**1.CREATE AUTHENTICATION SERVICE THAT RETURNS JWT**

CONTEXT OF THE EXAMPLE:

In this exercise, we **create a Spring Boot REST API** that exposes an endpoint to **authenticate users** based on credentials. If the credentials are valid, the service **generates a JWT token** using a utility class and returns it in the response. This token can then be used for accessing secured APIs.

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>3.5.3</version>  
 <relativePath/> <!-- lookup parent from repository -->  
 </parent>  
 <groupId>com.dinesh</groupId>  
 <artifactId>Sboot7</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <name>Sboot7</name>  
 <description>Demo project for Spring Boot</description>  
 <url/>  
 <licenses>  
 <license/>  
 </licenses>  
 <developers>  
 <developer/>  
 </developers>  
 <scm>  
 <connection/>  
 <developerConnection/>  
 <tag/>  
 <url/>  
 </scm>  
 <properties>  
 <java.version>17</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-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> <!-- or jjwt-gson if you prefer -->  
 <version>0.11.5</version>  
 <scope>runtime</scope>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-devtools</artifactId>  
 <scope>runtime</scope>  
 <optional>true</optional>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-security</artifactId>  
 </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>

**config:**

Securityconfig.java:

package com.dinesh.Sboot7.config;  
  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.security.authentication.AuthenticationManager;  
import org.springframework.security.authentication.ProviderManager;  
import org.springframework.security.authentication.dao.DaoAuthenticationProvider;  
import org.springframework.security.config.annotation.web.builders.HttpSecurity;  
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.crypto.bcrypt.BCryptPasswordEncoder;  
import org.springframework.security.crypto.password.PasswordEncoder;  
import org.springframework.security.provisioning.InMemoryUserDetailsManager;  
import org.springframework.security.web.SecurityFilterChain;  
  
@Configuration  
public class SecurityConfig {  
  
 @Bean  
 public UserDetailsService 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 AuthenticationManager authenticationManager(UserDetailsService userDetailsService) {  
 DaoAuthenticationProvider provider = new DaoAuthenticationProvider();  
 provider.setUserDetailsService(userDetailsService);  
 provider.setPasswordEncoder(passwordEncoder());  
 return new ProviderManager(provider);  
 }  
  
 @Bean  
 public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {  
 http  
 .csrf().disable()  
 .authorizeHttpRequests()  
 .requestMatchers("/authenticate").permitAll()  
 .anyRequest().authenticated()  
 .and()  
 .httpBasic(); // Enables Basic Auth  
 return http.build();  
 }  
}

**controller:**

**AuthController.java:**

package com.dinesh.Sboot7.controller;  
  
import com.dinesh.Sboot7.util.JwtUtil;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.Base64;  
  
import jakarta.servlet.http.HttpServletRequest;  
  
@RestController  
public class AuthController {  
  
 private final JwtUtil jwtUtil;  
  
 public AuthController(JwtUtil jwtUtil) {  
 this.jwtUtil = jwtUtil;  
 }  
  
 @RequestMapping(value = "/authenticate", method = RequestMethod.*GET*)  
 public String authenticate(HttpServletRequest request) {  
 String authHeader = request.getHeader("Authorization");  
  
 if (authHeader != null && authHeader.startsWith("Basic ")) {  
 String base64Credentials = authHeader.substring("Basic ".length());  
 byte[] credDecoded = Base64.*getDecoder*().decode(base64Credentials);  
 String credentials = new String(credDecoded);  
 String[] userDetails = credentials.split(":", 2);  
  
 String username = userDetails[0];  
 // You can validate the password here, skipped for simplicity  
  
 String token = jwtUtil.generateToken(username);  
 return "{\"token\": \"" + token + "\"}";  
 }  
 throw new RuntimeException("Missing or invalid Authorization header");  
 }  
}

application.properties:

server.port=8090  
spring.security.user.name=user  
spring.security.user.password=pwd

**model:**

AuthRequest.java:

package com.dinesh.sboot7.model;  
  
public class AuthRequest {  
 private String username;  
 private String password;  
  
 public AuthRequest() { }  
  
 public AuthRequest(String username, String password) {  
 this.username = username;  
 this.password = password;  
 }  
  
 // Getters and setters  
 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;  
 }  
}

AuthResponse.java

package com.dinesh.sboot7.model;  
  
public class AuthResponse {  
 private String token;  
  
 public AuthResponse() { }  
  
 public AuthResponse(String token) {  
 this.token = token;  
 }  
  
 // Getter and setter  
 public String getToken() {  
 return token;  
 }  
 public void setToken(String token) {  
 this.token = token;  
 }  
}

**util:**

JwtUtil.java:

package com.dinesh.Sboot7.util;  
  
import io.jsonwebtoken.Jwts;  
import io.jsonwebtoken.SignatureAlgorithm;  
import org.springframework.stereotype.Component;  
  
import java.util.Date;  
  
@Component  
public class JwtUtil {  
  
 private final String secret = "mySecretKey"; // Use env variable in real app  
 private final long expirationMs = 1000 \* 60 \* 60; // 1 hour  
  
 public String generateToken(String username) {  
 return Jwts.*builder*()  
 .setSubject(username)  
 .setIssuedAt(new Date(System.*currentTimeMillis*()))  
 .setExpiration(new Date(System.*currentTimeMillis*() + expirationMs))  
 .signWith(SignatureAlgorithm.*HS256*, secret)  
 .compact();  
 }  
}

Sboot7Application.java:

package com.dinesh.Sboot7;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
@SpringBootApplication  
public class Sboot7Application {  
 public static void main(String[] args) {  
 SpringApplication.*run*(Sboot7Application.class, args);  
 }  
}

OUTPUT:

