

JWT 토큰 생성 및 검사

Springboot JWT

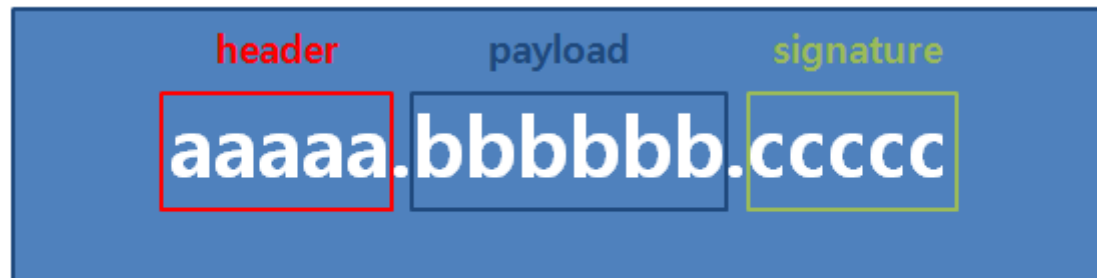
https://github.com/dron512/git231229/tree/main/springboot_work/jwtmh

1.1 JWT란

<https://jwt.io/>

1. JWT는 유저를 인증하고 식별하기 위한 토큰(Token) 기반 인증입니다.
2. 토큰 자체에 사용자의 권한 정보나 서비스를 사용하기 위한 정보가 포함됩니다.
3. RESTful과 같은 무상태(Stateless)인 환경에서 사용자 데이터를 주고받을 수 있게 됩니다.
4. 세션(Session)을 사용하게 될 경우 쿠키 등을 통해 사용자를 식별하고 서버에 세션을 저장했지만, 토큰을 클라이언트에 저장하고 요청시 HTTP 헤더에 토큰을 첨부하는 것만으로도 단순하게 데이터를 요청하고 응답받을 수 있습니다.

1. 2 JWT(Json Web Token) 구조



JWT 구조 예시 (출처: BizSpring)

1.3 JWT구조

(1) Header (헤더)

서명 시 사용하는 키(kid), 사용할 타입(typ), 서명 암호화 알고리즘(alg)의 정보가 담겨 있습니다.

```
HEADER: ALGORITHM & TOKEN TYPE

{
  "kid": "Key ID",
  "typ": "JWT",
  "alg": "ES256"
}
```

JWT Header 예시 (출처: BizSpring)

kid : 서명 시 사용하는 키(Public/Private Key)를 식별하는 값

typ : 토큰 유형

alg : 서명 암호화 알고리즘 HS256(HMAC SHA-256), HS512, RS256(RSASSA SHA-256), ES256(ECDSA P-256 curve SHA-256)

1.3 JWT구조

(2) Payload (페이로드)

```
PAYLOAD: DATA

{
  "iss": "growthplatform",
  "sub": "bizspring",
  "iat": 1702189795,
  "exp": 1702193395,
  "roles": "ROLE_SUPER"
}
```

JWT Payload 예시 (출처: BizSpring)

iss : 토큰 발급자(issuer) – Public Claims
sub : 토큰 제목(subject) – Public Claims
iat : 토큰 발급 시간(issued at) – Public Claims
exp : 토큰 만료 시간(expiration) – Public Claims
roles : 권한 – Private Claims

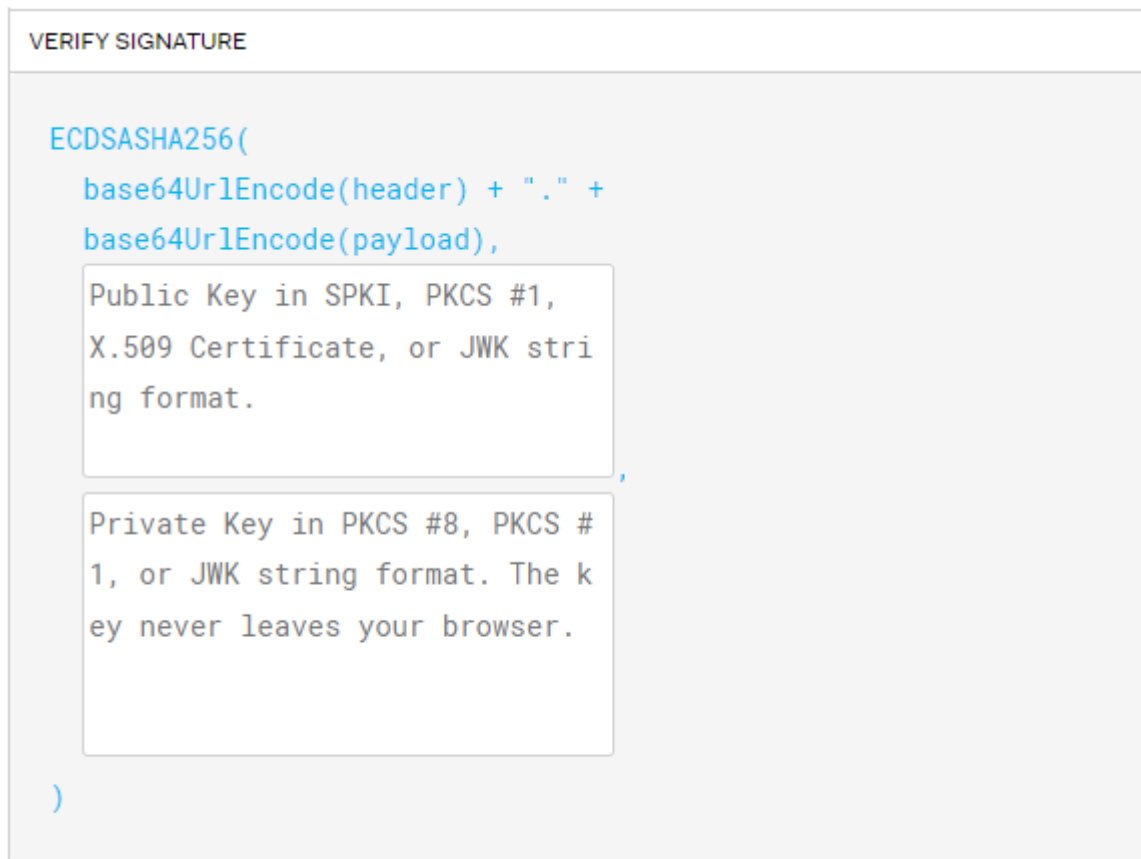
토큰에서 사용할 정보의 조각들인 클레임(Claim)이 담겨 있습니다.

클레임(Claim)은 Key/Value 형태로 된 값을 가집니다.

저장되는 정보에 따라 등록된 클레임(Registered Claims), 공개 클레임(Public Claims), 비공개 클레임(Private Claims)로 구분됩니다.

1.3 JWT구조

(3) Signature (서명)



JWT Signature 예시 (출처: BizSpring)

Header(헤더) 에서 정의한 알고리즘 방식(alg)을 활용합니다.

Header(헤더)+ 페이로드(Payload)와 서버가 갖고 있는 유일한 key 값을 합친 것을 헤더에서 정의한 알고리즘으로 암호화합니다.

Header(헤더) 와 페이로드(Payload)는 단순히 인코딩된 값이기 때문에 제 3자가 복호화 및 조작할 수 있지만, Signature(서명)는 서버 측에서 관리하는 비밀키가 유출되지 않는 이상 복호화할 수 없습니다. 이는 토큰의 위변조 여부를 확인하는데 사용됩니다.

1.1 JWT 생성

dependency 추가

```
<!-- https://mvnrepository.com/artifact/io.jsonwebtoken/jjwt-api -->
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt-api</artifactId>
  <version>0.12.5</version>
</dependency>
<!-- https://mvnrepository.com/artifact/io.jsonwebtoken/jjwt-impl -->
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt-impl</artifactId>
  <version>0.12.5</version>
  <scope>runtime</scope>
</dependency>
<!-- https://mvnrepository.com/artifact/io.jsonwebtoken/jjwt-jackson -->
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt-jackson</artifactId>
  <version>0.12.5</version>
  <scope>runtime</scope>
</dependency>
```



1.1 JWT 생성

Password는 application.properties 입력
@Value 사용해서 비밀번호 가져오기

```
spring.jwt.secret=passwordpasswordpasswordpasswordpasswordpassword
```

```
@Value("${spring.jwt.secret}")  
private String mykey;
```

1.1 JWT 생성

1 usage  dron512

```
public String buildToken(){  
    return Jwts.builder()  
        .subject(s: "token")  
        .expiration(createTokenExpiration())  
        .claim(s: "memberId", o: "member1")  
        .claim(s: "role", o: "User")  
        .signWith(Keys  
            .hmacShaKeyFor(Base64.getEncoder().encode(mykey.getBytes(StandardCharsets.UTF_8))))  
        .compact();  
}
```


1.1 JWT 유효성검사

[1 usage](#)  dron512 *

```
public void validateToken(String token){
    try{
        Jws<Claims> claimsJws = Jwts.parser() JwtParserBuilder
            .setSigningKey(Keys.hmacShaKeyFor(
                Base64.getEncoder().encode(mykey.getBytes(StandardCharsets.UTF_8))))
            .build() JwtParser
            .parseSignedClaims(token);

        System.out.println(claimsJws);
    }catch (ExpiredJwtException e){
        System.out.println("토큰만료");
    }catch (Exception e){
        System.out.println("유효하지 않은 토큰");
    }
}
```

1.1 인터셉터 등록

토큰을 발행하는 주소는 exclude 하도록 합니다.

```
drone512 *  
@Configuration  
@RequiredArgsConstructor  
public class TokenWebConfig implements WebMvcConfigurer {  
  
    private final TokenInterceptor tokenInterceptor;  
    no usages drone512 *  
    @Override  
    public void addInterceptors(InterceptorRegistry registry) {  
        registry.addInterceptor(tokenInterceptor)  
            .excludePathPatterns("/token");  
    }  
}
```

1.1 요청시에 Authorization해더 검사

```
dr0n512
@Component
@RequiredArgsConstructor
public class TokenInterceptor implements HandlerInterceptor {
    private final TokenManager tokenManager;
    no usages   dr0n512 *
    @Override
    public boolean preHandle(HttpServletRequest request,
                             HttpServletResponse response,
                             Object handler) throws Exception {
        String auth = request.getHeader("Authorization");
        try {
            Jws<Claims> claimsJws = tokenManager.validateToken(auth.split(regex: "Bearer ")[1]);
            System.out.println(claimsJws);
        } catch (ExpiredJwtException e) {
            throw new TokenException(ErrorCode.TOKEN_EXPIRED);
        }
        catch (Exception e) {
            throw new TokenException(ErrorCode.TOKEN_VALID);
        }
        return true;
    }
}
```

1.1 TokenException 추가

```
0 usages new
@Getter
public class TokenException extends RuntimeException{
    private ErrorCode errorCode;
    2 usages new *
    public TokenException(ErrorCode errorCode) {
        super(errorCode.getMessage());
        this.errorCode = errorCode;
    }
}
```

1.1 ErrorCode 추가

@Getter

```
public enum ErrorCode {
```

```
    TOKEN_EXPIRED(HttpStatus.UNAUTHORIZED, message: "토큰 에러", desc: "토큰 유효기간이 끝났습니다."),  
    TOKEN_VALID(HttpStatus.UNAUTHORIZED, message: "토큰 에러", desc: "토큰이 유효하지 않습니다.")  
    ;
```

```
    private HttpStatus httpStatus;  
    private String message;  
    private String desc;
```

4 usages  dron512 *

```
    ErrorCode(HttpStatus httpStatus, String message, String desc) {  
        this.httpStatus = httpStatus;  
        this.message = message;  
        this.desc = desc;  
    }
```

```
}
```

1.1 ErrorResponse 추가

@Getter

@Builder

```
public class ErrorResponse {
```

```
    private String errorCode;
```

```
    private String message;
```

```
1 usage    new *
```

```
public static ErrorResponse of(ErrorCode errorCode){
```

```
    return ErrorResponse.builder()
```

```
        .errorCode(errorCode.getHttpStatus().toString())
```

```
        .message(createErrorMessage(errorCode))
```

```
        .build();
```

```
}
```

```
1 usage    ⤴ dron512 *
```

```
public static String createErrorMessage(ErrorCode code)
```

```
{
```

```
    return String.format("[ %s %s]", code.getMessage(), code.getDesc());
```



```
}
```


```
}
```

1.1 토큰 테스트 Controller 수행

```
@RestController
@RequiredArgsConstructor
public class TokenTestController {

    private final TokenManager tokenManager;

     dron512 *
    @GetMapping( "token")
    public String getToken(){
        String result = tokenManager.buildToken();
        return result;
    }

    new *
    @GetMapping( "valid")
    public String getValid() { return "valid"; }
}
```

고맙습니다.

