Page No.
Date

JWT

Header . Payload . SIGNATURE

{
alg:
type:
}

{
subject:
issuedat:
exp:
claim:
}

ALGORITH(A.B; KEY)

↓

Base64Encode A . Base64Encode B . Base64Encode C

SECRETE = " "

KEY = KeyObject(SECRETE)

Verification:

* Jwts.parserBuilder().setSigningKey(key).build().parseClaimsJws(token)

Token Verification:

i) Server splits JWT into parts:
Header, payload, Signature.

ii) The Base64URL-encoded header is decoded to reveal the JSON, and the algorithm.
This tells which algorithm to use for verification.

iii) Recompute the signature with recived header, payload & key.

iv) Compare the signature

v) Check expiration.

Page No.
Date

1) Authentication [interface] =>
getPrincipal(), getCredentials(), getAuthorities(),
isAuthenticated()

⇓ [implementation class]

UsernamePasswordAuthenticationToken

i) (principal, credentials)
   ↓ username, ↓ password

ii) (principal, credentials, authorities)
   ↓ userDetails, ↓ null, ↓ roles

2) UserDetails [interface] =>
getUsername(), getPassword(), getAuthorities(),
isAccountNonExpired(), isAccountNonLocked()

⇓ Implementation

User(username, password, authorities)

OR

Custom UserDetails implements UserDetails

3) AuthenticationProvider [interface]

⇓ Implementation

DaoAuthenticationProvider

4) UserDetailsService

loadUserByUsername(username)

(3)

Page No.
Date

The backend websocket sends an logout event and front end listens to it and redirects to login page.

username ↑ password ↑

Authentication (principal, credentials)

[UsernamePasswordAuthenticationToken]

↓

AuthenticationManager.authenticate()

↓

AuthenticationProvider

↓

(1) AOAuthenticationProvider

↓

UserDetailService.LoadUser()

↓

Compare credentials

↓

Authentication(principal, credential, authorities)

↓ UserDetails ↓ null ↓ roles

[UsernamePasswordAuthenticationToken]

↓

Generate Token.