JWT

- **What is Jwt?**

->JWT, or JSON Web Token, is an open standard used to share security information between two parties — a client and a server. Each JWT contains encoded JSON objects, including a set of claims. JWTs are signed using a cryptographic algorithm to ensure that the claims cannot be altered after the token is issued.

How JWT Works

JWTs differ from other web tokens in that they contain a set of claims. Claims are used to transmit information between two parties. What these claims are depends on the use case at hand. For example, a claim may assert who issued the token, how long it is valid for, or what permissions the client has been granted.

A JWT is a string made up of three parts, separated by dots (.), and serialized using base64. In the most common serialization format, compact serialization, the JWT looks something like this:

xxxxx.yyyyy.zzzzz.

Once decoded, you will get two JSON strings:

1. The **header** and the **payload.**
2. The **signature.**

The **JOSE (JSON Object Signing and Encryption) header** contains the type of token — JWT in this case — and the signing algorithm.

The **payload** contains the claims. This is displayed as a JSON string, usually containing no more than a dozen fields to keep the JWT compact. This information is typically used by the server to verify that the user has permission to perform the action they are requesting.

There are no mandatory claims for a JWT, but overlaying standards may make claims mandatory. For example, when using JWT as bearer access token under OAuth2.0, iss, sub, aud, and exp must be present. some are more common than others.

The **signature** ensures that the token hasn’t been altered. The party that creates the JWT signs the header and payload with a secret that is known to both the issuer and receiver, or with a private key known only to the sender. When the token is used, the receiving party verifies that the header and payload match the signature.

Class for provide username and password

```java
package com.example.security;



public class AuthenticationRequest {

    private String username;
    private String password;
     public String getUsername() {
          return username;
     }
     public void setUsername(String username) {
          this.username = username;
     }
     public String getPassword() {
          return password;
     }
     public void setPassword(String password) {
          this.password = password;
     }


}
```

```
============================================================


package com.example.security;


import com.auth0.jwt.JWT;
import com.auth0.jwt.algorithms.Algorithm;
import com.auth0.jwt.interfaces.DecodedJWT;
import org.springframework.stereotype.Component;

import java.util.Date;

@Component
public class JwtTokenUtil {

    private static final String SECRET_KEY =
"mySecretKey";
    private static final long EXPIRATION_TIME = 1000 * 60
* 60; // 1 hour

    public String generateToken(String username) {
        Algorithm algorithm =
Algorithm.HMAC256(SECRET_KEY);
        return JWT.create()
                .withSubject(username)
                .withIssuedAt(new Date())
                .withExpiresAt(new
Date(System.currentTimeMillis() + EXPIRATION_TIME))
                .sign(algorithm);
    }

    public String extractUsername(String token) {
        return extractClaims(token).getSubject();
    }

    public DecodedJWT extractClaims(String token) {
        Algorithm algorithm =
Algorithm.HMAC256(SECRET_KEY);
        return JWT.require(algorithm)
                .build()
```

```
                .verify(token);
    }


 public boolean isTokenExpired(String token) {
        return
extractClaims(token).getExpiresAt().before(new Date());
    }

    public boolean validateToken(String token, String
username) {
        return (username.equals(extractUsername(token)) &&
!isTokenExpired(token));
    }
}
```

```
==========================================================
package com.example.security;


import java.io.IOException;
import java.util.ArrayList;

import
org.springframework.security.core.context.SecurityContextH
older;
import
org.springframework.web.filter.OncePerRequestFilter;

import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;

public class JwtRequestFilter extends OncePerRequestFilter
{

    private final JwtTokenUtil jwtTokenUtil;

    public JwtRequestFilter(JwtTokenUtil jwtTokenUtil) {
        this.jwtTokenUtil = jwtTokenUtil;
    }

    @Override
    protected void doFilterInternal(HttpServletRequest
request,HttpServletResponse response, FilterChain
filterChain) throws ServletException, IOException {
        String jwtToken =
request.getHeader("Authorization");

        if (jwtToken != null &&
jwtToken.startsWith("Bearer ")) {
            // Extract the token from the Authorization
header
            String token = jwtToken.substring(7);
```

```
            String username =
jwtTokenUtil.extractUsername(token);

            if (username != null &&
SecurityContextHolder.getContext().getAuthentication() ==
null) {
                // Validate token and set authentication
if valid
                if (jwtTokenUtil.validateToken(token,
username)) {

SecurityContextHolder.getContext().setAuthentication(new
org.springframework.security.authentication.UsernamePasswo
rdAuthenticationToken(username, null, new ArrayList<>()));
                }
            }
        }
        filterChain.doFilter(request, response);
    }
}
```

```
============================================================
package com.example.security;


import org.springframework.context.annotation.Bean;
import
org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.web.builder
s.HttpSecurity;
import org.springframework.security.core.userdetails.User;
import
org.springframework.security.core.userdetails.UserDetailsS
ervice;
import
org.springframework.security.crypto.bcrypt.BCryptPasswordE
ncoder;
import
org.springframework.security.crypto.password.PasswordEncod
er;
import
org.springframework.security.provisioning.InMemoryUserDeta
ilsManager;
import
org.springframework.security.web.SecurityFilterChain;
import
org.springframework.security.web.authentication.UsernamePa
sswordAuthenticationFilter;
import
org.springframework.security.config.annotation.web.configu
ration.EnableWebSecurity;

@Configuration
@EnableWebSecurity
public class WebSecurityConfig {

    private final JwtTokenUtil jwtTokenUtil;

    // Constructor-based injection of JwtTokenUtil
    public WebSecurityConfig(JwtTokenUtil jwtTokenUtil) {
```

```
        this.jwtTokenUtil = jwtTokenUtil;
    }

    @Bean
    public SecurityFilterChain
securityFilterChain(HttpSecurity http) throws Exception {
        // Disable CSRF for simplicity in this example
        http.csrf().disable()
            .authorizeRequests()
                .requestMatchers("/authenticate",
"/register","create","/get/**", "/delete/**").permitAll()
// Allow public access to login/register endpoints
                .anyRequest().authenticated()  // Require
authentication for all other requests
            .and()
            .addFilterBefore(new
JwtRequestFilter(jwtTokenUtil),
UsernamePasswordAuthenticationFilter.class);  // Register
JwtRequestFilter before
UsernamePasswordAuthenticationFilter

        return http.build();
    }

    @Bean
    public UserDetailsService userDetailsService() {
        // Example in-memory user (replace with a real
user service in production)
        return new InMemoryUserDetailsManager(
            User.withUsername("root")

.password(passwordEncoder().encode("admin"))
                .roles("USER")
                .build()
        );
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}
```