**Create authentication service that returns JWT**

**pom.xml**

<?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>2.7.5</version>

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

    </parent>

    <groupId>com.cognizant.jwt</groupId>

    <artifactId>jwt-auth-service</artifactId>

    <version>0.0.1-SNAPSHOT</version>

    <name>jwt-auth-service</name>

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

    <properties>

        <java.version>17</java.version>

    </properties>

    <dependencies>

        <!-- Spring Boot Starter Web provides the necessary dependencies for building web applications, including RESTful services. -->

        <dependency>

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

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

        </dependency>

        <!-- Spring Boot Starter Security provides the security framework to protect your application. -->

        <dependency>

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

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

        </dependency>

        <!-- JJWT library is used for creating and parsing JSON Web Tokens. -->

        <dependency>

            <groupId>io.jsonwebtoken</groupId>

            <artifactId>jjwt</artifactId>

            <version>0.9.1</version>

        </dependency>

        <!-- Spring Boot Starter Test is for writing test cases. -->

        <dependency>

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

            <artifactId>spring-boot-starter-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>

**SecurityConfig.java**

package com.cognizant.jwt;

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

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.authentication.AuthenticationManager;

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.http.SessionCreationPolicy;

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.SecurityFilterChain;

@Configuration

@EnableWebSecurity

public class SecurityConfig {

    @Autowired

    private UserDetailsService userDetailsService;

    // This method configures the SecurityFilterChain, which is the core of Spring Security's web protection.

    @Bean

    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

        http.csrf().disable() // Disables CSRF protection for this demo

                .authorizeRequests()

                .antMatchers("/authenticate").permitAll() // Allows anyone to access the /authenticate endpoint

                .anyRequest().authenticated() // All other requests must be authenticated

                .and()

                .httpBasic() // Enables HTTP Basic Authentication

                .and()

                .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS); // Configures the session to be stateless

        return http.build();

    }

    // This method configures the AuthenticationManager, which is used to authenticate users.

    @Bean

    public AuthenticationManager authenticationManager(HttpSecurity http) throws Exception {

        return http.getSharedObject(AuthenticationManagerBuilder.class)

                .userDetailsService(userDetailsService) // We use our custom UserDetailsService to load user details

                .passwordEncoder(passwordEncoder())

                .and()

                .build();

    }

    // For this simple demo, we use a NoOpPasswordEncoder.

    // In a real application, you should use a strong password encoder like BCryptPasswordEncoder.

    @Bean

    public PasswordEncoder passwordEncoder() {

        return NoOpPasswordEncoder.getInstance();

    }

}

**JwtUserDetailsService.java**

package com.cognizant.jwt;

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 JwtUserDetailsService implements UserDetailsService {

    // This method is called by Spring Security to get user details.

    @Override

    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

        // For this demo, we'll hardcode a user.

        if ("user".equals(username)) {

            return new User("user", "pwd",

                    new ArrayList<>());

        } else {

            throw new UsernameNotFoundException("User not found with username: " + username);

        }

    }

}

**JwtTokenUtil.java**

package com.cognizant.jwt;

import io.jsonwebtoken.Claims;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

import org.springframework.stereotype.Component;

import java.util.Date;

import java.util.function.Function;

@Component

public class JwtTokenUtil {

    // The secret key is used to sign the token. It must be strong and kept private.

    // For this demo, we use a simple string.

    public static final String SECRET\_KEY = "my\_secret\_key\_for\_this\_demo\_application";

    public static final long JWT\_TOKEN\_VALIDITY = 5 \* 60 \* 60; // 5 hours

    // Retrieve username from jwt token

    public String getUsernameFromToken(String token) {

        return getClaimFromToken(token, Claims::getSubject);

    }

    // Retrieve expiration date from jwt token

    public Date getExpirationDateFromToken(String token) {

        return getClaimFromToken(token, Claims::getExpiration);

    }

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

        final Claims claims = getAllClaimsFromToken(token);

        return claimsResolver.apply(claims);

    }

    // For retrieving any information from token, we will need the secret key

    private Claims getAllClaimsFromToken(String token) {

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

    }

    // Check if the token has expired

    private Boolean isTokenExpired(String token) {

        final Date expiration = getExpirationDateFromToken(token);

        return expiration.before(new Date());

    }

    // Generate token for user

    public String generateToken(String username) {

        // Here you can add claims like roles, email etc.

        // For this demo, we just add the subject (username).

        Claims claims = Jwts.claims().setSubject(username);

        return doGenerateToken(claims, username);

    }

    // While creating the token:

    // 1. Define claims of the token, like Issuer, Expiration, Subject, and the ID

    // 2. Sign the JWT using the HS256 algorithm and the secret key.

    // 3. According to JWS Compact Serialization(https://tools.ietf.org/html/draft-ietf-jose-json-web-signature-41#section-3.1)

    //    compaction of the JWT to a URL-safe string

    private String doGenerateToken(Claims claims, String subject) {

        return Jwts.builder()

                .setClaims(claims)

                .setSubject(subject)

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

                .setExpiration(new Date(System.currentTimeMillis() + JWT\_TOKEN\_VALIDITY \* 1000))

                .signWith(SignatureAlgorithm.HS256, SECRET\_KEY)

                .compact();

    }

    // Validate token

    public Boolean validateToken(String token, UserDetails userDetails) {

        final String username = getUsernameFromToken(token);

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

    }

}

**JwtAuthenticationController.java**

package com.cognizant.jwt;

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

import org.springframework.http.ResponseEntity;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

import org.springframework.security.core.Authentication;

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

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

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

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

import java.util.Base64;

@RestController

public class JwtAuthenticationController {

    @Autowired

    private AuthenticationManager authenticationManager;

    @Autowired

    private JwtTokenUtil jwtTokenUtil;

    @Autowired

    private JwtUserDetailsService userDetailsService;

    // This method handles the POST request to /authenticate.

    // It reads the Authorization header, decodes the username and password,

    // and then generates a JWT token upon successful authentication.

    @PostMapping("/authenticate")

    public ResponseEntity<?> createAuthenticationToken(@RequestHeader("Authorization") String authorizationHeader) throws Exception {

        // The Authorization header is in the format "Basic base64EncodedUsernamePassword"

        // We need to extract the base64 part and decode it.

        String base64Credentials = authorizationHeader.substring("Basic".length()).trim();

        byte[] decodedCredentials = Base64.getDecoder().decode(base64Credentials);

        String credentials = new String(decodedCredentials);

        // The decoded string is "username:password"

        String[] parts = credentials.split(":", 2);

        String username = parts[0];

        String password = parts[1];

        // Authenticate the user with Spring Security's AuthenticationManager.

        // This will call our JwtUserDetailsService to load the user and verify the password.

        final Authentication authentication = authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(username, password));

        // If authentication is successful, generate a JWT token.

        final UserDetails userDetails = userDetailsService.loadUserByUsername(username);

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

        // Return the token in the response body.

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

    }

}

**JwtResponse.java**

package com.cognizant.jwt;

import java.io.Serializable;

// This class is a simple POJO to hold the JWT token in the response.

public class JwtResponse implements Serializable {

    private static final long serialVersionUID = -8091879091924046844L;

    private final String token;

    public JwtResponse(String token) {

        this.token = token;

    }

    public String getToken() {

        return this.token;

    }

}

**JwtAuthServiceApplication.java**

package com.cognizant.jwt;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.annotation.ComponentScan;

@SpringBootApplication

@ComponentScan(basePackages = "com.cognizant.jwt")

public class JwtAuthServiceApplication {

    public static void main(String[] args) {

        SpringApplication.run(JwtAuthServiceApplication.class, args);

    }

}