Creating a JWT Authentication Service

Step 1: Project Setup

xml

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt-api</artifactId>

<version>0.11.5</version>

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt-impl</artifactId>

<version>0.11.5</version>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt-jackson</artifactId>

<version>0.11.5</version>

<scope>runtime</scope>

</dependency>

</dependencies>

Step 2: Create JWT Utility Class

import io.jsonwebtoken.Claims;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

import io.jsonwebtoken.security.Keys;

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

import org.springframework.stereotype.Service;

import java.security.Key;

import java.util.Date;

import java.util.HashMap;

import java.util.Map;

import java.util.function.Function;

@Service

public class JwtUtil {

private final Key secretKey = Keys.secretKeyFor(SignatureAlgorithm.HS256);

private final long expirationMs = 3600000; // 1 hour

public String generateToken(String username) {

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

return createToken(claims, username);

}

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() + expirationMs))

.signWith(secretKey)

.compact();

}

public Boolean validateToken(String token, UserDetails userDetails) {

final String username = extractUsername(token);

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

}

private Boolean isTokenExpired(String token) {

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

}

public String extractUsername(String token) {

return extractClaim(token, Claims::getSubject);

}

public Date extractExpiration(String token) {

return extractClaim(token, Claims::getExpiration);

}

private <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.parserBuilder()

.setSigningKey(secretKey)

.build()

.parseClaimsJws(token)

.getBody();

}

}

Step 3: Create Authentication Controller

import org.springframework.http.ResponseEntity;

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

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

import org.springframework.security.provisioning.InMemoryUserDetailsManager;

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

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

@RestController

public class AuthenticationController {

private final JwtUtil jwtUtil;

private final InMemoryUserDetailsManager userDetailsManager;

public AuthenticationController(JwtUtil jwtUtil, InMemoryUserDetailsManager userDetailsManager) {

this.jwtUtil = jwtUtil;

this.userDetailsManager = userDetailsManager;

// Create a default user (in production, use a proper UserDetailsService)

UserDetails user = User.withDefaultPasswordEncoder()

.username("user")

.password("pwd")

.roles("USER")

.build();

userDetailsManager.createUser(user);

}

@PostMapping("/authenticate")

public ResponseEntity<?> createAuthenticationToken() {

// The user is already authenticated by Spring Security at this point

final UserDetails userDetails = userDetailsManager.loadUserByUsername(

SecurityContextHolder.getContext().getAuthentication().getName());

final String token = jwtUtil.generateToken(userDetails.getUsername());

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

}

// Simple DTO for the response

private static class AuthenticationResponse {

private final String token;

public AuthenticationResponse(String token) {

this.token = token;

}

public String getToken() {

return token;

}

}

}

Step 4: Configure Security

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

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

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

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

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

import org.springframework.security.provisioning.InMemoryUserDetailsManager;

import org.springframework.security.web.authentication.www.BasicAuthenticationFilter;

@Configuration

public class SecurityConfig extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http

.csrf().disable()

.authorizeRequests()

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

.anyRequest().authenticated()

.and()

.httpBasic();

}

@Bean

@Override

public InMemoryUserDetailsManager userDetailsService() {

// This will be overridden in the controller, but needed for startup

UserDetails user = User.withDefaultPasswordEncoder()

.username("temp")

.password("temp")

.roles("USER")

.build();

return new InMemoryUserDetailsManager(user);

}

}

Step 5: Main Application Class

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class JwtAuthServiceApplication {

public static void main(String[] args) {

SpringApplication.run(JwtAuthServiceApplication.class, args);

}

}

Testing the Service

Start the application (it will run on port 8080 by default)

Execute the curl command:

curl -s -u user:pwd http://localhost:8080/authenticate

You should receive a response like:

{"token":"eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1c2VyIiwiaWF0IjoxNTcwMzc5NDc0LCJleHAiOjE1NzAzODA2NzR9.t3LRvlCV-hwKfoqZYlaVQqEUiBloWcWn0ft3tgv0dL0"}

Key Points:

The service uses Basic Authentication for the initial credential verification

Upon successful authentication, it generates a JWT token

The token contains:

Subject (username)

Issued at timestamp

Expiration time (1 hour)

The secret key is generated automatically on startup