2021.07.30 금요일 \_ 업무내용(정준호)

**업무전달**

API 호출시, 데이터 받아올 때, 인증 그리고 Csrf, Csrf Token 작업시 문제점, 구현

**1. JWT 토큰이란**  
-JWT, JSON web Token의 약자로 토큰 기반 인증에 사용되는 토큰을 의미한다. JWT 토큰은 문자열로 구성되어 있기 때문에, HTTP의 해더든, URL이든, Body든 어디에도 위치할 수 있따는 장점이 있다. 이러한 토큰 기반 인증 방식은 기존의 세션 방식의 인증에 비해 여러 장점을 갖고 있다. 기존 서버 – 클라이언트 구조에서는 세션을 서버가 저장하게 되고 이는 유저가 많아질수록 서버에 메모리에 저장되는 정보가 많아지므로 과부하를 가져올 수 있다.

그리고 세션 정보를 크라이언트가 쿠키에 저장하게 되는데, 이 쿠키는 stateful하기 때문에 이 정보를 서버와 클라이언트가 모두 갖고 있어야 한다는 단점이 있다.

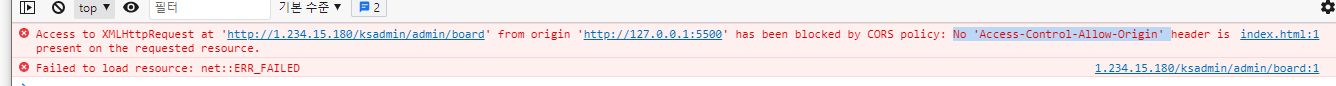
-JWT 토큰의 문제는 Token자체의 문자열 길이가 길기 때문에 payload 데이터가 커진다. Session ID 보다 token의 문자열 길이가 더 길기 때문에 리퀘스트에 이러한 긴 토큰 정보가 포함되는 것 자체가 좋지는 않다.

- 한번 발급된 jwt토큰은 계속해서 사용할 수 있다. 해커가 한번 토큰을 해킹하면 계속해서 사용할 수 있기 때문에 Access Token 의 유효 기간을 짧게 하고 Refresh Token 을 발급해주면 비교적 피해를 줄일 수 있다.

**2. CSRF(Cross Sitre Request Forgery)**

한마디로 사용자의 의도와 관계 없이 행해지는 공격 기법을 의미한다. 예를 들어 사용자가 그저 버튼을 눌렀을 뿐인데 해커가 심어 놓은 스크립트에 의해 내 계정의 인증과정을 거쳐서 나의 의도와 관계없이 블로그에 광고성 글이 올라갈 수도 있는 것이다.

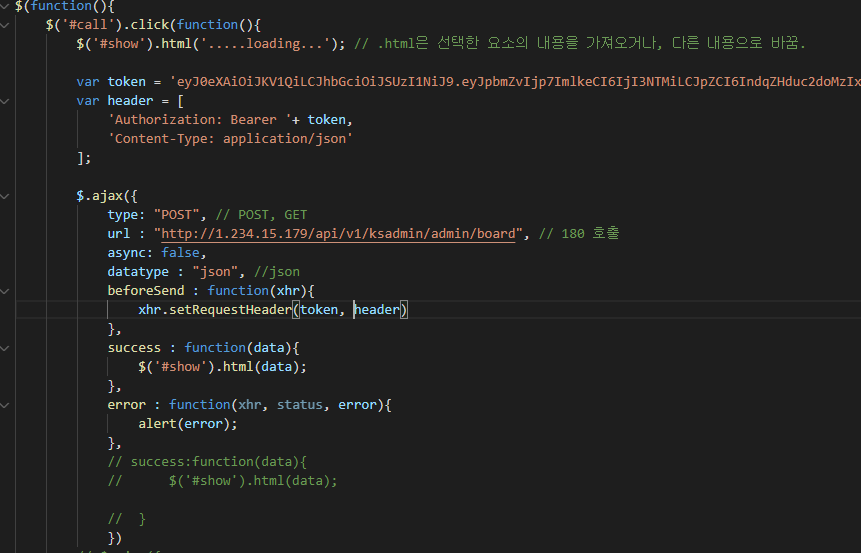
**3. CSRF Token**CSRF token은 임의의 난수를 생성하고 세션에 저장한다. 그리고 사용자의 매 요청마다 해당 난수 값을 포함시켜 전송시킨다. 이후 백엔드에서는 요청을 받을 때 마다 세션에 저장된 토큰값과 요청 파라미터에 전달된 토큰 값이 같은지 검사한다.

****

**4. Ajax beforeSend 는 ajax를 요청하기 직전의 콜백함수이다.**특정 기능을 구현할 때 ajax를 이용하여 API 에 직접 접근하여 데이터를 가져오는 상황에서 보안관련 이슈가 생길 수 있다. 해당 API에 접근하려면 header의 token을 전달하여 호출해야 하는 상황일 때, Ajax의 경우 beforeSend 부분을 추가하여 쉽게 처리가 가능하다.

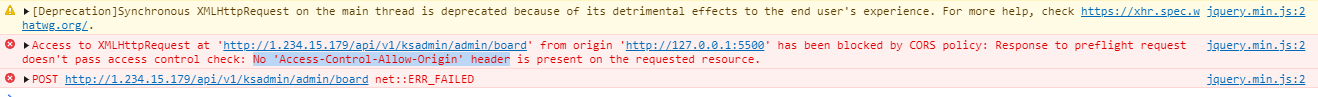


178(백엔드) API 서버 호출을 위해서 직접 작성한 나의 코드



**5. javaScript 엔진 표준 스페엑 있는 동일 출처 정책(Same-Origin Policy)이라는 보안 규칙**

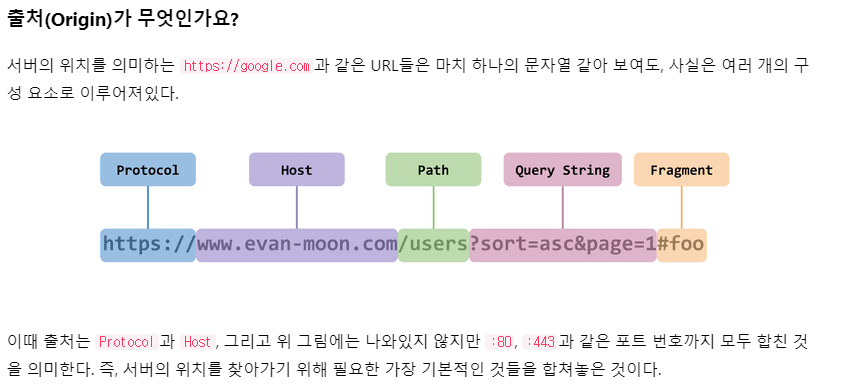
No 'Access-Control-Allow-Origin' header



javaScript 엔진 표준 스페엑 있는 동일 출처 정책(Same-Origin Policy)이라는 보안 규칙으로 인해 발생한다. (JavaScript로 다른 웹 페이지에 접근할 때는 같은 출처-프로토콜, 호스트명, 포트—의 페이지에만 접근 가능하다. 때문에 같은 서버에 있는 주소로만 ajax 요청을 할 수 있다.

여러 도메인에 걸쳐서 구성되는 대규모 웹 어플리케이션이 늘어나고 Rest API 등을 이용한 외부 공개 API의 호출이 많아지는 상황에서 이 보안 정책은 Cors(Cross-Origin Resource Sharing)이라는 정책으로 바뀌었따. 외부 요청을 허용할 경우 ajax요청이 가능해지는 방식이다.

즉 CORS 관련 이슈는 모두 CORS 정책을 위반했기 때문이다. [[1]](#footnote-1)



우리가 다른 출처로 리소스를 요청한다면 SOP(Same-origin policy) 정책을 위반한 것이 되고, 거기다가 SOP의 예외 조항인 CORS 정책까지 지키지 않는다면 아예 다른 출처의 리소스를 사용할 수 없게 되는 것이다.

그렇다면 어떤 경우에 출처가 같다고 판단하고, 어떤 경우에 출처가 다르다고 판단하는가?

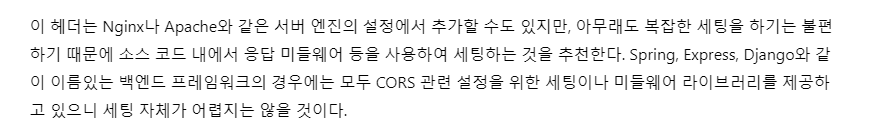
2가지 URL의 구성요소 중에서 Scheme(protocol), Host, port 이 3가지만 동일하면 된다.

여기서 중요한 사실 한 가지는 이렇게 출처를 비교하는 로직이 서버에 구현된 스펙이 아니라, 브라우저에 구현되어 있는 스펙이라는 것이다. 만약 우리가 CORS 정책을 위반하는 리소스 요청을 하더라도 해당 서버가 같은 출처에서 보낸 요청만 받겠다는 로직을 가지고 있는 경우가 아니라면 서버는 정상적으로 응답을 하고, 이후 브라우저가 이 응답을 분석해서 CORS 정책 위반이라고 판단되면 그 응답을 사용하지 않고 그냥 버리는 순서인 것이다,

즉 CORS는 브라우저의 구현 스펙에 포함되는 정책이기 때문에, 브라우저를 통하지 않고 서버 간 통신을 할 때는 이 정책이 적용되지 않는다. 또한 CORS 정책을 위반하는 리소스 요청 때문에 에러가 발생했다고 해도 서버 쪽 로그에서는 정상적으로 응답을 했다는 로그만 남기 때문에, CORS 가 돌아가는 방식을 파악하기 어렵다.

->해결방법

정석대로 서버에서 Access-Control-Allow-Origin 헤더에 알맞은 값을 세팅해주는 것이다. 이때 와일드 카드인 \*을 사용하여 이 헤더를 세팅하게 되면 모든 출처에서 오는 요청을 받아먹겠다는 의미이므로 당장은 편할 수 있지만, 보안적 이슈가 발생할 수 있다.



1. [CORS는 왜 이렇게 우리를 힘들게 하는걸까? | Evans Library (evan-moon.github.io)](https://evan-moon.github.io/2020/05/21/about-cors/) [↑](#footnote-ref-1)