[개발환경] 무료 api 사용하며 Front에서 CORS 처리하기 with Proxy

구현 어려움 정도	⭐⭐
구현 완료	☑
기간	@November 29, 2024
정리 완료	☑

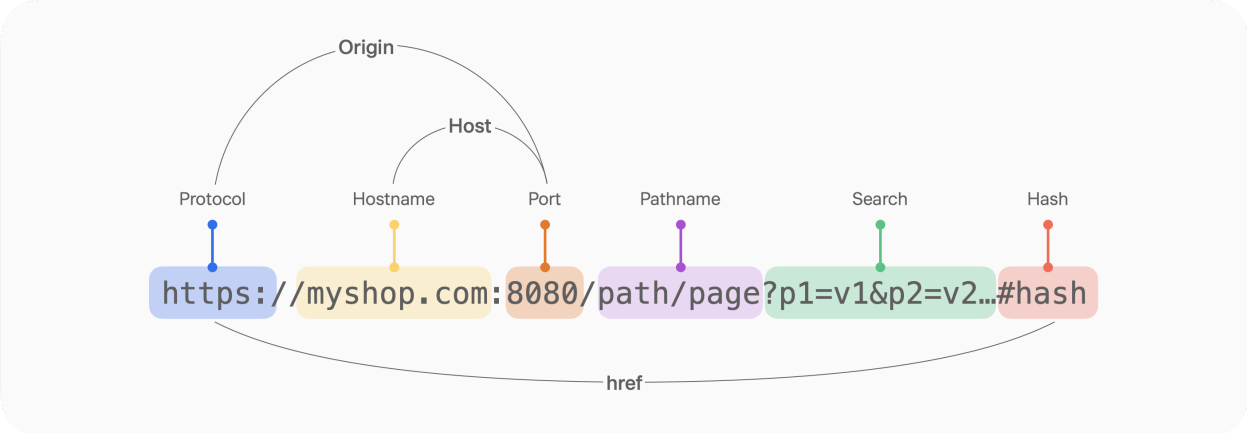
CORS

> Cross-Origin Resource Sharing(교차 출처 리소스 공유)

출처는 오리진의 번역 표현입니다. 흔히 알고 있는 URL에서 도메인만 뜻하는 것이 아닌 프로토콜과 포트까지 포함하는 개념입니다.

출처를 구성하는 세 요소는 `프로토콜`, `도메인(호스트 이름)`, `포트`로 이 중 하나라도 다르면 CORS 에러를 만나게 됩니다.

- 도메인(Hostmane): `myshop.com`
- 출처(Origin): `https://www.myshop.com`



'출처가 교차한다'는 것은 리소스를 주고받으려는 '두 출처가 서로 다르다'는 뜻입니다. CORS를 설정한다는 건 **출처가 다른 서버간의 리소스 공유를 허용한다**는 것 입니다.

▼ [참고] 동일 출처 정책 (Same-Origin Policy)

SOP 정책은 단어 그대로 동일한 출처에 대한 정책을 말합니다.

출처가 다른 두 어플리케이션이 자유롭게 소통할 수 있는 환경은 꽤 위험한 환경이라고 합니다. 제약이 없다면, 해커가 CSRF(Cross-Site Request Forgery)나 XSS(Cross-Site Scripting) 등의 방법을 이용해 우리가 만든 어플리케이션에서 해커가 심어놓은 코드가 실행되 개인정보 등을 쉽게 가로챌 수 있습니다,

사용하려는 API가 localhost:3000에서의 요청을 허용하지 않기 때문에 CORS가 발생했습니다.

일할 때 CORS 처리는 주로 백엔드(API 제공자)에게 허용하도록 요청했지만, 현재는 그럴 수 없으므로 프록시를 사용해 웹 애플리케이션에서 리소스로의 요청 전달하도록 하겠습니다.

CORS 에러 해결하기

CORS 에러를 우회하거나 외부 API와 통신하는 방법은 크게 3가지로 나눌 수 있습니다.

1. redirect를 사용하는 방법
2. next.config.ts에서 rewrites를 사용하는 방법
3. NextJS API Routes를 사용하는 방법

1. redirect를 사용하는 방법

redirect는 클라이언트의 요청을 다른 URL로 리다이렉트하는 방식입니다.

```
// next.config.ts
module.exports = {
  async redirects() {
    return [
      {
        source: '/api/proxy',
        destination: 'https://api.coingecko.com/api/v3/coins/markets',
        permanent: false,
      },
    ];
  },
};
```

클라이언트에서 `/api/proxy` 로 요청하면 NextJS가 외부 URL로 리다이렉트

클라이언트가 외부 API에 직접 요청

장점

- 설정이 간단하다.
- 클라이언트가 API에 직접 요청하므로 NextJS 서버 부하가 없다.
- 리다이렉트로 인해 요청 처리가 매우 빠르다.

단점

- CORS 문제가 발생한다. (클라이언트가 직접 외부 API에 요청하므로)
- 데이터 가공이 불가능하다.
- API key 등 민감 정보가 노출될 가능성이 있어 보안 측면상 안좋다.

2. rewrites를 사용하는 방법

rewrites는 클라이언트에서 요청한 URL을 다른 URL로 매핑하여 요청을 처리합니다.

```
// next.config.ts
module.exports = {
  async rewrites() {
    return [
      {
        source: '/api/proxy',
        destination: 'https://api.coingecko.com/api/v3/coins/markets',
      },
    ];
  },
};
```

클라이언트에서 `/api/proxy` 로 요청을 보내면 NextJS가 이를 destination URL로 리다이렉트

CORS 문제가 발생하지 않음(NextJS 서버가 요청을 전달하기 때문)

장점

- 설정이 간단하다.
- NextJS 서버가 요청을 가공하지 않고 바로 전달하기 때문에 추가 부하가 거의 없어서 성능적 측면에서 유리하다.

- 데이터 가공이 필요없기 때문에 빠른 개발이 가능하다.

단점

- 데이터 가공이 불가능하다.
- 요청 파라미터를 동적으로 변경하는 로직을 구현할 수 없다.
- 클라이언트가 외부 API 요청과 URL을 직접 처리하므로 민감한 데이터(API key 등)가 노출될 가능성이 있어 보안 측면에서 안좋다.

3. NextJS API Routes를 사용하는 방법

API Routes는 NextJS 서버에서 API가 요청을 처리하고 데이터를 클라이언트에 전달하는 방식입니다.

```
// /pages/api/proxy.ts
import { NextApiRequest, NextApiResponse } from 'next';
import axios from 'axios';

export default async function handler(req: NextApiRequest, res: NextApiResponse) {
  try {
    const response = await axios.get('https://api.coingecko.com/api/v3/coins/markets', {
      params: { vs_currency: 'KRW', order: 'gecko_desc' },
    });
    res.status(200).json(response.data);
  } catch (error: any) {
    res.status(error.response?.status || 500).json({ message: error.message });
  }
}
```

`/api/proxy` 를 호출하면 NextJS 서버에서 외부 API를 요청하고 데이터를 클라이언트로 전달

요청과 응답을 중간에서 가공 가능

장점

- API key 등 민감한 정보를 클라이언트에 노출하지 않고 안전하게 처리 가능하다.
- 요청 파라미터를 동적으로 수정하거나 응답 데이터를 가공할 수 있다.
- NextJS 서버가 프록시 역할을 하므로 클라이언트에서 CORS 에러가 발생하지 않는다.
- 인증, 로깅, 요청 제한 등 추가 로직 구현이 가능하다.

단점

- NextJS 서버가 모든 요청을 처리하므로 서버 리소스가 많이 사용돼 서버 부하가 증가한다.
- API 라이투를 작성해야 하므로 초기 구현 비용이 다른 방법보다 상대적으로 높다.

제 프로젝트에서는 API를 가공하고 API key를 사용하므로 3번 방법인 NextJS API Routes를 사용하도록 하겠습니다.

▼ [참고] 프록시 서버의 역할

프록시 서버는 클라이언트와 원격 서버 사이에서 데이터를 중계하는 서버입니다.

- 원래 브라우저가 허용하지 않는 교차 출처 요청을 허용해 CORS 우회가 가능합니다.
- API key와 같은 민감한 정보를 클라이언트가 직접 노출하지 않도록 보호합니다.
- 클라이언트에서 API 요청을 단순화하거나, 로깅/캐싱 등을 추가로 처리할 수 있습니다.
- 원격 서버의 응답 데이터를 클라이언트에서 쉽게 사용할 수 있도록 데이터 변환이 가능합니다.

```
import { NextApiRequest, NextApiResponse } from 'next';
import axios from 'axios';

export const handler = async (req: NextApiRequest, res: NextApiResponse) => {
```

```
  try {
    const response = await axios.get(
      `${process.env.NEXT_PUBLIC_BASE_URL}`,
      {
        params: {
          vs_currency: 'KRW',
          order: 'gecko_desc',
          per_page: 10,
          page: 1,
          sparkline: false,
          price_change_percentage: '24h',
        },
      }
    )
    res.status(200).json(response.data);
  } catch (error: any) {
    res.status(error.response?.status || 500).json({
      message: error.message || '500 Error',
    });
  }
}
```

1. 클라이언트가 `http://localhost:3000/api/proxy`로 요청을 보냅니다.
2. NextJS API Routes(handler)가 이 요청을 처리합니다.
    a. 내부적으로 axios를 사용해 api.coingeko.com에 데이터를 요청
    b. 요청 시 필요한 쿼리 파라미터를 params에 포함
        i. 필수: vs_currency, 옵션은 아래 링크에서 선택해서 넣어주었습니다.
        ii. https://docs.coingecko.com/reference/coins-markets
            - `vs_currency`: 반환된 코인의 통화 기준 (필수)
            - `order`: 특정 순서로 데이터를 정렬
            - `per_page`: 반환할 데이터 개수
            - `page`: 페이지 번호
3. 원격 서버(api.coingecko.com)에서 데이터를 반환하면 NextJS API Routes는 클라이언트에 그대로 전달합니다.
4. 브라우저는 프록시 서버에서 전달받은 응답을 정상적으로 수신합니다.

통신 흐름

`클라이언트 브라우저` — 요청 GET /api/proxy →

`프록시 서버(NextJS API Routes)` — 요청 GET https://api.coingecko.com →

`외부 API 서버 (코인게코)` — 응답 JSON 데이터 →

`프록시 서버` — 응답 JSON 데이터 →

`클라이언트 브라우저`

참고

1. https://inpa.tistory.com/entry/WEB-📚-CORS-💯-정리-해결-방법-👏
2. https://docs.tosspayments.com/resources/glossary/cors
3. https://velog.io/@lucky-jun/proxy-nextjs
4. https://prod.velog.io/@dev_sony503/Next.js-API-Routes로-CORS에러-해결하기