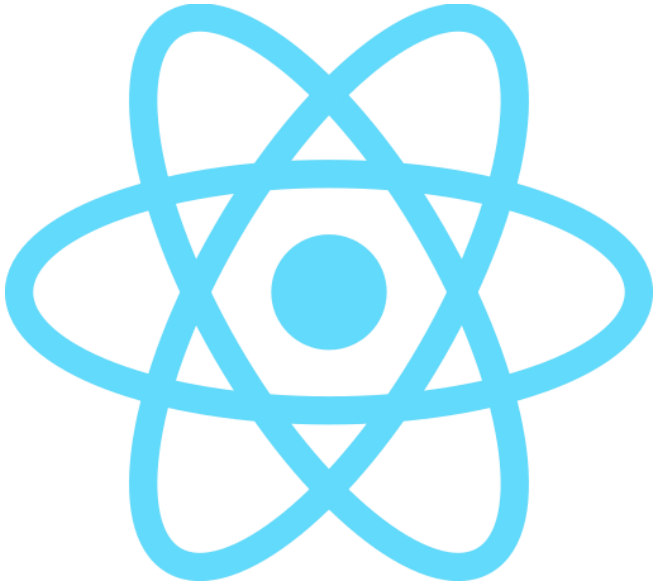


React

With Next.js

4. Api

연사 : 박승범 (@killerwhale)

연사 & 멘토 소개

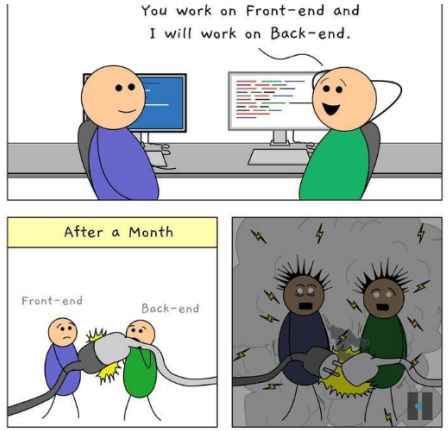
연사 : 박승범 (@killerwhale)

멘토 : 박승범 (@killerwhale)

멘토 : 박승범 (@killerwhale)

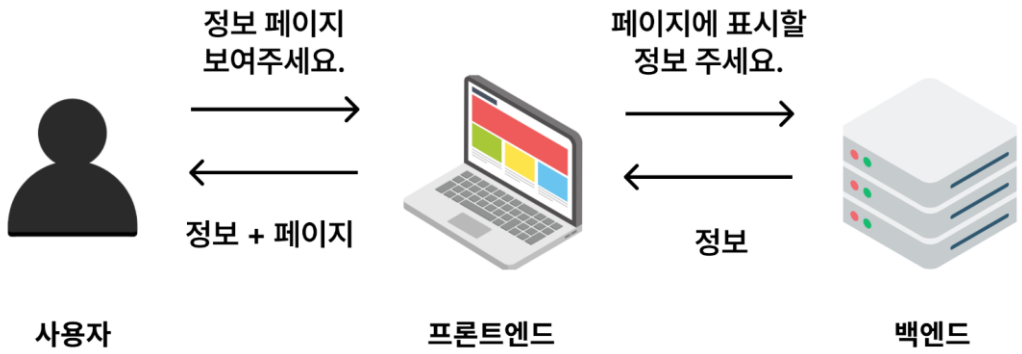
Api

API와 HTTP



- **API**(Application Programming Interface)
  : 소프트웨어 간의 **상호작용을** 가능하게 하는 **규칙**의 집합
- 프론트엔드와 백엔드는 미리 합의된 API를 바탕으로 통신한다!
- **HTTP**(HyperText Transfer Protocol)
  : 클라이언트가 서버에 데이터를 요청하거나 전송하기 위해 사용하는 프로토콜
- 웹 API는 보통 HTTP를 사용하여 요청과 응답을 주고 받는다!
- 전통적인 HTTP에서는 서버에서 먼저 데이터를 보내줄 수 없기에,
  필요한 경우 **폴링**(Polling), **Server-Sent Event**, **웹 소켓**(WebSocket)등의 기술을 사

프론트엔드와 백엔드



출처 : @static

Restful Api

API 설계 철학

객체 (대상)을 URL Routing으로, 행위를 Method로 표현

Ex) api/friends, method : get -> 친구 목록 가져오는 api
Ex) api/chat/message/{roomId}, method : Post -> 특정 채팅방에 메시지를 전송 (생성) 하는 api

<Method별 역할>

GET : 데이터 조회

POST : 데이터 생성

DELETE : 데이터 삭제

PUT / **PATCH** : 데이터 전체 수정, 데이터 일부 수정

참고) 잘못된 RESTful Api 설계 예시

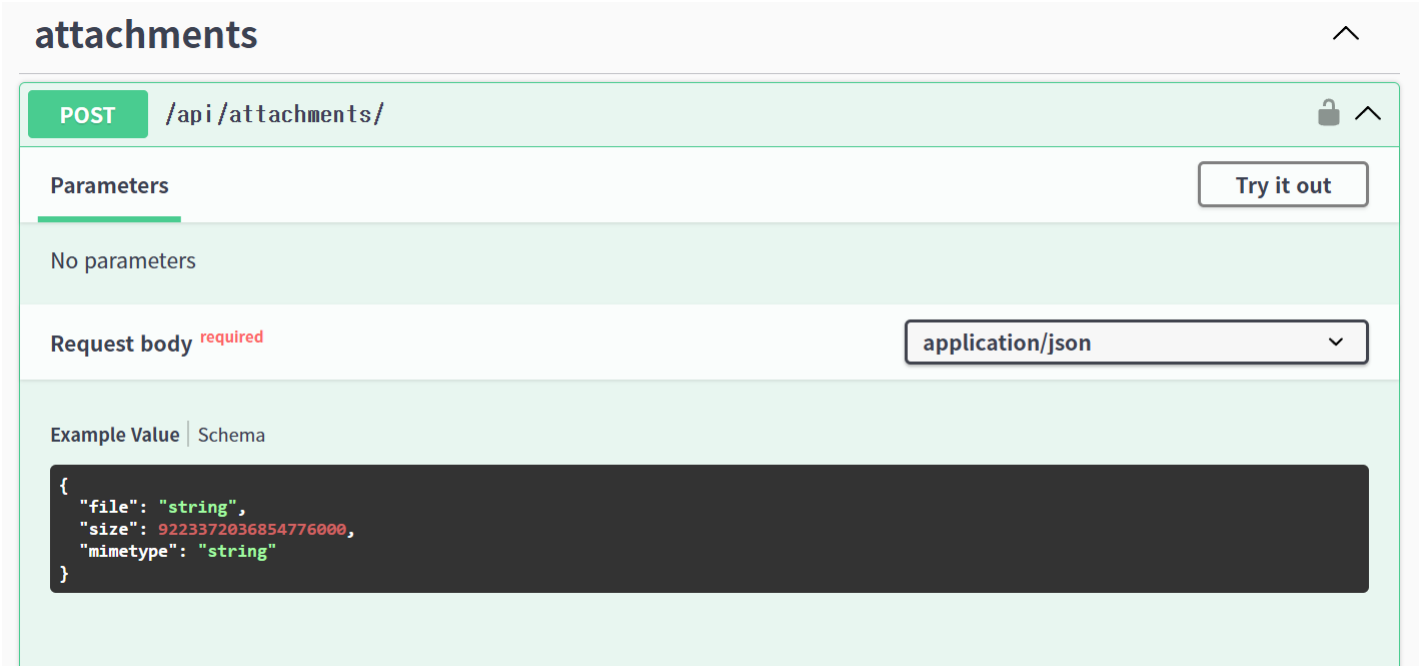
Api/get_user_by_id/{user_id} -> url에 행위가 포함되어 있음

데이터 삭제 (회원 탈퇴)를 POST Method로 받는 경우

Swagger

API의 구체적인 Spec을 정의한 문서

직접 Request Body와 Query Param을 넣어 요청을 해 볼 수도 있다.



직접 요청을 날려볼 수 있다!

Request와 Response의 형식 (타입)을 볼 수 있다.

프론트엔드 개발을 or 백엔드 테스트를 위한 유용한 도구!

fetch

브라우저에 내장된 HTTP(S) 요청을 위한 함수. 요청을 보내고 응답을 받아온다.

```
await fetch("https://api.example.com/login", {
  method: "POST",
  headers: {
    "Content-Type": "application/json",
    Authorization: "Bearer token123",
  },
  body: JSON.stringify({ username: "test", password: "1234" }),
});
```

요청을 보낼 URL, Method, Header, Request Body 를 설정하여 fetch 할 수 있다.

Return 값은 서버에서 받은 response 가 된다.

일반적으로 .json()을 활용해 json 형식으로 변환하여 사용한다.

axios

Fetch를 조금 더 쉽게 사용하기 위한 라이브러리

Base Url, **인증 관련 쿠키 관리**, 에러 처리 등을 보다 더 편리하게 할 수 있다.

```
export const apiUrl = (() => {
  if (process.env.NEXT_PUBLIC_API_HOST) {
    return process.env.NEXT_PUBLIC_API_HOST
  }

  const mode = process.env.NEXT_PUBLIC_APP_ENV
  if (mode === 'production') return 'https://newara.sparcs.org'
  if (mode === 'development') return 'https://newara.dev.sparcs.org'
  throw new Error('Unknown NEXT_PUBLIC_APP_ENV')
})()

const baseApiAddress = `${apiUrl}/api`

const http = axios.create({
  baseURL: baseApiAddress,
  withCredentials: true,
})
```

공용으로 사용되는 Base URL, 인증 관련 설정

```
// DM 생성 (userId: 상대방 user id)
export const createDM = async (userId: number) => {
    try {
        const { data } = await http.post('chat/dm/', { dm_to: userId });
        return data;
    } catch (error) {
        const err = error as AxiosError<{ detail?: string }>;

        if (err.response && err.response.data && err.response.data.detail) {
            throw new Error(err.response.data.detail);
        }
        throw new Error('DM 생성 중 오류가 발생했습니다.');
    }
}
```

Request Body (Payload)

Try-catch를 활용한 간결한 에러 처리

Async

API 요청과 응답 사이에 다른 Task 처리하기

Suspense : 동시 렌더링 (Concurrent Rendering)을 위해 도입된 개념 (React 18 +)

```jsx
import { Suspense } from 'react';
import { useQuery } from '@tanstack/react-query';

function UserInfo() {
  const { data } = useQuery({
    queryKey: ['user'],
    queryFn: () => fetch('/api/user').then((res) => res.json()),
    suspense: true,
  });

  return <p>{data.name}</p>;
}

export default function App() {
  return (
    <div>
      <h1>React Seminar 🚀</h1>
      <Suspense fallback={<p>유저 정보 불러오는 중...</p>}>
        <UserInfo />
      </Suspense>
      <button>다른 Task 실행하기</button>
    </div>
  );
}
```

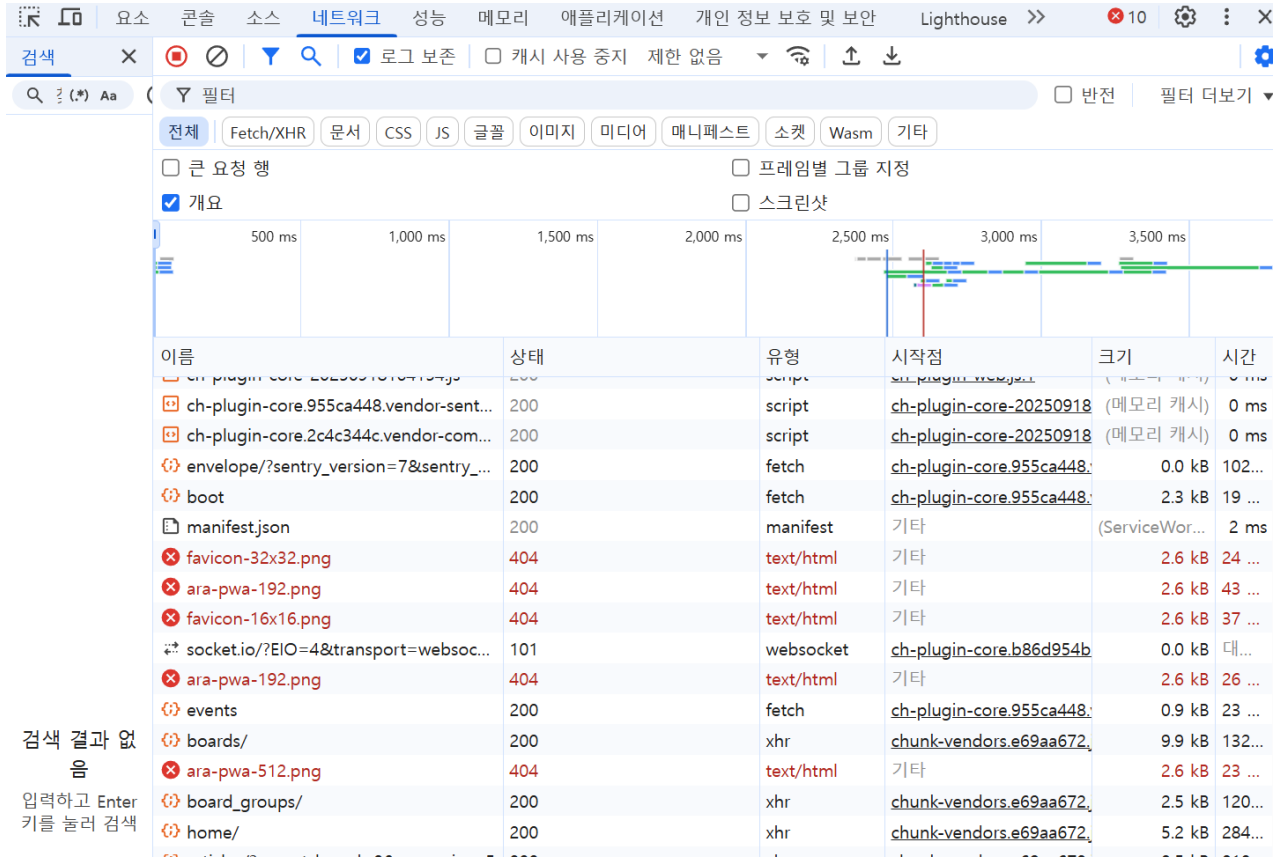
Suspense로 감싼 컴포넌트들의 렌더링이 모두 완료되면 표시

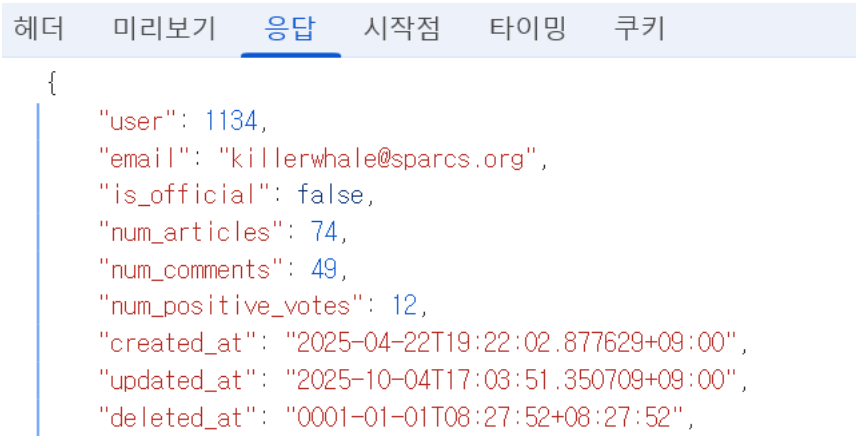
그 이전에는 fallback으로 설정된 UI 표시

Suspense는 로딩 화면 표시와 같은 곳에 활용 가능

Tip) 개발자 도구

개발자 도구를 활용하면 Api 연결과 관련된 디버깅을 쉽게 할 수 있다!





Homework

Hw4 완성하기

Due : 11/9(일)

The End

감사합니다