


SeSAC 4기, 

웹 풀스택 과정 JWT 수업

WITH 팀 리쳐드



JWT

JWT란?

- Json Web Token
- 정보를 비밀리에 전달하거나 인증할 때 주로 사용하는 토큰으로 **JSON 객체를** 이용하는 토큰
- 클라이언트와 서버 사이의 통신에서 권한을 위해 사용하는 토큰



JWT란?

- JWT는 사용자 정보를 JSON 객체에 담아 이를 암호화하고 해싱 작업을 거쳐 문자열 토큰을 생성한다.
- JWT는 서버에 저장되지 않기 때문에 서버 부하를 일으키지 않고, 해싱을 통해 **데이터의 무결성**을 보장하는 인증 방식이다.
- JWT는 토큰 자체를 정보로 사용하는 Self-Contained 방식으로 정보를 안전하게 전달한다.
- Session의 단점을 보완하기 위해 탄생한 개념

JWT 구성요소



JWT는 헤더(header), 페이로드(payload), 서명(signature) 세 파트로 나뉘져 있다.

JWT 구성요소 – Header

- JWT에서 사용할 타입과 해시 알고리즘의 종류가 포함되어 있다.
- 대표적으로 사용하는 알고리즘
 - HMAC, SHA256, RSA, HS256 또는 RS256

```
{  
  "typ": "JWT",  
  "alg": "HS256"  
}  
// typ : 토큰 유형  
// alg : 서명 암호화 알고리즘 ( ex) HMAC SHA256, RSA )
```

JWT 구성요소 - Payload

- 서버와 클라이언트가 주고받는 시스템에서 실제로 사용될 정보에 대한 내용
용을 담고 있는 부분
- 서버에서 첨부한 사용자 권한 정보와 데이터가 포함되어 있다.
- 토큰에서 사용할 정보의 조각들인 Claim이 포함되어 있는 곳
 - Claim이란? key-value 형식으로 이루어진 한 쌍의 정보

JWT 구성요소 – Payload

- Payload의 데이터 타입은 대표적으로 3개가 존재
 - 1) Registered Claims : 미리 정의된 클레임
 - 2) Public Claims : 사용자가 정의할 수 있는 클레임으로 공개용 정보 전달을 위해 사용되는 데이터
 - 3) Private Claims : 해당하는 당사자들 간의 정보를 공유하기 위해 만들어진 사용자 지정 클레임. 외부에 공개되도 상관없지만 해당 유저를 특정할 수 있는 정보가 들어온다.

```
{  
  "sub": "1234567890",  
  "name": "Hong",  
  "iat": "15876513"  
}
```


JWT 구성요소 – Payload

- Registered Claims에 정의되어 있는 클레임
 - iss (issuer; 발행자)
 - exp (expiration time; 만료 시간)
 - sub (subject; 제목)
 - iat (issued At; 발행 시간)
 - jti (JWI ID)

JWT 구성요소 - Signature

- 서명 부분으로 Header, Payload를 Base64 URL-safe Encode를 한 이후 Header에 명시된 Hash 함수를 적용하고, **개인키(Private Key)**로 서명한 전자서명이 담겨 있는 부분
- 전자서명이란?
 - **비대칭 암호화 알고리즘**을 사용해 만들어진 것
 - 암호화(전자서명)에는 개인키를, 복호화(검증)에는 공개키가 사용된다.

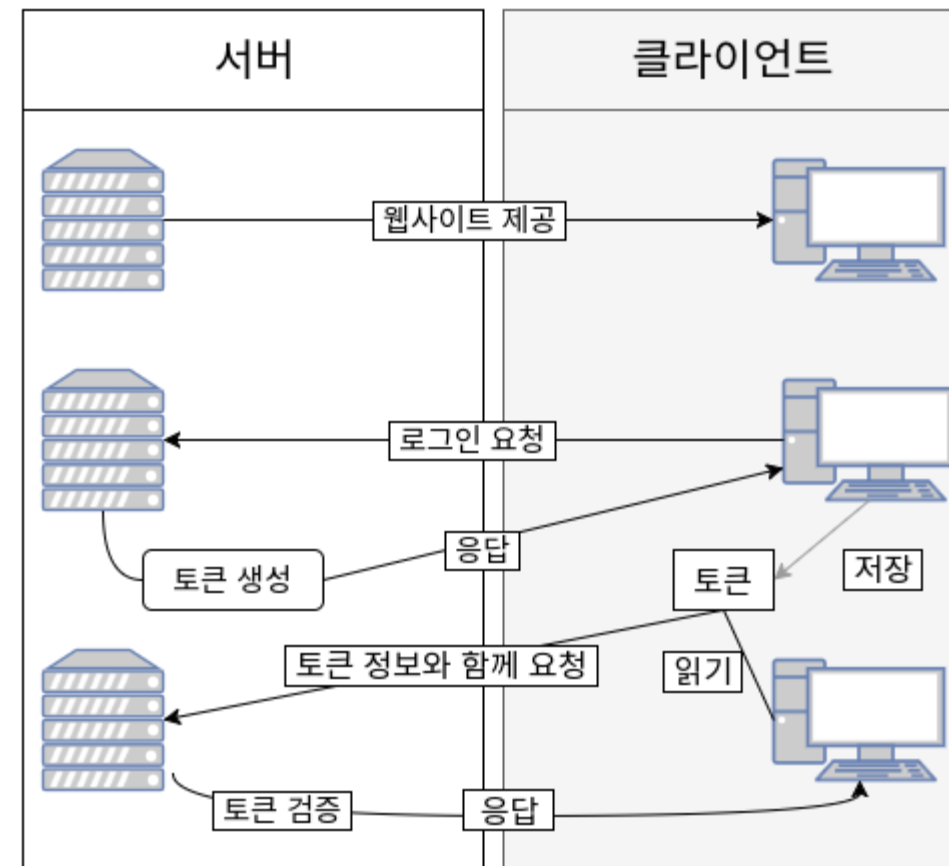
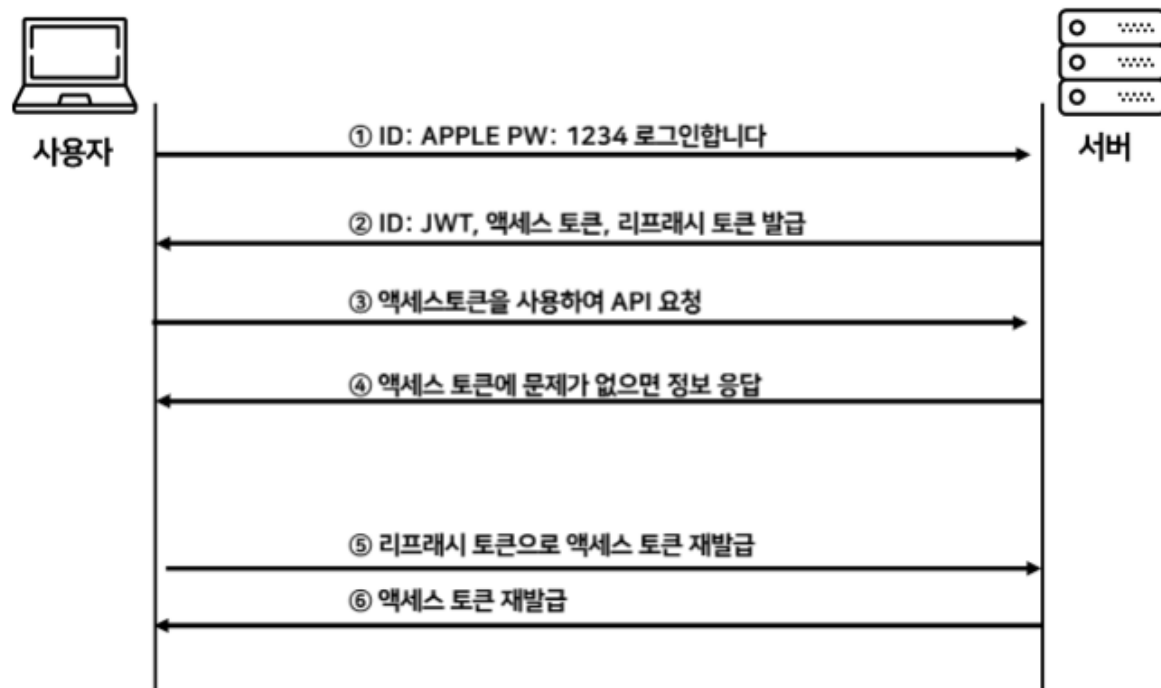
JWT 구성요소 - Signature

Signature = Base64Url(Header) + . + Base64Url(PayLoad) + server's key

HS256

- 구조는 **Header + Payload**와 서버가 갖고 있는 **유일한 key 값**을 합친 것을 Header에서 정의한 알고리즘으로 암호화를 진행한다.
- Signature은 서버 측에서 관리하는 비밀 키를 이용하기 때문에 비밀키가 유출되지 않는 한 복호화가 불가능하다.
- Signature은 토큰의 위변조 여부를 확인하는데 사용된다.

JWT 흐름



JWT 흐름

1. 사용자가 ID와 PW를 입력해 서버에 로그인 인증을 요청한다.
2. 서버에서 클라이언트로부터 요청을 받으면, Header, Payload, Signature을 정의해 JWT을 만들고 쿠키에 담아 클라이언트에게 발급한다.
3. 클라이언트는 전달받은 JWT를 로컬 스토리지나 쿠키 등의 공간에 저장한다.
4. 클라이언트가 다시 한 번 서버에 요청할 때 Access Token을 Authorization Header에 담아서 보낸다.
5. 서버는 JWT를 검증하고 인증을 통과하면 Payload에 있는 정보를 이용해 요청을 처리한 후 클라이언트에게 돌려준다.

JWT와 Cookie & Session

	장점	단점
Cookie & Session	<ul style="list-style-type: none">• Cookie만 사용하는 방식보다 보안 향상• 서버 쪽에서 Session 통제가 가능하다.• 네트워크의 부하가 낮음	<ul style="list-style-type: none">• 세션 저장소 사용으로 인한 서버 부하가 발생할 수 있다.
JWT	<ul style="list-style-type: none">• 인증을 위한 별도의 저장소가 필요 없다.• 별도의 I/O 작업이 필요 없어 빠른 인증 처리가 가능하다.• 확장성이 우수하다.	<ul style="list-style-type: none">• 토큰의 길이가 늘어날수록 네트워크 부하가 발생할 수 있다.• 처음 설정한 것 이외에 특정 토큰을 강제로 만료시키는 것은 어렵다.

JWT의 장점

- Header와 Payload를 가지고 Signature을 생성하므로 데이터 위변조를 막을 수 있다.
- 인증 정보에 대한 별도의 저장소가 필요 없다.
- 확장성이 우수하다.
- 토큰 기반으로 다른 로그인 시스템에 접근 및 권한 공유가 가능하다. (쿠키와의 차이점)
- 모바일 어플리케이션 환경에서도 잘 동작한다. (세션과의 차이점)

JWT의 단점

- 쿠키, 세션과 다르게 토큰 길이가 길어 인증 요청이 많아질수록 네트워크 부하가 심해진다.
- Payload 자체는 암호화되지 않기 때문에 유저의 중요한 정보는 담을 수 없다.
- 토큰을 탈취당하면 대처하기가 어렵다.

Access & Refresh Token

- Access Token
 - 인증을 위한 JWT이며, 보안을 위해 유효기간이 매우 짧다.
- Refresh Token
 - 유효기간이 짧은 Access Token을 보완하기 위한 JWT
 - Access Token에 비해 유효기간이 길다.
 - Refresh Token을 이용해 Access Token이 만료되었을 때 새로 발급한다.

JWT 사용하기

JWT 사용하기

```
npm i express jsonwebtoken
```

```
const jwt = require('jsonwebtoken');  
const secret = 'JWT-SECRET-KEY';
```

JWT 사용하기

jwt.sign(payload, 비밀키, [options, callback]);

```
const token = jwt.sign({
  type: 'JWT',
  name: req.body.name,
  email: req.body.email
}, secret, {
  expiresIn: '15m', // 만료시간 15분
  issuer: '토큰발급자',
});
```

```
req.decoded = jwt.verify(req.headers.authorization, secret);
```