Frontend, Backend 분리할 때 CORS 및 쿠키 문제

SPA(Single Page Application)은 기본적인 HTML, CSS, JavaScript 파일들만 제공받은 다음에 렌더링이 필요한 부분에 대해서만 직접 백엔드 서버에게 API 요청을 보내서 리렌더링을 하는 방식이다.

백엔드와 프론트엔드 서버가 서로 다른 도메인을 가지는 경우, 두가지 문제점이 발생한다.

1. CORS 정책은 다른 오리진의 자원에 접근하는 것을 막는다.
2. 다른 오리진에 대한 요청 시에는 쿠키를 전송 혹은 수신할 수 없다.

이를 해결하려면 백 엔드 서버에게 API 요청을 보낼 때 JavaScript 단에서 특정 설정 (XMLHttpRequest.withCredentials 옵션을 true로 설정하거나, fetch API라면 credentials 옵션을 include로 설정)을 해줘야 하며, 백 엔드 서버 쪽에도 응답의 헤더에서 Access-Control-Allow-Credentials 옵션을 true로 설정해줘야 한다.

Origin

스키마(프로토콜, 예를 들어 HTTP 또는 HTTPS), 호스트 이름, 포트(지정된 경우)의 조합

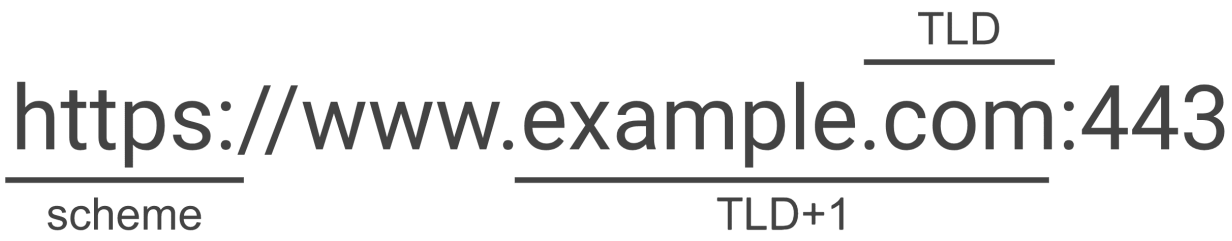
Origin A	Origin B	Explanation of whether Origin A and B are "same-origin" or "cross-origin"
https://www.example.com:443	https://**www.evil.com**:443	cross-origin: different domains
	https://**example.com**:443	cross-origin: different subdomains
	https://**login**.example.com:443	cross-origin: different subdomains
	http://www.example.com:443	cross-origin: different schemes
	https://www.example.com:**80**	cross-origin: different ports
	https://www.example.com:443	**same-origin: exact match**
	https://www.example.com	**same-origin: implicit port number (443) matches**

Site

.com, .org 등의 최상위 도메인 (TLD)은 루트 영역 데이터베이스에 나열됩니다. 위의 예에서 '사이트'는 스키마, TLD, 바로 앞의 도메인 부분 (TLD+1이라고 함)의 조합입니다. 예를 들어 URL이 https://www.example.com:443/foo 라면 '사이트'는 https://example.com입니다.

SOP(Same-Origin Policy)

- 같은 출처에서만 리소스를 공유할 수 있다



Origin A	Origin B	Explanation of whether Origin A and B are "same-site" or "cross-site"
https://www.example.com:443	https://**www.evil.com**:443	cross-site: different domains
	https://**login**.example.com:443	**same-site: different subdomains don't matter**
	http://www.example.com:443	cross-site: different schemes
	https://www.example.com:**80**	**same-site: different ports don't matter**
	https://www.example.com:443	**same-site: exact match**
	https://www.example.com	**same-site: ports don't matter**

Front에서 쿠키 값을 가져오지 못할 경우

- **서버응답확인: 개발자 도구로 Backend 요청 시 쿠키 값을 응답하는 지 확인**
- **CORS (Cross-Origin Resource Sharing) 설정 확인**
  - Backend 설정 값 확인
    - cors 설정 확인
    - 쿠키 속성 설정 확인
      - **HttpOnly**
      - **Secure 속성:**
      - **도메인 및 경로 설정**
  - Front 설정 확인
    - Axios 또는 다른 HTTP 클라이언트를 사용할 때, 요청에 withCredentials: true를 추가하여 쿠키를 전송하도록 설정

- **브라우저 쿠키 정책**

  SameSite 속성: 브라우저의 최신 버전에서는 SameSite 속성이 추가되어 있습니다. SameSite 속성이 설정되어 있다면, 쿠키가 어떤 상황에서 전송되는지 확인합니다.

```
cookie.setSameSite(SameSite.None);
```

- **브라우저 캐시 비활성화** 되었는지 확인

기타 설명

Request Header에서의 CORS 확인:

- **Origin:** 이는 현재 요청이 어떤 오리진에서 시작되었는지를 나타냅니다.
- **Sec-Fetch-Mode:** 이는 브라우저가 CORS 요청을 보내고 있다는 것을 나타냅니다.

```
Host: localhost:5173
Origin: http://localhost:5173
Referer: http://localhost:5173/login
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
```

프론트엔드에서 서버로의 요청을 보낼 때, 서버는 설정된 쿠키를 응답 헤더에 포함시켜야함.

withCredentials: true,

```
axios.defaults.withCredentials = true;

const submit = () => {
  axios.post("/api/account/signin", state.form,{
  })
      .then((data) => {
        console.log(data.headers)
        console.log(data.headers['set-cookie'])

        // 서버로부터 설정된 쿠키에 접근
        const setCookieHeader = data.headers['set-cookie'];
        if (setCookieHeader) {
```

```
        // 여기서 setCookieHeader 변수에 서버로부터 설정된 쿠키 값이 포함됩니다.
        console.log('Set-Cookie Header:', setCookieHeader);

        // 쿠키 값을 원하는 방식으로 처리 가능
        // 예를 들면, 쿠키 값을 추출하거나 저장할 수 있습니다.
      }


      window.alert("로그인 성공")
      router.push({path: "/"})
    })
    .catch((error) => {
      console.log(error)
      window.alert("로그인 실패")
    });
```

Response Header에서의 CORS 확인:

- **Access-Control-Allow-Origin:** *으로 설정되어 있습니다. 이는 모든 오리진에서의 요청을 허용하고 있다는 것을 의미합니다.

Backend - CORS 허용 설정:

CORS :: Spring Security

```
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    http
            .csrf(AbstractHttpConfigurer::disable)
            .cors(AbstractHttpConfigurer::disable)
```

```
CorsConfigurationSource corsConfigurationSource() {
    // CorsConfiguration 객체 생성
    CorsConfiguration configuration = new CorsConfiguration();

    // 자격 증명(크로스 도메인 요청에서 쿠키 및 HTTP 인증과 같은 자격 증명이 포함되도록 허용)
    configuration.setAllowCredentials(true);

    // 허용할 오리진(도메인) 목록 설정
```

```
    // 실제 운영 환경에서는 보안상의 이유로 가능한한 특정 도메인만을 명시하는 것이 좋습니다.
    configuration.setAllowedOrigins(Arrays.asList("http://localhost:8080",
"http://localhost:5173"));

    // 허용할 HTTP 메서드 목록 설정 (여기서는 GET과 POST만 허용)
    configuration.setAllowedMethods(Arrays.asList("GET", "POST"));

    // UrlBasedCorsConfigurationSource 객체 생성
    UrlBasedCorsConfigurationSource source = new
UrlBasedCorsConfigurationSource();

    // 모든 경로에 대해 위에서 설정한 CorsConfiguration을 적용
    source.registerCorsConfiguration("/**", configuration);

    // 설정이 적용된 CorsConfigurationSource 반환
    return source;
}
```

- 쿠키 생성할 때 path가 있어야한다.

```
Cookie cookie = new Cookie( name: "jwt", token);
cookie.setMaxAge(7 * 24 * 60 * 60); // expires in 7 days
cookie.setSecure(false);
cookie.setHttpOnly(false);
cookie.setPath("/"); // Global
response.addCookie(cookie);
```

#dev/개발일지