JWT(JSON Web Token)

1. 기존 cookie나 session을 통한 인증 처리의 문제점

- 쿠키는
  - 쿠키는 노출이 되기때문에 id, pw에 대한 민감 정보까지 다 노출이 되어 보안이 좋지 않다.
  - 브라우져 별 cookie에 대한 지원 형태 문제로 브라우져 간에 공유할 수 없다.
  - 쿠키 사이즈가 커질수록 네트워크의 부하가 가중된다.
- 세션
  - 세션 저장소의 문제가 발생하면 인증 체계가 무너져 이전에 다른 인증된 유저 또한 인증을 사용할 수 없다..
  - stateful 하기 때문에 http 의 장점을 발휘하지 못하고 scale out 에 걸림돌이 생긴다.
  - 세션 저장소가 필수적으로 존재하기 때문에 이를 사용하기 위한 비용이 든다.
  - 세션 ID 도 쿠키로 저장되므로 탈취 당했을 경우 해당 클라이언트 인척 위장하는 보안의 약점이 있을 수 있다.
  - 사용자가 많아질수록 메모리를 많이 차지하게 된다.
  - 매번 요청 시 세션 저장소를 조회해야 하는 단점이 있다.

2. JWT란

- JWT(Json Web Token)은 인증에 필요한 정보를 암호화 시킨 JSON 토큰으로 인터넷 표준 인증 방식입니다
- 공개/개인 키를 쌍으로 사용하여 생성한 토큰 내부에는 위변조 방지를 위한 개인키를 통한 전자서명(signature)이 포함되어 있어 보안성까지 갖추고 있다.
- JWT는 별도의 세션 저장소를 강제하지 않기 때문에 stateless 하여 확장성이 뛰어나다.

3. JWT 구조

JWT는 각각의 구성요소가 점(.)으로 구분이 되어있으며 구성 요소는 다음과 같습니다.

Header, Payload, Signature

3.1 Header

header에는 보통 토큰의 타입이나, 서명 생성에 사용할 해싱 알고리즘을 지정한다.

```
{
  "typ": "JWT",
  "alg": "HS512"
}
```

위 헤더는 현재 토큰의 타입이 JWT이고, 개인키로 HS512 알고리즘으로 암호화를 적용.

3.2 Payload(정보)

- **Payload** 부분에는 토큰에 담을 정보를 설정한다. 여기에 담는 정보의 한 '조각' 을 클레임(**claim**)이라고 부르고, 이는 name / value 의 한 쌍으로 구성되어 있다.
- 토큰에는 여러 개의 클레임을 넣을 수 있습니다.
- JWT 는 공개 키이므로 decoding 하면 누구나 해당 정보를 알 수 있으므로 Payload 에 민감한 정보를 담지 않는다.
- 클레임 의 종류는 다음과 같이 크게 세 분류로 나뉘어져 있다:
  - 등록된 (**registered**) 클레임,
  - 공개 (**public**) 클레임,
  - 비공개 (**private**) 클레임

- 등록된 (**registered**) 클레임

  등록된 클레임들은 서비스에서 필요한 정보들이 아닌, 토큰에 대한 정보들을 담기 위하여 이름이 이미 정해진 클레임이다. 등록된 클레임의 사용은 모두 선택적 (optional)이며, 이에 포함된 클레임 이름들은 다음과 같습니다

클레임	풀 네임	설명
iss	issuer	토큰 발급자
sub	subject	토큰 제목
aud	audience	토큰 대상자
exp	expiration	토큰의 만료시간 시간은 NumericDate 형식으로 되어야한다 항상 현재시간 이후로 설정되어 있어야 한다.
nbf	Not Before	토큰의 활성 날짜 nbf 이전 날짜의 토큰은 활성화되지 않음을 보장
iat	issued At	토큰 발급 시간
jti	JWT id	JWT 토큰 식별자

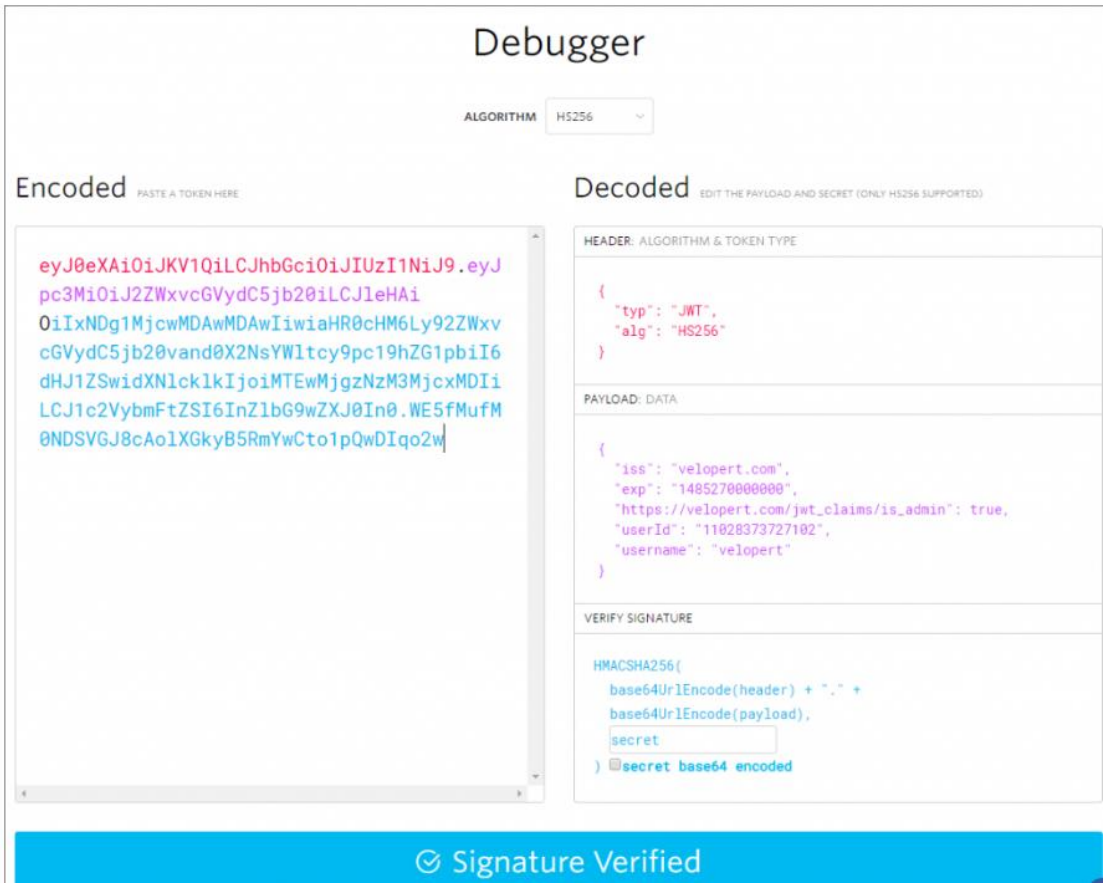
- 공개 (**public**) 클레임,
  공개 클레임들은 충돌이 방지된 (collision-resistant) 이름을 가지고 있어야 한다. 충돌을 방지하기 위해서는, 클레임 이름을 URI 형식으로 짓는다

```
{
    "https://velopert.com/jwt_claims/is_admin": true
}
```

- 비공개 (**private**) 클레임

보통 클라이언트와 서버 사이에서 협의 하에 사용되는 클레임이다. 공개 클레임과는 달리 이름이 중복되어 충돌이 될 수 있으니 사용할 때에 유의해야 한다.

https://jwt.io/



3.3 Signature

Signature는 헤더의 인코딩 값과, 정보의 인코딩 값을 합친 후 주어진 비밀키(개인 키)로 해쉬를 하여 해당 서버 생성한다. 따라서 Signature는 서버에 있는 개인 키로만 해당 서버에서 발급한 토큰인지 확인 할 수 있으므로 따로 토큰을 위조하거나 변경할 수 없다.

4. 장점 단점

- 장점

  - 이미 토큰 자체가 인증된 정보이기 때문에 세션 저장소와 같은 별도의 인증 저장소가 "필수적"으로 필요하지 않다.
  - 세션과는 다르게 클라이언트의 상태를 서버가 저장해 두지 않아도 된다.
  - signature 를 개인키로 암호화했기 때문에 데이터에 대한 보완성 뛰어나다.
  - 다른 서비스에 이용할 수 있는 공통적인 스펙으로써 사용할 수 있다.

- 단점
  - JWT 는 기본적으로 Claim 에 대한 정보를 암호화하지 않는다. 단순히 BASE64 로 인코딩만 하기 때문에, 중간에 패킷을 가로채거나, 기타 방법으로 토큰을 취득했으면 토큰 내부 정보를 통해서 사용자 정보가 누출 될 수 있는 가능성이 있다 ➔JWT 가 이미 인증을 의미 하므로 사용자의 중요 정보를 Claim 에 넣지 않거나 토큰 자체를 암호화하는 방법이 있다. JSON 을 암호화하기 위한 스펙으로는 JWE(JSON Web Encryption)있다.
  - 한번 발행한 토큰은 만료 시간 전에 폐기 할 수 없으므로 토큰이 탈취 당하면 만료될 때까지 대처가 불가능하다.

    ➔만료 시간을 짧게 한다.

- 짧은 유효 시간을 보안 방법
  - Sliding Session

    글쓰기나 결제 등과 같은 특정 서비스를 계속 사용하고 있는 특정 유저에 대한 만료 시간을 연장시켜준다.

  - Refresh Token

    긴 유효 시간을 가진 Refresh Token 은 Access Token 을 Refresh 해 주는 것을 보장하는 토큰이다. Access Token 이 만료되면 Refresh Token 으로 서버에게 새로운 Access Token 을 발급 하도록 요청한다.

5. Spring에서 JWT 구현

5.1 라이브러리 추가

```
<!-- JWT -->
<!-- https://mvnrepository.com/artifact/io.jsonwebtoken/jjwt-api -->
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-api</artifactId>
    <version>0.12.6</version>
</dependency>
<!-- https://mvnrepository.com/artifact/io.jsonwebtoken/jjwt-impl -->
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt-impl</artifactId>
    <version>0.12.6</version>
    <scope>runtime</scope>
</dependency>
```

5.2 JWT 발급 받기

5.2.1 UserService를 통해 인증을 확인한다.

5.2.2 인증된 경우 JWT 토큰을 발급한다.

함수명	설명
signWith(key,암호)	개인키와 HS512 암호화 알고리즘으로 header와 payload로 Signature를 생성
setHeaderParam	header에 typ로 JWT 지정
setSubject("제목")	토큰 제목
setIssuer("발급자")	JWT 발급자 지정
setExpiration(날짜)	현재 시간으로부터 지정된 accessTime만큼의 만료시간 지정
setIssueAt(날짜)	JWT를 발급한 시간 지정
claim(key, value)	공개 또는 비공개 클래임 설정
compact( )	키를 직렬화해서 생성

```
////TODO 4. Token 발급하는 함수 작성하기
//      JWT 구성 : Header + Payload(Claim) + Signature
private String generateToken(Map<String, Object> claims, String subject, long expireTime) {

    //Header 설정.
    Map<String, String> headers = new HashMap<>();
    headers.put("typ", "JWT");
    log.debug("generateToken- expireTime:{}", expireTime);
    return Jwts.builder().header().add(headers).and().claims(claims).subject(subject)
            .issuedAt(new Date(System.currentTimeMillis()))
            .expiration(new Date(System.currentTimeMillis() + expireTime))
            .signWith(getSigningKey()).compact();
}

////TODO 5. Signature 설정에 들어갈 key 생성.
private SecretKey getSigningKey() {
    byte[] keyBytes = SALT.getBytes(StandardCharsets.UTF_8);
    return Keys.hmacShaKeyFor(keyBytes);
}

////TODO 6. id 정보를 이용한 AccessToken을 생성하는 함수 작성하기
public String createAccessToken(String userId) {
    Map<String, Object> claims = new HashMap<>();
    claims.put("userId", userId);
    claims.put("tokenType", "ACCESS");
    return generateToken(claims, "access-token", accessTokenExpireTime);
}
```

5.2.3 토큰 정보를 응답한다.

5.3 전달 받은 토큰이 유효한지 검사

```
////TODO 8. 전달 받은 토큰이 제대로 생성된 것인지 확인 하고 문제가 있다면 UnauthorizedException 발생시키는  함수 작성하기
public boolean checkToken(String token) {
    try {
        Json Web Signature? 서버에서 인증을 근거로 인증 정보를 서버의 private key 서명 한것을 토큰화 한것
        setSigningKey : JWS 서명 검증을 위한  secret key 세팅
        parseClaimsJws : 파싱하여 원본 jws 만들기
        Jws<Claims> claims = Jwts.parser().verifyWith(getSigningKey()).build().parseSignedClaims(token);
        Claims 는 Map 구현체 형태
        log.debug("claims: {}", claims);

        return true;
    } catch (Exception e) {
        log.error(e.getMessage());
        return false;
    }
}
```

Intercepter에서 유효성 검사

```
if (accessToken != null) {
    accessToken = accessToken.replace("Bearer ", "");
    log.debug("HEADER_AUTH:{}", accessToken);
    if (jwtUtil.checkToken(accessToken)) {
        log.info("Access Token 사용 가능 : {}", accessToken);
        return true;
    }
}
```

5.4 JWT 토큰에서 필요한 정보 추출

```java
public String getUserId(String authorization ) {
    Jws<Claims> claims = null;
    try {
        claims = Jwts.parser().verifyWith(getSigningKey()).build().parseSignedClaims(authorization);
    } catch (Exception e) {
        log.error(e.getMessage());
        throw new UnAuthorizedException();
    }
    Map<String, Object> value = claims.getPayload();
    log.info("value : {}", value);
    return (String) value.get("userId");
}
```

5.5 access Token이 만료된 경우 refresh token으로 재발급 받기

```java
////TODO 10. Access Token 만료시 Refresh Token을 이용해 새로운 Access Token 발급하는 메서드 작성하기
public String generateAccessTokenFromRefreshToken(String refreshToken) {
    if (!checkToken(refreshToken)) {
        throw new UnAuthorizedException(); // Refresh Token이 유효하지 않으면 예외 발생
    }
    String userId = getUserId(refreshToken);
    return createAccessToken(userId); // 새로운 Access Token 생성
}
```