**Week 4 – Hands-on : JWT hands-on**

Exercise : Create authentication service that returns JWT

1. Spring Security Setup : We add Spring Security dependencies and JWT dependency to “pom.xml”.

****

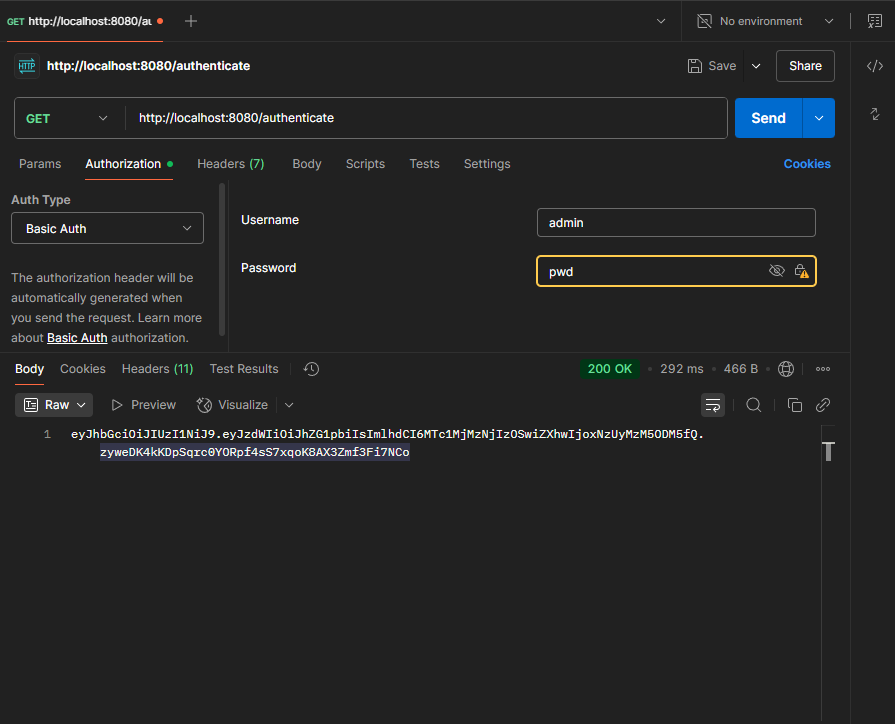
2. Create a package “security” and create class “SecurityConfig.java”

package com.cognizant.springlearn.config;  
  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.security.authentication.AuthenticationManager;  
import org.springframework.security.config.Customizer;  
import org.springframework.security.config.annotation.authentication.configuration.AuthenticationConfiguration;  
import org.springframework.security.config.annotation.web.builders.HttpSecurity;  
import org.springframework.security.core.userdetails.User;  
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 InMemoryUserDetailsManager userDetailsService(PasswordEncoder passwordEncoder) {  
 return new InMemoryUserDetailsManager(  
 User.*withUsername*("admin").password(passwordEncoder.encode("pwd")).roles("ADMIN").build(),  
 User.*withUsername*("user").password(passwordEncoder.encode("pwd")).roles("USER").build()  
 );  
 }  
  
 @Bean  
 public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {  
 http  
 .csrf(csrf -> csrf.disable())  
 .authorizeHttpRequests(auth -> auth  
 .requestMatchers("/authenticate").authenticated()  
 .anyRequest().permitAll()  
 )  
 .httpBasic(Customizer.*withDefaults*());  
  
 return http.build();  
 }  
  
 @Bean  
 public AuthenticationManager authenticationManager(AuthenticationConfiguration config) throws Exception {  
 return config.getAuthenticationManager();  
 }  
  
 @Bean  
 public PasswordEncoder passwordEncoder() {  
 return new BCryptPasswordEncoder();  
 }  
}

3. Build /authenticate endpoint, In the controller package create a new class “AuthenticationController.java”

package com.cognizant.springlearn.controller;  
  
import io.jsonwebtoken.Jwts;  
import io.jsonwebtoken.SignatureAlgorithm;  
import io.jsonwebtoken.security.Keys;  
import org.springframework.http.ResponseEntity;  
import org.springframework.security.core.Authentication;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
import java.security.Key;  
import java.util.Date;  
  
@RestController  
public class AuthenticationController {  
  
 private static final Key *key* = Keys.*hmacShaKeyFor*("mysecretkeymysecretkeymysecretkey!".getBytes());  
  
 @GetMapping("/authenticate")  
 public ResponseEntity<String> authenticate(Authentication authentication) {  
 String jwt = Jwts.*builder*()  
 .setSubject(authentication.getName())  
 .setIssuedAt(new Date())  
 .setExpiration(new Date(System.*currentTimeMillis*() + 3600000)) // 1 hour  
 .signWith(*key*, SignatureAlgorithm.*HS256*)  
 .compact();  
  
 return ResponseEntity.*ok*(jwt);  
 }  
}

4. Then we test this endpoint using Postman by creating a Basic Auth header



5. As this is working as expected we go forward a step and proceed with protecting endpoints using JWT

6. We create a custom “JwtAuthorizationFilter.java” file in security package

package com.cognizant.springlearn.security;  
  
import io.jsonwebtoken.Claims;  
import io.jsonwebtoken.Jwts;  
import jakarta.servlet.FilterChain;  
import jakarta.servlet.ServletException;  
import jakarta.servlet.http.HttpServletRequest;  
import jakarta.servlet.http.HttpServletResponse;  
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;  
import org.springframework.security.core.context.SecurityContextHolder;  
import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;  
import org.springframework.web.filter.OncePerRequestFilter;  
  
import java.io.IOException;  
import java.util.Collections;  
  
public class JwtAuthorizationFilter extends OncePerRequestFilter {  
  
 private final String SECRET\_KEY = "mysecretkeymysecretkeymysecretkey!";  
  
 @Override  
 protected void doFilterInternal(HttpServletRequest request,  
 HttpServletResponse response,  
 FilterChain filterChain)  
 throws ServletException, IOException {  
  
 String authHeader = request.getHeader("Authorization");  
  
 if (authHeader != null && authHeader.startsWith("Bearer ")) {  
 String token = authHeader.substring(7);  
  
 try {  
 Claims claims = Jwts.*parserBuilder*()  
 .setSigningKey(SECRET\_KEY.getBytes())  
 .build()  
 .parseClaimsJws(token)  
 .getBody();  
  
 String username = claims.getSubject();  
  
 if (username != null && SecurityContextHolder.*getContext*().getAuthentication() == null) {  
 UsernamePasswordAuthenticationToken authToken =  
 new UsernamePasswordAuthenticationToken(username, null, Collections.*emptyList*());  
  
 authToken.setDetails(  
 new WebAuthenticationDetailsSource().buildDetails(request)  
 );  
  
 SecurityContextHolder.*getContext*().setAuthentication(authToken);  
 }  
  
 } catch (Exception e) {  
 response.sendError(HttpServletResponse.*SC\_UNAUTHORIZED*, "Invalid JWT Token");  
 return;  
 }  
 }  
  
 filterChain.doFilter(request, response); }}

7. Update the “SecurityConfig.java” to import the filter

8. Update the “securityFilterChain” method as

@Bean  
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {  
 http  
 .csrf(csrf -> csrf.disable())  
 .authorizeHttpRequests(auth -> auth  
 .requestMatchers("/authenticate").permitAll()  
 .requestMatchers("/protected").authenticated()  
 .anyRequest().permitAll()  
 )  
 .addFilterBefore(new JwtAuthorizationFilter(), UsernamePasswordAuthenticationFilter.class)  
 .httpBasic(Customizer.*withDefaults*());  
  
 return http.build();  
}

9. Create a Protected endpoint to test JWT in AuthenticationController.java

@GetMapping("/protected")  
public ResponseEntity<String> protectedResource() {  
 return ResponseEntity.*ok*("You have accessed a PROTECTED endpoint with a valid JWT!");  
}

10. Finally we test this protected endpoint in Postman by authentication type “Bearer Token” and add the token we got from the previous “/authenticate” endpoint.

