

# JWT(JSON Web Token)

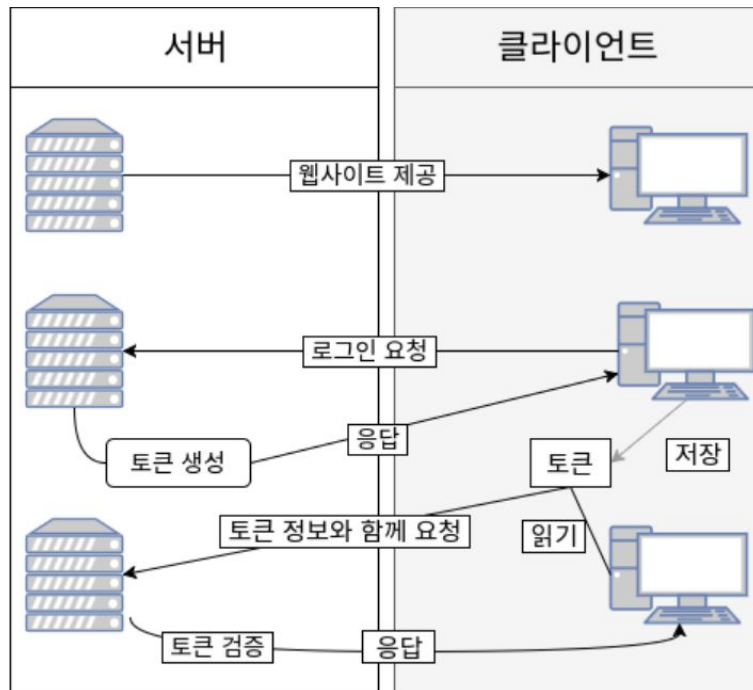
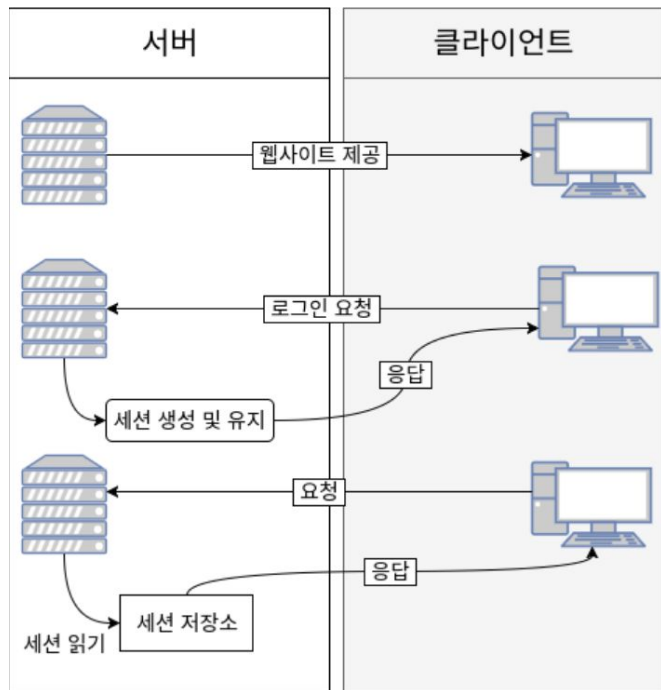
디지털무역기술실 유동관

# 목차

1. JWT 정의
2. JWT 특징
3. 어떤 상황에서 사용하는 것이  
좋을까?
4. 생김새
5. 실습

# 1. JWT 정의

- 토큰 기반 인증 시스템의 구현체이다.



## 1-1 토큰의 장점

1. 무상태(Stateless) : 토큰이 클라이언트쪽에 저장되어 서버에 대한 부하 감소
2. 확장성(Scalability) : 어떤 서버로 요청이 들어가던 동일한 토큰 검증과정을 거치기 때문에 높은 서버 확장성 가짐
3. 보안성(Security) : 서버에 요청시, 더 이상 쿠키를 전달하지 않음으로 XSS를 통한 권한 복제 문제 해결 가능
4. 확장성(Extensibility) : 토큰에 선택적인 권한만 부여하여 발급 가능  
ex) 페이스북 계정으로 로그인시, 프로필 정보를 가져오는 권한은 있지만,  
포스트를 작성할 권한은 없다.

## 2. JWT 특징

1. JSON 객체를 사용 : 가볍고 안전한 정보 전달 방식
2. 주류 프로그래밍 언어(C, JAVA, PYTHON, C++, PHP, JS,,,) 등 대부분 지원
3. 자가 수용적(Self-acceptable)이다 : 필요한 모든 정보를 자체적으로 지님
  - a. 토큰에 대한 기본 정보,
  - b. 전달할 정보(유저 정보)
  - c. 검증되었다는 것을 증명해주는 서명
4. 쉽게 전달 방식
  - a. HTTP헤더에 넣어서 전달
  - b. URL의 파라미터에 넣어 전달

### 3. 어떤 상황에서 사용하는 것이 좋을까?

---

#### 1. 회원 인증

- a. 유저 로그인 - 서버는 유저 정보에 기반한 토큰을 발급하여 유저에게 전달가능
- b. 유저는 서버에 요청할 때마다 JWT를 포함하여 전달
- c. 서버는 유저에게 요청받을 때마다, 해당 토큰이 유효한지, 혹은 인증된 토큰인지 검증
- d. 유저가 요청한 작업에 권한이 있는지 확인

#### 2. 정보 교류

- a. 두 개체 사이에서 안정성 있게 정보를 교환하기에 용이

헤더(header)                  내용(payload)                  서명(signature)

1

2

3

1

2

3

```

    HMACSHA256(
      BASE64URL(header)
      .
      BASE64URL(payload) ,
      secret)

```

## 4-1. 헤더

- typ:토큰의 타입 지정 JWT
- alg:해싱 알고리즘 지정 sha256,RSA

```
# Node.js에서의 인코딩
const header = {
  "typ": "JWT",
  "alg": "HS256"
};

// encode to base64
const encodedPayload = new Buffer(JSON.stringify(payload))
  .toString('base64')
  .replace('=', '');

console.log('payload: ', encodedPayload);

/* Result:
header: eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9
*/
```



## 4-2. 정보(payload)

- 클레임(claim) : 정보의 한 ‘조각’
  - 등록된 클레임 : iss, sub, aud, exp, nbf, iat, jti
  - 공개 클레임 : URL 형식으로 된 클레임
  - 비공개 클레임 : 충돌에 유의하여 사용

```
const payload = {
  "iss": "velopert.com", // 등록된 클레임
  "exp": "1485270000000", // 등록된 클레임
  "https://velopert.com/jwt_claims/is_admin": true, // 공개 클레임
  "userId": "11028373727102", // 비공개 클레임
  "username": "velopert" // 비공개 클레임
};

// encode to base64
const encodedPayload = new Buffer(JSON.stringify(payload))
  .toString('base64')
  .replace('=', '');

console.log('payload: ', encodedPayload);

/* result
payload:
eyJpc3MiOiJ2ZWxvcGVydC5jb20iLCJleHAiOiIxNDg1MjcwMDAwMDAwIiwiaHR0cHM6Ly92ZWxvcGVydC5jb20vYW50b2N5YWltcy9pc19hZG1pb20iLCJ1ZSwidXN1cklkIjoimTEwMjgzNzZMjcwMDIiLCJ1c2VybmFtZSI6InZlbG9wZXJ0In0
*/
```

## 4-3. 서명(Signature)

- 헤더의 인코딩값, payload의 인코딩값을  
합쳐 사용(중간에 `.`을 넣어 결합)
- 비밀키로 재해쉬하여 생성

// 수도코드

```
HMACSHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  secret)
```

```
const crypto = require('crypto');  
const signature = crypto.createHmac('sha256', 'secret')  
  .update(encodedHeader + '.' + encodedPayload)  
  .digest('base64')  
  .replace('=', '');
```

```
console.log('signature: ',signature);
```

/\* result

```
Signature: WE5fMuFM0NDSVGJ8cAo1XGkyB5RmYwCto1pQwDIqo2w
```

\*/

## 4-4. 결과

### Encoded

PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36P0k6yJV_adQssw5c
```

### Decoded

EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

## 5. 실습

- jjwt를 이용하여 JWT를 만드는 코드
- JWT의 헤더, 클레임, 암호 등의 필요한 정보를 넣고 직렬화(compact())
- 단순히 Session처럼 정보를 넣어놓고 빼쓰기 위해서는 클레임에 데이터 삽입

```
@Service("jwtService")
public class JwtServiceImpl implements JwtService{

    private static final String SALT = "luvookSecret";

    @Override
    public <T> String create(String key, T data, String subject){
        String jwt = Jwts.builder()
            .setHeaderParam("typ", "JWT")
            .setHeaderParam("regDate", System.currentTimeMillis())
            .setSubject(subject)
            .claim(key, data)
            .signWith(SignatureAlgorithm.HS256, this.generateKey())
            .compact();

        return jwt;
    }

    private byte[] generateKey(){
        byte[] key = null;
        try {
            key = SALT.getBytes("UTF-8");
        } catch (UnsupportedEncodingException e) {
            if(log.isInfoEnabled()){
                e.printStackTrace();
            }else{
                log.error("Making JWT Key Error ::: {}", e.getMessage());
            }
        }

        return key;
    }
}
```

## 5. 실습

- 꺼낼 때는 parse 방식으로 꺼냄
  - HTTP Header -> JWT -> Claim -> Key -> Value.

```
@Service("jwtService")
public class JwtServiceImpl implements JwtService{
    @Override
    public Map<String, Object> get(String key) {
        HttpServletRequest request = ((ServletRequestAttributes)
        RequestContextHolder.currentRequestAttributes()).getRequest();
        String jwt = request.getHeader("Authorization");
        Jws<Claims> claims = null;
        try {
            claims = Jwts.parser()
                .setSigningKey(SALT.getBytes("UTF-8"))
                .parseClaimsJws(jwt);
        } catch (Exception e) {
            throw new UnauthorizedException();
        }
        @SuppressWarnings("unchecked")
        Map<String, Object> value = (LinkedHashMap<String,
        Object>)claims.getBody().get(key);
        return value;
    }
}
```

- 전체 소스 : <https://github.com/viviennes7/luvook>

# Q&A

감사합니다.