 확인해보자!

👉 빌드 먼저 하고, post man에서 확인해봅시다!

- 빌드하기

```
yarn build
```

- post man에서 확인하기



```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <link href="/favicon.ico" rel="icon">
        <meta content="width=device-width,initial-scale=1" name="viewport">
        <meta content="#000000" name="theme-color">
        <meta content="Web site created using create-react-app" name="description">
        <link href="/logo192.png" rel="apple-touch-icon">
        <link href="/manifest.json" rel="manifest">
        <title>page one</title>
        <link href="/static/css/main.8c8b27cf.chunk.css" rel="stylesheet">
        <meta content="page one" data-react-helmet="true" property="og:title">
        <meta content="hi there :) page one" data-react-helmet="true" property="og:description">
        <meta content="%PUBLIC_URL%/logo192.png" data-react-helmet="true" property="og:image">
    </head>
    <body>
        <noscript>You need to enable JavaScript to run this app.</noscript>
        <div id="root">
```

04. 성능 지표 보기

👉 CRA를 통해 프로젝트를 만들면 기본적으로 설치되는 것들이 있죠!
그 중 하나가 webVitals입니다!
CRA 4 버전부터 새로 추가되었어요.
추가된 김에 이 webVitals를 써봅시다.😉

▼ 8) webVitals 써보기

▼ report를 콘솔에 찍어서 확인해보자

```
//index.js
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './shared/App';
import reportWebVitals from './reportWebVitals';
import store from "./redux/configureStore";

import { Provider } from "react-redux";

ReactDOM.render(
  <Provider store={store}>
    <App />
  </Provider>,
```

```
  document.getElementById("root")
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals(console.log);
```

▼ 어딘가에 보내고 싶다면?

```
// 함수를 만들고,
function sendToAnalytics(metric) {
  const body = JSON.stringify(metric);
  const url = 'https://example.com/analytics';

  // Use `navigator.sendBeacon()` if available, falling back to `fetch()`
  if (navigator.sendBeacon) {
    navigator.sendBeacon(url, body);
  } else {
    fetch(url, { body, method: 'POST', keepalive: true });
  }
}

// console.log 대신 넘겨줘요!
reportWebVitals(sendToAnalytics);
```

▼ 9) firebase analytics 연결하기

▼ analytics 연결하기

```
...
import "firebase/analytics";

const firebaseConfig = {
  // 인증 정보!
};
...

const analytics = firebase.analytics();

export{auth, apiKey, firestore, storage, realtime, analytics};
```

▼ 리포트 보내기

```
//index.js
...
import { analytics } from "./shared/firebase";
...
function sendToAnalytics(metric) {
```

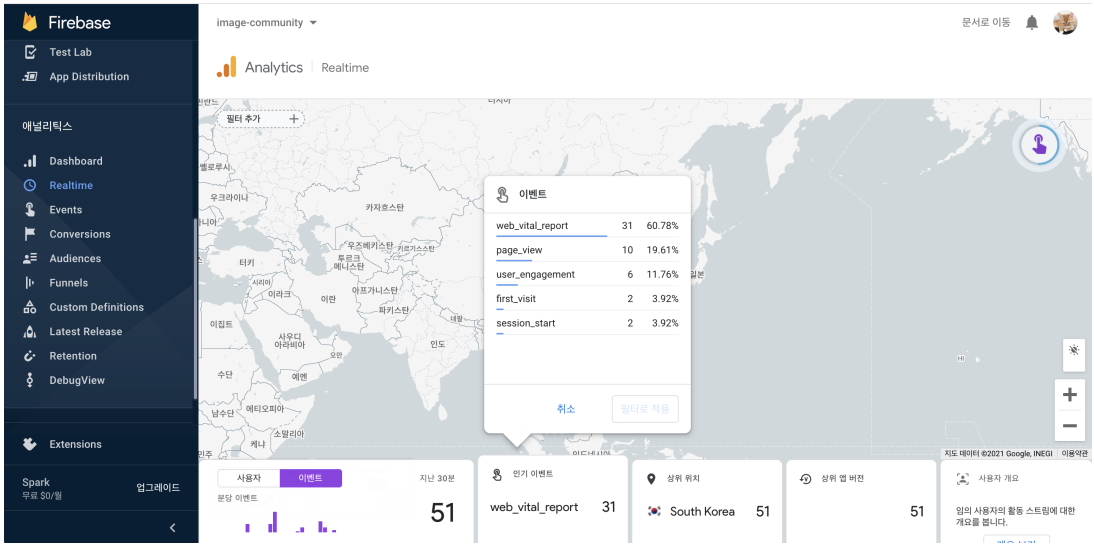
```
  const _report = JSON.stringify(metric);

  analytics.logEvent("web_vital_report", _report);

  console.log({ _report });
}
reportWebVitals(sendToAnalytics);
```

▼ 확인하기

👉 대시보드에서 확인해볼까요! 🙂



05. React.memo()

👉 **부모가 바뀔 때 자식은 왜 꼭 렌더링 되어야만 할까? 🤔**
부모 컴포넌트가 바뀔 때마다 바뀔 게 없는 컴포넌트까지 다시 렌더링하는 걸 막아주는 방법이 있으면 참 좋을 것 같죠?
→ 사실 있습니다! 컴포넌트를 렌더링하고, 결과를 메모이제이션해두는 거예요!

React.memo를 사용해서 할 수 있어요!

▼ 10) React.memo()

👉 useMemo가 렌더링 때마다 연산하지 않도록, 연산된 값을 재 사용하는 훅이라면 memo는 컴포넌트의 리렌더링을 방지하는 함수예요!

▼ 써보기

```
// components/Post.js

import React from "react";
import { Grid, Image, Text, Button } from "../elements";

import {history} from "../redux/configureStore";

 const Post = React.memo((props) => {
   console.log("in post");
   return (
     <React.Fragment>
       ...
     </React.Fragment>
   );
 });
...
```

💡 간단한 연산을 수행할 때는 메모이제이션에 쓰는 비용이 연산 비용보다 큽니다. 이 경우 메모이제이션 없이 사용하는 편이 좋아요.

06. Portal

▼ 11) portal

👉 Portal이란?
우리의 root element! 기억하시나요? 😊
우리는 루트 엘리먼트에 모든 컴포넌트를 올리고 지우고 올리고 지우고 반복하며 컴포넌트를 만들었어요.
포탈은 루트 외의 요소에 컴포넌트를 띄울 수 있게 해주는 거예요!

- 사용법

```
ReactDOM.createPortal(child, container)
```

- 써보자!
  - 포탈로 쓸 div를 만들고,

```
<div id="root"></div>
<div id="portal"></div>
```

  - 포탈 만들기

```
// ReactPortal.js
import { createPortal } from 'react-dom';

function ReactPortal({ children }) {
  return createPortal(children, document.getElementById("portal"));
}
export default ReactPortal;
```

  - 불러쓰자!

```
// App.js
import React from "react";
import logo from "./logo.svg";
import "./App.css";
import ReactPortal from "./ReactPortal";

function App() {

  return (
    <div className="App">
      <header className="App-header">
        <img src={logo} className="App-logo" alt="logo" />


        <ReactPortal>
          <p>안녕하세요! 저 여기 있어요!</p>
        </ReactPortal>
      </header>
    </div>
  );
}
```

  - 브라우저에서 확인해보기

07. 상태관리 ++ - react-query 1

▼ 12) 상태의 종류

> 👉 **상태의 종류**
> 프론트엔드에서 다루는 상태는 크게 나누면 두 종류가 있어요! 🙂
> 클라이언트 상태와 서버 상태지요!

- 클라이언트 상태 : 뷰를 위한 데이터
  - 한 유저만을 위한 데이터 (세션간 지속될 필요없는 데이터)
    예시) input 입력 값을 state로 관리(렌더링에 반영하기 위한 데이터)
- 서버 상태 : 서버에서 가지고 오는 데이터
  - 여러 유저가 공유해야하는 데이터 (세션간 지속되어야 하는 데이터)

    예시) 게시물 리스트(DB에 저장되어 있는 데이터)

▼ 13) React Query

> 👉 React-query
> 서버 상태를 잘 관리하기 위한 상태 관리 라이브러리예요!
>
> 우리가 어떤 데이터를 서버에 요청하고 나서 부터 요청을 받은 후까지 데이터를 받아오기 전까지 참조 못하게 하기, 게시글 목록 중 한 데이터 수정 api를 호출했다면 게시글 목록 자체를 리패칭하기 등 직접할 일이 참 많습니다.
>
> 리액트 쿼리는 이런 비동기 처리를 편히 할 수 있도록 굉장히 많은 기능을 제공해요. 😊
> **(공식문서 바로가기→)**

▼ React Query 상태 관리 흐름

- fetching : 데이터 요청 중
- fresh : 데이터를 갓 받아온 직후 / 컴포넌트의 상태가 변하더라도 데이터 재요청 하지 않음
- stale : 데이터 만료 / 최신화가 필요한 데이터

- inactive : 쿼리가 언마운트 된 상태 (더는 사용하지 않는 상태)
  !주의!
  쿼리가 언마운트된다고 해서 비동기 요청이 취소되는 것은 아닙니다. 프라미스가 일단 만들어지고 언마운트된 거라면 데이터는 캐시에 살아 있을 수 있습니다.
  (공식문서에서 확인하기→)
- delete : 완전히 삭제된 상태(캐시 데이터가 메모리에서 삭제!)

▼ 주요 개념

- query : query key + query function
- query key : 쿼리를 구분하기 위한 특정 값. 문자, 배열, 딕셔너리 등을 넣을 수 있음
- query function : 서버에서 데이터를 요청하고 Promise를 리턴하는 함수(= 비동기 요청)
- data : 쿼리 함수가 리턴한 Promise에서 resolve된 데이터
- staleTime : 쿼리 데이터가 fresh 에서 stale로 전환되는데 걸리는 시간. 기본값은 `0`
- cacheTime : unused 또는 inactive 캐시 데이터가 메모리에서 유지될 시간 기본값은 5분이며 설정한 시간을 초과하면 메모리에서 제거

> 🔥 **Parallel queries(병렬 쿼리)는 후에 배워볼 suspense 모드에서 동작하지 않아요!**
> 첫번째 쿼리가 프라미스를 만들면(서버 요청 등 비동기 요청을 보내면 프라미스가 만들어져요!) 다른 쿼리가 실행되기 전에 서스펜스가 동작을 일시 중지시켜버리기 때문인데요,

08. 상태관리 ++ - react-query 2

▼ 14) 필요 패키지 설치부터!

👉 전에 만들어둔 mock API를 사용해서 요일 별 수면 시간을 가져오고, 추가해 볼 거예요.
통신을 위해 axios를, 상태 관리를 위해 react-query를 설치하고 시작합시다!

- axios 설치

```
yarn add axios
```

- react query 설치

```
yarn add react-query
```

▼ 15) QueryClientProvider 설정하기

👉 QueryClient는 QueryClientProvider와 한쌍입니다.
상태를 주입할 Provider로 감싸주고,
쿼리 클라이언트를 주입해요!

- QueryClient / QueryClientProvider

```
import React from "react";
import ReactDOM from "react-dom/client";
import "./index.css";
import App from "./App";
import reportWebVitals from "./reportWebVitals";

import { QueryClient, QueryClientProvider } from "react-query";
import { ReactQueryDevtools } from "react-query/devtools";

const queryClient = new QueryClient();

const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(
  <QueryClientProvider client={queryClient}>
    {/* devtools */}
    <ReactQueryDevtools initialIsOpen={true} />
    <App />
  </QueryClientProvider>
);

// If you want to start measuring performance in your app, pass a function
```

```
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();
```

▼ 16) 첫번째 query instance 만들기 : useQuery!

👉 쿼리 인스턴스를 만들어봅시다.

▼ useQuery 사용법

👉 useQuery는 어떤 데이터를 가져와서 캐싱하기 위해 씁니다!
쿼리 키, 쿼리 함수, 옵션으로 구성되어 있어요.
- 쿼리 키 : 문자열 / 배열 등을 넘길 수 있습니다. 이 키는 캐싱 처리하는 데에 사용해요. 우리가 불러다 쓰기 위해서도 중요합니다.
- 쿼리 함수 : Promise를 리턴하는 어떤 함수를 넘깁니다. api와 통신하는 함수가 주로 들어가요.
- 옵션 : useQuery()를 위한 옵션들을 넣어주면 됩니다!

```
const my_first_query = useQuery(쿼리 키, 쿼리 함수, 옵션);
```

▼ my_first_query에는 아래같은 데이터가 담겨요.

- data : 서버 요청에 대한 데이터
- isLoading : 캐시가 없는 상태에서의 데이터 요청 중인 상태 (true / false)
- isFetching : 캐시의 유무 상관없이 데이터 요청 중인 상태 (true / false)
- isError : 서버 요청 실패에 대한 상태 (true / false)
- error : 서버 요청 실패 (object)
- refetch : 데이터를 새로 패칭하는 함수

▼ 넘길 수 있는 옵션 중 알아두면 좋을 내용입니다.

- cacheTime : 언마운트된 후 어느 시점까지 메모리에 데이터를 저장하여 캐싱할 것인지를 결정
  기본 값 : 30000(5분)

- staleTime : 쿼리가 fresh 상태에서 stale 상태로 전환되는 시간
  기본 값 : 0
- refetchOnMount : 컴포넌트 마운트시 새로운 데이터 패칭
  기본 값 : true (false일 경우엔 마운트해도 새로운 데이터를 가지고 오지 않아요.)
- refetchOnWindowFocus : 브라우저 클릭 시 새로운 데이터 패칭
  기본 값 : true (false일 경우엔 다른 탭으로 갔다가 돌아와도 데이터를 새로 가져오지 않아요.)
- refetchInterval : 지정한 시간 간격만큼 데이터 패칭 / 브라우저 바깥에 있을 경우(다른 탭에 있는 등) 실행되지 않아요!
  기본 값 : 0
- refetchIntervalInBackground : 브라우저에 포커스가 없어도 refetchInterval에서 지정한 시간 간격만큼 데이터 패칭
  기본 값 : false
- enabled : 컴포넌트가 마운트 되어도 데이터 패칭 ❌
  기본 값 : true
  useQuery의 리턴 값 중 refetch가 있었죠! enabled가 true일 때는 refetch를 통해 데이터를 패칭해야 합니다.
- onSuccess : 데이터 패칭 성공 시 콜백 함수
- onError : 데이터 패칭 실패 시 콜백
- initialData : 초기 데이터
  쿼리 생성 전이나 아직 캐싱되지 않았을 경우, 이 데이터를 초기 데이터로 써요!
- select : 데이터 패칭 성공 시 가져온 데이터를 변환해주고 싶을 때, 원하는 데이터 형식으로 변환하기 위한 콜백 (여기서 return으로 변환한 데이터를 반환해주면 useQuery의 결과값인 data가 변해요.)

▼ 수면 기록들을 가져와보자!

```
// App.js

import React from "react";
import "./App.css";
import { useQuery } from "react-query";
import axios from "axios";

// promise를 반환합니다!
```

```
// mock api에서 sleepList를 가져와요
const getSleepList = () => {
  return axios.get("http://localhost:5001/sleep_times");
};

function App() {
  const sleep_query = useQuery("sleep_list", getSleepList, {
    onSuccess: (data) => {
      console.log("성공했어!", data);
    },
  });

  console.log(sleep_query);
  console.log(sleep_query.data);

  return (
    <div className="App">
     <div>
        {sleep_query.data.data.map((d) => {
          return (
            <div>
              <p>{d.day}</p>
              <p>{d.sleep_time}</p>
            </div>
          );
        })}
      </div>
    </div>
  );
}

export default App;
```

👉 useQuery()를 사용해 데이터를 가져오는 목록 컴포넌트를 만들고, 만든 목록 컴포넌트를 여러개 불러온다면 어떻게 될까요?
컴포넌트를 5개 불러온다면, useQuery()로 쿼리 인스턴스를 5개 만들게 됩니다.
이 경우, getSleepList는 딱 한 번만 호출되겠지만, 옵션으로 넣어준 onSuccess는 5번 실행될거예요.
select 옵션처럼 값을 바꿔주는 경우라면? 마찬가지로 5번 실행될테니 주의해서 사용해야 해요!
특히 커스텀 훅으로 사용하실 경우에는 더더욱 주의하셔야 합니다.

09. 상태관리 ++ - react-query 3

▼ 17) useMutation 사용법

👉 useMutation은 어떤 데이터를 변경할 때(생성/수정/삭제할 때!) 사용합니다.
useQuery()와 같은 리턴 값 + mutate()를 리턴 해요.
뮤테이션 함수 :
useMutation은 정말 많은 옵션이 있어요!
공식문서 링크를 걸어둘테니 꼭 문서에서 옵션을 확인하세요! (공식문서 보러가기→)

컴포넌트가 언마운트되면?
→ onSuccess, onError같은 추가 콜백이 실행되지 않아요!

👉 가장 중요한 옵션 하나만 짚어볼게요!
onMutate: mutate 함수가 실행되기 전 실행하는 함수 / mutation 함수가 받을 변수가 전달됩니다.

```
const my_first_mutation = useMutation(뮤테이션 함수, 옵션);
```

▼ my_first_mutation에는 아래같은 데이터가 담겨요.

- data : 서버 요청에 대한 데이터
- isLoading : 캐시가 없는 상태에서의 데이터 요청 중인 상태 (true / false)
- isFetching : 캐시의 유무 상관없이 데이터 요청 중인 상태 (true / false)
- isError : 서버 요청 실패에 대한 상태 (true / false)
- error : 서버 요청 실패 (object)
- **mutate(useMutation의 콜백으로 넘겨줄 데이터, 옵션) : 뮤테이션을 실행해 줄 함수**

▼ 18) 수면 기록을 추가해보자!

👉 가장 중요한 옵션 하나만 짚어볼게요!
onMutate: 뮤테이션 함수가 실행되기 전 실행하는 함수 / mutation 함수가 받을 변수가 전달됩니다.

```
// App.js

import React from "react";
import "./App.css";
import { useMutation, useQuery } from "react-query";
import axios from "axios";

// promise를 반환합니다!
// mock api에서 sleepList를 가져와요
const getSleepList = () => {
  return axios.get("http://localhost:5001/sleep_times");
};

// 데이터 추가 api 요청을 반환해요!
const addSleepData = (data) => {
  return axios.post("http://localhost:5001/sleep_times", data);
};

function App() {

  // ref를 잡아줘요.
  const day_input = React.useRef();
  const time_input = React.useRef();

  const sleep_query = useQuery("sleep_list", getSleepList, {
    onSuccess: (data) => {
      console.log("성공했어!", data);
    },
  });

  console.log(sleep_query);
  console.log(sleep_query.data);

// useMutation을 써줍시다.
  const { mutate } = useMutation(addSleepData);


  if (sleep_query.isLoading) {
    return null;
  }


// 입력 인풋 두개와 버튼도 만들어봐요.
  return (
    <div className="App">
      <div>
        {sleep_query.data.data.map((d) => {
          return (
            <div>
              <p>{d.day}</p>
              <p>{d.sleep_time}</p>
            </div>
          );
        })}
      </div>
      <input ref={day_input} />
```

```
      <input ref={time_input} />
      <button
        onClick={() => {
          if (
            day_input.current.value === "" ||
            time_input.current.value === ""
          ) {
            return;
          }
          const data = {
            day: day_input.current.value,
            sleep_time: time_input.current.value,
          };

          mutate(data);
        }}
      >
        데이터 추가하기
      </button>
    </div>
  );
}

export default App;
```

👉 어라? Mutate한 후 화면 변화가 없네요!
왜냐하면 캐싱된 데이터는 조금도 건드려주지 않았기 때문입니다.
뮤테이트 후에 캐싱된 데이터를 새로 받아오도록 해줘야 해요.

▼ 19) 후처리도 해보자!

- invalidateQueries()를 써보자!

```
// App.js

..
import { useMutation, useQuery, useQueryClient } from "react-query";
..

// promise를 반환합니다!
// mock api에서 sleepList를 가져와요
const getSleepList = () => {
  return axios.get("http://localhost:5001/sleep_times");
};

// 데이터 추가 api 요청을 반환해요!
const addSleepData = (data) => {
  return axios.post("http://localhost:5001/sleep_times", data);
};

function App() {
```

```
  const day_input = React.useRef();
  const time_input = React.useRef();
  const sleep_query = useQuery("sleep_list", getSleepList, {
    onSuccess: (data) => {
      console.log("성공했어!", data);
    },
  });

  console.log(sleep_query);
  console.log(sleep_query.data);

  const queryClient = useQueryClient();

  const { mutate } = useMutation(addSleepData, {
    onSuccess: () => {
      // key를 넣지 않을 경우 모든 쿼리가 무효화됩니다.
      // mutation을 성공하면 수면 데이터 목록을 불러오는 useQuery를 무효화 시켜줍니다!
      queryClient.invalidateQueries("sleep_list");
    },
  });

  if (sleep_query.isLoading) {
    return null;
  }

  return (
    <div className="App">
      <div>
        {sleep_query.data.data.map((d) => {
          return (
            <div>
              <p>{d.day}</p>
              <p>{d.sleep_time}</p>
            </div>
          );
        })}
      </div>
      <input ref={day_input} />
      <input ref={time_input} />
      <button
        onClick={() => {
          if (
            day_input.current.value === "" ||
            time_input.current.value === ""
          ) {
            return;
          }
          const data = {
            day: day_input.current.value,
            sleep_time: time_input.current.value,
          };

          mutate(data);
        }}
      >
        데이터 추가하기
      </button>
    </div>
```

```
  );
}

export default App;
```

- 입력 영역도 클리어 해주자!

```
const { mutate } = useMutation(addSleepData, {
    onSuccess: () => {
      // key를 넣지 않을 경우 모든 쿼리가 무효화됩니다.
      // mutation을 성공하면 수면 데이터 목록을 불러오는 useQuery를 무효화 시켜줍니다!
      queryClient.invalidateQueries("sleep_list");
      day_input.current.value = "";
      time_input.current.value = "";
    },
  });
```

10. 서스펜스 써보기

▼ 20) 서스펜스 사용하기

> 👉 서스펜스란?
> 이전 시간에서 isLoading일 때는 return null으로 화면이 보이지 않게 가려주었습니다.
> 만약, error가 난 경우도 처리해주려면 isError로 에러났어! 같은 텍스트를 보여주는 등의 작업을 더 해야겠지요!
>
> 서스펜스는 로딩 중, 에러가 난 경우처럼 어떤 상황마다 화면 분기 처리를 할 때 편히 하기 위해 탄생했습니다.

- 서스펜스 적용하기

```
//index.js
import React from "react";
import ReactDOM from "react-dom/client";
import "./index.css";
import App from "./App";
import reportWebVitals from "./reportWebVitals";

import { QueryClient, QueryClientProvider } from "react-query";
import { ReactQueryDevtools } from "react-query/devtools";
```

```
const queryClient = new QueryClient({
  defaultOptions: {
    queries: {
      staleTime: 5 * 60 * 1000,
      suspense: true,
    },
  },
});

const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(
  <React.Suspense fallback={<div>로딩 중이에요! :)</div>}>
    <QueryClientProvider client={queryClient}>
      {/* devtools */}
      <ReactQueryDevtools initialIsOpen={true} />
      <App />
    </QueryClientProvider>
  </React.Suspense>
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();
```

- 개발자 도구의 네트워크탭으로 가서 네트워크 속도 늦춰주기
  - slow 3G로 바꿔봅시다!
- 확인해보자!

11. 끝 & 숙제 설명

완강을 축하합니다! 이번 주에 배운 내용은 테크닉에 관한 내용이었습니다. 모든 프로젝트에 똑같이 적용하는 게 아니라 더 나은 방식을 고민할 초석으로 삼아주세요.

[숙제]
이번주는 숙제가 없어요!
그래도 이번주에 배운 내용은 꼭꼭 잘 정리해두세요. 😊 써먹을데가 많을거예요.