**WEEK-4\_JWT-HandsOn**

**SUPERSET ID:6393674**

**Create authentication service that returns JWT**

**SOLUTION:**

**AuthRequest.java**package com.example.jwtauth.model;  
  
public class AuthRequest {  
 private String username;  
 private String password;  
  
 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; }  
}

**AuthController.java**package com.example.jwtauth.controller;  
  
import com.example.jwtauth.service.JwtService;  
import jakarta.servlet.http.HttpServletRequest;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.security.authentication.AuthenticationManager;  
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.Base64;  
import java.util.Map;  
  
@RestController  
public class AuthController {  
  
 @Autowired  
 private AuthenticationManager authManager;  
  
 @Autowired  
 private JwtService jwtService;  
  
 @PostMapping("/authenticate")  
 public Map<String, String> authenticate(HttpServletRequest request) {  
 String[] creds = extractBasicAuthCredentials(request);  
 UsernamePasswordAuthenticationToken authInputToken =  
 new UsernamePasswordAuthenticationToken(creds[0], creds[1]);  
  
 authManager.authenticate(authInputToken);  
 String token = jwtService.generateToken(creds[0]);  
  
 return Map.of("token", token);  
 }  
  
 private String[] extractBasicAuthCredentials(HttpServletRequest request)

{  
 String authHeader = request.getHeader("Authorization").substring("Basic ".length());  
 String decoded = new String(Base64.getDecoder().decode(authHeader));  
 return decoded.split(":", 2);  
 }  
}

**JwtService.java**package com.example.jwtauth.service;  
  
import io.jsonwebtoken.Jwts;  
import io.jsonwebtoken.SignatureAlgorithm;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.stereotype.Service;  
  
import java.util.Date;  
  
@Service  
public class JwtService {  
  
 @Value("${jwt.secret}")  
 private String secret;  
  
 public String generateToken(String username) {  
 return Jwts.builder()  
 .setSubject(username)  
 .setIssuedAt(new Date(System.currentTimeMillis()))  
 .setExpiration(new Date(System.currentTimeMillis() + 1000 \* 60 \* 60 \* 10))  
 .signWith(SignatureAlgorithm.HS256, secret)  
 .compact();  
 }  
}

**CustomerUserDetailService.java**package com.example.jwtauth.service;  
  
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 CustomUserDetailsService implements UserDetailsService {  
  
 @Override  
 public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  
 if ("user".equals(username)) {  
 return new User("user", "pwd", new ArrayList<>());  
 } else {  
 throw new UsernameNotFoundException("User not found");  
 }  
 }  
}

**JwtUtil.java**package com.example.jwtauth.util;  
  
import io.jsonwebtoken.Claims;  
import io.jsonwebtoken.Jwts;  
import io.jsonwebtoken.SignatureException;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.stereotype.Component;  
  
import java.util.Date;  
import java.util.function.Function;  
  
@Component  
public class JwtUtil {  
  
 @Value("${jwt.secret}")  
 private String secret;  
  
 public String extractUsername(String token) {  
 return extractClaim(token, Claims::getSubject);  
 }  
  
   
 public Date extractExpiration(String token) {  
 return extractClaim(token, Claims::getExpiration);  
 }

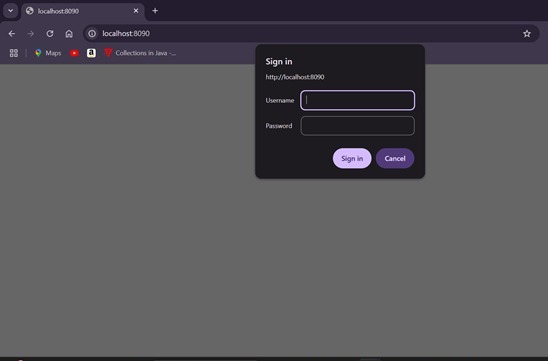
public <T> T extractClaim(String token, Function<Claims, T> claimsResolver) {  
 final Claims claims = extractAllClaims(token);  
 return claimsResolver.apply(claims);  
 }

public boolean isTokenExpired(String token) {  
 return extractExpiration(token).before(new Date());  
 }  
  
 public boolean validateToken(String token, String username) {  
 final String tokenUsername = extractUsername(token);  
 return (tokenUsername.equals(username) && !isTokenExpired(token));  
 }  
  
 private Claims extractAllClaims(String token) throws SignatureException

{  
 return Jwts.parser()  
 .setSigningKey(secret)  
 .parseClaimsJws(token)  
 .getBody();  
 }  
}

**SecurityConfig.java**  
package com.example.jwtauth.config;  
  
import com.example.jwtauth.service.CustomUserDetailsService;  
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.core.userdetails.User;  
import org.springframework.security.core.userdetails.UserDetails;  
import org.springframework.security.crypto.password.NoOpPasswordEncoder;  
import org.springframework.security.web.SecurityFilterChain;  
  
@Configuration  
@EnableWebSecurity  
public class SecurityConfig {  
  
 private final CustomUserDetailsService userDetailsService;  
  
 public SecurityConfig(CustomUserDetailsService userDetailsService) {  
 this.userDetailsService = userDetailsService;  
 }  
  
 @Bean  
 public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {  
 http.csrf().disable()  
 .authorizeHttpRequests().anyRequest().authenticated()  
 .and().httpBasic();  
 return http.build();  
 }  
  
 @Bean  
 public AuthenticationManager authManager(HttpSecurity http) throws Exception {  
 return http.getSharedObject(AuthenticationManagerBuilder.class)  
 .userDetailsService(userDetailsService)  
 .passwordEncoder(NoOpPasswordEncoder.getInstance())  
 .and().build();  
 }  
}

**OUTPUT:**

****