# Create authentication service that returns JWT

## JwtUtil.java

package com.cognizant.spring\_learn.security;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

import io.jsonwebtoken.security.Keys; // <-- New Import

import org.springframework.stereotype.Component;

import java.nio.charset.StandardCharsets; // <-- New Import

import java.security.Key; // <-- New Import

import java.util.Date;

import java.util.HashMap;

import java.util.Map;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

*@Component*

public class JwtUtil {

private static final Logger ***LOGGER*** = LoggerFactory.*getLogger*(JwtUtil.class);

private static final String ***SECRET\_KEY*** = "your-secret-key-that-is-long-enough";

private final Key key = Keys.*hmacShaKeyFor*(***SECRET\_KEY***.getBytes(StandardCharsets.***UTF\_8***));

public String generateToken(String username) {

***LOGGER***.info("START");

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

String token = Jwts.*builder*()

.setClaims(claims)

.setSubject(username)

.setIssuedAt(new Date(System.*currentTimeMillis*()))

.setExpiration(new Date(System.*currentTimeMillis*() + 1000 \* 60 \* 30)) // 30 minutes validity

.signWith(key, *SignatureAlgorithm*.***HS256***) // <-- CORRECTED LINE

.compact();

***LOGGER***.debug("Generated Token: {}", token);

***LOGGER***.info("END");

return token;

}

}

## 

## AuthenticationResponse.java

package com.cognizant.spring\_learn.controller;

import com.cognizant.spring\_learn.security.AuthenticationResponse;

import com.cognizant.spring\_learn.security.JwtUtil;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import org.springframework.security.core.Authentication;

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

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

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

*@RestController*

public class AuthenticationController {

private static final Logger ***LOGGER*** = LoggerFactory.*getLogger*(AuthenticationController.class);

*@Autowired*

private JwtUtil jwtUtil;

*@GetMapping*("/authenticate")

public AuthenticationResponse authenticate(Authentication authentication) {

***LOGGER***.info("START");

// Spring Security provides the authenticated user's details

// in the 'authentication' object after processing the HTTP Basic Auth header.

UserDetails userDetails = (UserDetails) authentication.getPrincipal();

String username = userDetails.getUsername();

// Generate the JWT using our utility class

String token = jwtUtil.generateToken(username);

// Return the token in the response object

AuthenticationResponse response = new AuthenticationResponse(token);

***LOGGER***.debug("Response: {}", response);

***LOGGER***.info("END");

return response;

}

}

## SecurityConfig.java

package com.cognizant.spring\_learn.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.EnableWebSecurity;

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

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

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

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

import org.springframework.security.provisioning.InMemoryUserDetailsManager;

import org.springframework.security.web.SecurityFilterChain;

*@Configuration*

*@EnableWebSecurity*

public class SecurityConfig {

*@Bean*

public InMemoryUserDetailsManager userDetailsService() {

UserDetails user = User.*builder*()

.username("user")

.password(passwordEncoder().encode("pwd"))

.roles("USER")

.build();

return new InMemoryUserDetailsManager(user);

}

*@Bean*

public PasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

*@Bean*

public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

http

.~~csrf~~().disable()

.authorizeHttpRequests(authz -> authz

.requestMatchers("/authenticate").authenticated()

.anyRequest().permitAll()

)

.~~httpBasic~~();

return http.build();

}

}

# **Output**

