**JWT (Json web token)**

JSON Web Tokens are an open, industry standard RFC 7519 method for representing claims securely between two parties. This information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret (with the HMAC algorithm) or a public/private key pair using RSA or ECDSA.

Although JWTs can be encrypted to also provide secrecy between parties, we will focus on signed tokens. Signed tokens can verify the integrity of the claims contained within it, while encrypted tokens hide those claims from other parties. When tokens are signed using public/private key pairs, the signature also certifies that only the party holding the private key is the one that signed it.

**When JWT is used ?**

1. **Authorization**

Once user logged in or successfully authenticated, then in each subsequent request will include JWT token. It allows the user to access resources.

1. **Information Exchange**JSON web token are good way of securely transmitting information between two parties  
   because JWT can be signed by using public/private keys. you can be sure the senders are who they say they are. In addition the signature is calculated header and playload.

**What is the JSON Web Token structure?**

In its compact form JWT consists of three parts separated by (. dots). Which are

Header

Payload

Signature

Therefore the JWT token look like as follows

XXXXXXXX.XXXXXXXX.XXXXXXXXX

**1-Header :**

The header consist two parts, the type of the token, which is JWT, and the signing   
 algorithm being used, such as HMAC SHA256 or RSA.

For Example :

{

“alg”:”HS256”

“typ”:”jwt”

}

**2-Payload** The payload contains the claims. Claims provide information about an entity, typically   
 a user which has been authenticated and other data for example expiration time of   
 the token. There are three types of claims: Registered, Public and Private.

**Registered claims**- these are a set of predefined claims which are not mandatory but  
 recommended, to provide a set of useful, interoperable claims. Some of them are: **iss**   
 (issuer), **exp** (expiration time), **sub** (subject), **aud** (audience), and others.

**Public claims** -These can be defined at will by those using JWTs.

**Private claims**- These are the custom claims created to share information between   
 parties that agree on using them and are neither registered or public claims.

For Example :

{

"sub": "1234567890",

"name": "John Doe",

"admin": true

}

**3-Signature** To create the signature part you have to take the encoded header, the   
 encoded payload, a secret, the algorithm specified in the header, and sign   
 that.

For example if you want to use the HMAC SHA256 algorithm, the signature will be  
 created in the following way:

HMACSHA256(

base64UrlEncode(header) + "." +

base64UrlEncode(payload),

secret)

The signature is used to verify the message wasn't changed along the way, and, in the   
 case of tokens signed with a private key, it can also verify that the sender of the JWT is   
 who it says it is.

**Putting it all together:**

**eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV\_adQssw5c**

If you want to play with JWT and put these concepts into practice, you can use jwt.io   
 Debugger to decode, verify, and generate JWTs.

**How do JSON Web Tokens work?**

1-In authentication, when the user successfully logs in using their   
 credentials, a JSON Web Token will be returned.

2-Whenever the user wants to access a protected route or resource, the   
 user agent should send the JWT, typically in the Authorization header   
 using the Bearer schema.

Authorization: Bearer <token>

The server's protected routes will check for a valid JWT in the Authorization header,  
 and if it's present, the user will be allowed to access protected resources.

--------------------------------------------------------------------------------------------------------------------

**Spring Security with JWT**

We need to create a project “Spring-Security-JWT-Demo” in eclipse.

**Step 1**

Create a pom.xml file which having dependency as like

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.1.9.RELEASE</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>io.javabrains</groupId>

<artifactId>spring-security-jwt</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>spring-security-jwt</name>

<description>Demo project for Spring Boot</description>

<properties>

<java.version>1.8</java.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt</artifactId>

<version>0.9.1</version>

</dependency>

<dependency>

<groupId>javax.xml.bind</groupId>

<artifactId>jaxb-api</artifactId>

<version>2.3.0</version>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

**Step 2**

**Create a “SpringSecurityJWTApplication.java file which having main methods**

package com.jwt.demo

import com.jwt.demo.filters.JwtRequestFilter;

import com.jwt.demo.models.AuthenticationRequest;

import com.jwt.demo.models.AuthenticationResponse;

import com.jwt.demo.util.JwtUtil;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.annotation.Bean;

import org.springframework.http.ResponseEntity;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.authentication.BadCredentialsException;

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

import org.springframework.security.config.http.SessionCreationPolicy;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.security.core.userdetails.UserDetailsService;

import org.springframework.security.crypto.password.NoOpPasswordEncoder;

import org.springframework.security.crypto.password.PasswordEncoder;

import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication

public class SpringSecurityJWTApplication {

public static void main(String[] args) {

SpringApplication.run(SpringSecurityJWTApplication.class, args);

}

}

@RestController

class HelloWorldController {

@Autowired

private AuthenticationManager authenticationManager;

@Autowired

private JwtUtil jwtTokenUtil;

@Autowired

private MyUserDetailsService userDetailsService;

@RequestMapping({ "/hello" })

public String firstPage() {

return "Hello World";

}

@RequestMapping(value = "/authenticate", method = RequestMethod.POST)

public ResponseEntity<?> createAuthenticationToken(@RequestBody AuthenticationRequest authenticationRequest) throws Exception {

try {

authenticationManager.authenticate(

new UsernamePasswordAuthenticationToken(authenticationRequest.getUsername(), authenticationRequest.getPassword())

);

}

catch (BadCredentialsException e) {

throw new Exception("Incorrect username or password", e);

}

final UserDetails userDetails = userDetailsService

.loadUserByUsername(authenticationRequest.getUsername());

final String jwt = jwtTokenUtil.generateToken(userDetails);

return ResponseEntity.ok(new AuthenticationResponse(jwt));

}

}

@EnableWebSecurity

class WebSecurityConfig extends WebSecurityConfigurerAdapter {

@Autowired

private UserDetailsService myUserDetailsService;

@Autowired

private JwtRequestFilter jwtRequestFilter;

@Autowired

public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {

auth.userDetailsService(myUserDetailsService);

}

@Bean

public PasswordEncoder passwordEncoder() {

return NoOpPasswordEncoder.getInstance();

}

@Override

@Bean

public AuthenticationManager authenticationManagerBean() throws Exception {

return super.authenticationManagerBean();

}

@Override

protected void configure(HttpSecurity httpSecurity) throws Exception {

httpSecurity.csrf().disable()

.authorizeRequests().antMatchers("/authenticate").permitAll().

anyRequest().authenticated().and().

exceptionHandling().and().sessionManagement()

.sessionCreationPolicy(SessionCreationPolicy.STATELESS);

httpSecurity.addFilterBefore(jwtRequestFilter, UsernamePasswordAuthenticationFilter.class);

}

}

**Step 3   
 Create MyUserDetailsService.java**

package com.jwt.demo

import org.springframework.security.core.userdetails.User;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.security.core.userdetails.UserDetailsService;

import org.springframework.security.core.userdetails.UsernameNotFoundException;

import org.springframework.stereotype.Service;

import java.util.ArrayList;

@Service

public class MyUserDetailsService implements UserDetailsService {

@Override

public UserDetails loadUserByUsername(String s) throws UsernameNotFoundException {

return new User("test", "test",

new ArrayList<>());

}

}

**Step 4**

package com.jwt.demo.models

import java.io.Serializable;

public class AuthenticationRequest implements Serializable {

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;

}

//need default constructor for JSON Parsing

public AuthenticationRequest()

{

}

public AuthenticationRequest(String username, String password) {

this.setUsername(username);

this.setPassword(password);

}

}

**Step 5**  
package com.jwt.demo.models

import java.io.Serializable;

public class AuthenticationResponse implements Serializable {

private final String jwt;

public AuthenticationResponse(String jwt) {

this.jwt = jwt;

}

public String getJwt() {

return jwt;

}

}

**Step 6**

package com.jwt.demo.util

import io.jsonwebtoken.Claims;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.stereotype.Service;

import java.util.Date;

import java.util.HashMap;

import java.util.Map;

import java.util.function.Function;

@Service

public class JwtUtil {

private String SECRET\_KEY = "secret";

public String extractUsername(String token) {

return extractClaim(token, Claims::getSubject);

}

public Date extractExpiration(String token) {

return extractClaim(token, Claims::getExpiration);

}

public <T> T extractClaim(String token, Function<Claims, T> claimsResolver) {

final Claims claims = extractAllClaims(token);

return claimsResolver.apply(claims);

}

private Claims extractAllClaims(String token) {

return Jwts.parser().setSigningKey(SECRET\_KEY).parseClaimsJws(token).getBody();

}

private Boolean isTokenExpired(String token) {

return extractExpiration(token).before(new Date());

}

public String generateToken(UserDetails userDetails) {

Map<String, Object> claims = new HashMap<>();

return createToken(claims, userDetails.getUsername());

}

private String createToken(Map<String, Object> claims, String subject) {

return Jwts.builder().setClaims(claims).setSubject(subject).setIssuedAt(new Date(System.currentTimeMillis()))

.setExpiration(new Date(System.currentTimeMillis() + 1000 \* 60 \* 60 \* 10))

.signWith(SignatureAlgorithm.HS256, SECRET\_KEY).compact();

}

public Boolean validateToken(String token, UserDetails userDetails) {

final String username = extractUsername(token);

return (username.equals(userDetails.getUsername()) && !isTokenExpired(token));

}

}

**Step 7**

package com.jwt.demo.filters

import com.jwt.demo.MyUserDetailsService;

import com.jwt.demo.util.JwtUtil;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

import org.springframework.security.core.context.SecurityContextHolder;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;

import org.springframework.stereotype.Component;

import org.springframework.web.filter.OncePerRequestFilter;

import javax.servlet.FilterChain;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import java.io.IOException;

@Component

public class JwtRequestFilter extends OncePerRequestFilter {

@Autowired

private MyUserDetailsService userDetailsService;

@Autowired

private JwtUtil jwtUtil;

@Override

protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain chain)

throws ServletException, IOException {

final String authorizationHeader = request.getHeader("Authorization");

String username = null;

String jwt = null;

if (authorizationHeader != null && authorizationHeader.startsWith("Bearer ")) {

jwt = authorizationHeader.substring(7);

username = jwtUtil.extractUsername(jwt);

}

if (username != null && SecurityContextHolder.getContext().getAuthentication() == null) {

UserDetails userDetails = this.userDetailsService.loadUserByUsername(username);

if (jwtUtil.validateToken(jwt, userDetails)) {

UsernamePasswordAuthenticationToken usernamePasswordAuthenticationToken = new UsernamePasswordAuthenticationToken(

userDetails, null, userDetails.getAuthorities());

usernamePasswordAuthenticationToken

.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));

SecurityContextHolder.getContext().setAuthentication(usernamePasswordAuthenticationToken);

}

}

chain.doFilter(request, response);

}

}