HelloController.java

package com.security;

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

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

@RestController

public class HelloController {

@GetMapping("/hello")

public String hello() {

return "hello";

}

}

JwtAuthenticationEntryPoint.java

package com.security;

import java.io.IOException;

import java.io.PrintWriter;

import java.io.Serializable;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import org.springframework.security.core.AuthenticationException;

import org.springframework.security.web.AuthenticationEntryPoint;

import org.springframework.stereotype.Component;

@Component

public class JwtAuthenticationEntryPoint implements AuthenticationEntryPoint {

@Override

public void commence(HttpServletRequest request, HttpServletResponse response, AuthenticationException authException) throws IOException, ServletException {

response.setStatus(HttpServletResponse.SC\_UNAUTHORIZED);

PrintWriter writer = response.getWriter();

writer.println("Access Denied !! " + authException.getMessage());

}

}

JwtController.java

package com.security;

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

import org.springframework.http.ResponseEntity;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.authentication.BadCredentialsException;

import org.springframework.security.authentication.DisabledException;

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

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

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

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

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

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

@RestController

@CrossOrigin

public class JwtController {

@Autowired

private JwtUserDetailsService userDetailsService;

@Autowired

private AuthenticationManager authenticationManager;

@Autowired

private TokenManager tokenManager;

@PostMapping("/login")

public ResponseEntity<> createToken(@RequestBody JwtRequestModel

request) throws Exception {

try {

authenticationManager.authenticate(

new

UsernamePasswordAuthenticationToken(request.getUsername(),

request.getPassword())

);

} catch (DisabledException e) {

throw new Exception("USER\_DISABLED", e);

} catch (BadCredentialsException e) {

throw new Exception("INVALID\_CREDENTIALS", e);

}

final UserDetails userDetails = userDetailsService.loadUserByUsername(request.getUsername());

final String jwtToken = tokenManager.generateJwtToken(userDetails);

return ResponseEntity.ok(new JwtResponseModel(jwtToken));

}

}

JwtFilter.java

package com.security;

import java.io.IOException;

import javax.servlet.FilterChain;

import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

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

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

import org.springframework.security.core.context.SecurityContextHolder;

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

import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;

import org.springframework.stereotype.Component;

import org.springframework.web.filter.OncePerRequestFilter;

import io.jsonwebtoken.ExpiredJwtException;

@Component

public class JwtFilter extends OncePerRequestFilter {

@Autowired

private JwtUserDetailsService userDetailsService;

@Autowired

private TokenManager tokenManager;

@Override

protected void doFilterInternal(HttpServletRequest request,

HttpServletResponse response, FilterChain filterChain)

throws ServletException, IOException {

String tokenHeader = request.getHeader("Authorization");

String username = null;

String token = null;

if (tokenHeader != null && tokenHeader.startsWith("Bearer ")) {

token = tokenHeader.substring(7);

try {

username = tokenManager.getUsernameFromToken(token);

} catch (IllegalArgumentException e) {

System.out.println("Unable to get JWT Token");

} catch (ExpiredJwtException e) {

System.out.println("JWT Token has expired");

}

} else {

System.out.println("Bearer String not found in token");

}

if (null != username &&SecurityContextHolder.getContext().getAuthentication() == null) {

UserDetails userDetails = userDetailsService.loadUserByUsername(username);

if (tokenManager.validateJwtToken(token, userDetails)) {

UsernamePasswordAuthenticationToken

authenticationToken = new UsernamePasswordAuthenticationToken(

userDetails, null,

userDetails.getAuthorities());

authenticationToken.setDetails(new

WebAuthenticationDetailsSource().buildDetails(request));

SecurityContextHolder.getContext().setAuthentication(authenticationToken);

}

}

filterChain.doFilter(request, response);

}

}

JwtHelper.java

package com.security;

import java.util.Date;

import java.util.HashMap;

import java.util.Map;

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

import org.springframework.stereotype.Component;

import io.jsonwebtoken.Claims;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

@Component

public class JwtHelper {

//requirement :

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

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

private String secret = "afafasfafafasfasfasfafacasdasfasxASFACASDFACASDFASFASFDAFASFASDAADSCSDFADCVSGCFVADXCcadwavfsfarvf";

//retrieve username from jwt token // JWS ( json web signature)

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 retrieveing any information from token we will need the secret key

private Claims getAllClaimsFromToken(String token) {

return Jwts.parser().setSigningKey(secret).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(UserDetails userDetails) {

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

return doGenerateToken(claims, userDetails.getUsername());

}

//while creating the token -

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

//2. Sign the JWT using the HS512 algorithm and 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(Map<String, Object> 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.HS512, secret).compact();

}

//validate token

public Boolean validateToken(String token, UserDetails userDetails) {

final String username = getUsernameFromToken(token);

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

}

}

JwtRequestModel.java

**package** com.security;

**import** java.io.Serializable;

**public** **class** JwtRequestModel **implements** Serializable {

/\*\*

\*

\*/

**private** **static** **final** **long** ***serialVersionUID*** = 2636936156391265891L;

**private** String username;

**private** String password;

**public** JwtRequestModel() {

}

**public** JwtRequestModel(String username, String password) {

**super**();

**this**.username = username; **this**.password = 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;

}

}

JwtResponseModel.java

**package** com.security;

**import** java.io.Serializable;

**public** **class** JwtResponseModel **implements** Serializable {

/\*\*

\*

\*/

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**private** **final** String token;

**public** JwtResponseModel(String token) {

**this**.token = token;

}

**public** String getToken() {

**return** token;

}

}

JwtUserDetailsService.java

package com.security;

import java.util.ArrayList;

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;

@Service

public class JwtUserDetailsService implements UserDetailsService {

@Override

public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

if ("randomuser123".equals(username)) {

return new User("randomuser123",

"$2a$10$slYQmyNdGzTn7ZLBXBChFOC9f6kFjAqPhccnP6DxlWXx2lPk1C3G6",

new ArrayList<>());

} else {

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

}

}

}

TokenManager.java

package com.security;

import java.io.Serializable;

import java.util.Base64;

import java.util.Date;

import java.util.HashMap;

import java.util.Map;

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

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

import org.springframework.stereotype.Component;

import io.jsonwebtoken.Claims; import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

@Component

public class TokenManager implements Serializable {

/\*\*

\*

\*/

private static final long serialVersionUID = 7008375124389347049L;

public static final long TOKEN\_VALIDITY = 10 \* 60 \* 60;

@Value("${secret}")

private String jwtSecret;

public String generateJwtToken(UserDetails userDetails) {

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

return Jwts.builder().setClaims(claims)

.setSubject(userDetails.getUsername())

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

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

.signWith(SignatureAlgorithm.HS512, jwtSecret).compact();

}

public Boolean validateJwtToken(String token, UserDetails userDetails) {

String username = getUsernameFromToken(token);

Claims claims = Jwts.parser().setSigningKey(jwtSecret).parseClaimsJws(token).getBody();

Boolean isTokenExpired = claims.getExpiration().before(new Date());

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

}

public String getUsernameFromToken(String token) {

final Claims claims = Jwts.parser().setSigningKey(jwtSecret).parseClaimsJws(token).getBody();

return claims.getSubject();

}

}

WebSecurityConfig.java

package com.security;

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.annotation.web.configuration.WebSecurityConfigurerAdapter;

import org.springframework.security.config.http.SessionCreationPolicy;

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.web.authentication.UsernamePasswordAuthenticationFilter;

import com.spring.security.jwtbasic.jwtutils.JwtAuthenticationEntryPoint;

import com.spring.security.jwtbasic.jwtutils.JwtFilter;

@Configuration

@EnableWebSecurity

public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

@Autowired

private JwtAuthenticationEntryPoint authenticationEntryPoint;

@Autowired

private UserDetailsService userDetailsService;

@Autowired

private JwtFilter filter;

@Bean

public PasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

@Override

protected void configure(AuthenticationManagerBuilder auth) throws Exception {

auth.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder());

}

@Bean

@Override

public AuthenticationManager authenticationManagerBean() throws

Exception {

return super.authenticationManagerBean();

}

@Override

protected void configure(HttpSecurity http) throws Exception {

http

.authorizeRequests().antMatchers("/login").permitAll()

.anyRequest().authenticated()

.and()

.exceptionHandling().authenticationEntryPoint(authenticationEntryPoint)

.and()

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);

// http.addFilterBefore(filter, UsernamePasswordAuthenticationFilter.class);

}

}

JwtTokenUtils.java

package com.security.config;

import java.io.Serializable;

import java.util.Date;

import java.util.HashMap;

import java.util.Map;

import java.util.function.Function;

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

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

import org.springframework.stereotype.Component;

import io.jsonwebtoken.Claims;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

@Component

public class JwtTokenUtil implements Serializable {

private static final long serialVersionUID = -2550185165626007488L;

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

@Value("${jwt.secret}")

private String secret;

//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 retrieveing any information from token we will need the secret key

private Claims getAllClaimsFromToken(String token) {

return Jwts.parser().setSigningKey(secret).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(UserDetails userDetails) {

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

return doGenerateToken(claims, userDetails.getUsername());

}

//while creating the token -

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

//2. Sign the JWT using the HS512 algorithm and 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(Map<String, Object> 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.HS512, secret).compact();

}

//validate token

public Boolean validateToken(String token, UserDetails userDetails) {

final String username = getUsernameFromToken(token);

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

}

}

WebSecurityConfig.java

package com.security.config;

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.WebSecurityConfigurerAdapter;

import org.springframework.security.config.http.SessionCreationPolicy;

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.web.authentication.UsernamePasswordAuthenticationFilter;

@Configuration

public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

@Bean

public PasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

}