JWT-Handson

Creating users and roles in Spring Security

SecurityConflg.java

package com.cognizant.springlearn.security;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration; import

org.springframework.security.config.annotation.authentication.builders.AuthenticationMana gerBuilder;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import

org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAd apter;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder; import org.springframework.security.crypto.password.PasswordEncoder;

@Configuration

public class SecurityConfig extends WebSecurityConfigurerAdapter {

private static final Logger LOGGER = LoggerFactory.getLogger(SecurityConfig.class);

// Define in-memory users and roles

@Override

protected void configure(AuthenticationManagerBuilder auth) throws Exception { auth.inMemoryAuthentication()

.withUser("admin").password(passwordEncoder().encode("pwd")).roles("ADMIN")

.and()

.withUser("user").password(passwordEncoder().encode("pwd")).roles("USER");

}

// Define password encoder bean @Bean

public PasswordEncoder passwordEncoder() { LOGGER.info("Start");

return new BCryptPasswordEncoder();

}

// Define access restrictions based on role @Override

protected void configure(HttpSecurity httpSecurity) throws Exception { httpSecurity.csrf().disable().httpBasic()

.and()

.authorizeRequests()

.antMatchers("/countries").hasRole("USER") // only users with "USER" role can access

.anyRequest().authenticated();

}

}

# Understanding JWT

What is JWT?

JWT (JSON Web Token) is a compact, URL-safe token format used to represent claims securely between two parties.

## ✅ Used for:

 Authentication: Proving who you are.

 Authorization: Verifying what you can access.

 Secure information exchange: Trustable because it’s signed.

# 📦 JWT Structure

A JWT has three parts, separated by dots (.): css

Copy code

HEADER.PAYLOAD.SIGNATURE

Each part is Base64Url encoded.

## Header

Contains metadata, including:

 Type: always JWT

 Algorithm: used for signing (e.g., HS256)

Example:

json

Copy code

{

"alg": "HS256",

"typ": "JWT"

}

## Payload

Contains the claims — the actual data.

Common claims include:

 sub (subject, often user ID)  name

 iat (issued at)

 exp (expiry time)

 roles (e.g., ADMIN, USER)

Example:

json

Copy code

{

"sub": "1234567890",

"name": "John Doe", "admin": true

}

⚠ Note: Payload is not encrypted, just encoded — don't put sensitive info.

## Signature

Used to verify the message wasn’t altered.

It’s generated using the header, payload, a secret key, and the algorithm (like HMAC-SHA256).

Signature formula:

scss

Copy code

HMACSHA256(

base64UrlEncode(header) + "." + base64UrlEncode(payload), secret

)

# 🧪 Hands-on Exercise with JWT

## 🔗 Step-by-step demo:

1. Open Wikipedia's JWT page:

👉 <https://en.wikipedia.org/wiki/JSON_Web_Token#Structure>

1. Open the JWT Debugger:

👉 [https://jwt.io](https://jwt.io/)

1. In jwt.io, go to the "Decoded" section and do the following:  Paste the header from Wikipedia:

 json

 Copy code  {

 "alg": "HS256",

 "typ": "JWT"  }

 Paste the payload from Wikipedia:  json

 Copy code  {

 "sub": "1234567890",

 "name": "John Doe",  "admin": true

 }

1. Type the secret key in "Verify Signature":
2. nginx
3. Copy code
4. secretkey
5. ✅ Now check if the Encoded JWT matches the token shown in Wikipedia.

Expected token:

Copy code

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.

eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gRG9lIiwiYWRtaW4iOnRydWV9. TJVA95OrM7E2cBab30RMHrHDcEfxjoYZgeFONFh7HgQ

# 🔁 JWT Authentication Flow

plaintext Copy code

1. Client → Server: Sends username + password
2. Server validates → Responds with JWT token
3. Client stores token (e.g., in localStorage/cookie)
4. Client → Server: Sends requests with JWT in Authorization header Authorization: Bearer <JWT>
5. Server validates token → Grants or denies access

Create authentication service that returns JWT

Create JwtUtil Utility Class

@Component

public class JwtUtil {

private static final String SECRET\_KEY = "secretkey";

public String generateToken(String username) {

return Jwts.builder()

.setSubject(username)

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

.setExpiration(new Date(System.currentTimeMillis() + 1000 \* 60 \* 10)) // 10 minutes

.signWith(SignatureAlgorithm.HS25 6, SECRET\_KEY)

.compact();

}

}

AuthenticationController

@RestController public class

AuthenticationController {

@Autowired

private AuthenticationManager authenticationManager;

@Autowired

private JwtUtil jwtUtil;

@GetMapping("/authenticate") public

ResponseEntity<Map<String,

String>> generateToken(HttpServletReques t request) {

String header = request.getHeader("Authorization"

);

if (header == null ||

!header.startsWith("Basic ")) {

throw new RuntimeException("Missing or invalid Authorization header");

}

// Decode Basic Auth

String[] credentials = new String(Base64.getDecoder().decod e(header.substring(6))).split(":");

String username = credentials[0];

String password = credentials[1];

// Authenticate user

authenticationManager.authentica te(new UsernamePasswordAuthentication Token(username, password));

// Generate token

String token = jwtUtil.generateToken(username);

return ResponseEntity.ok(Collections.sing letonMap("token", token));

}

}

SecurityConflg.java

@Configuration

@EnableWebSecurity

public class SecurityConfig extends WebSecurityConfigurerAdapter {

private static final Logger LOGGER = LoggerFactory.getLogger(SecurityConfig.class);

@Override

protected void configure(AuthenticationManagerBuilder auth) throws Exception { auth.inMemoryAuthentication()

.withUser("admin").password(passwordEncoder().encode("pwd")).roles("ADMIN")

.and()

.withUser("user").password(passwordEncoder().encode("pwd")).roles("USER");

}

@Bean

public PasswordEncoder passwordEncoder() { LOGGER.info("Start password encoder");

return new BCryptPasswordEncoder();

}

@Override

protected void configure(HttpSecurity http) throws Exception { http.csrf().disable()

.authorizeRequests()

.antMatchers("/authenticate").permitAll()

.antMatchers("/countries").hasRole("USER")

.anyRequest().authenticated()

.and()

.httpBasic();

}

@Bean @Override

public AuthenticationManager authenticationManagerBean() throws Exception { return super.authenticationManagerBean();

}

}

Create authentication controller and conflgure it in SecurityConflg

AuthenticationController.java

package com.example.demo.controller;

import java.util.HashMap; import java.util.Map;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

@RestController

public class AuthenticationController {

private static final Logger LOGGER =

LoggerFactory.getLogger(AuthenticationController.class);

@GetMapping("/authenticate")

public Map<String, String> authenticate(@RequestHeader("Authorization") String authHeader) {

LOGGER.info("Start of /authenticate");

LOGGER.debug("Authorization Header: {}", authHeader);

Map<String, String> map = new HashMap<>(); map.put("token", "");

LOGGER.info("End of /authenticate"); return map;

}

}

SecurityConflg.java

@Override

protected void configure(HttpSecurity http) throws Exception { http.csrf().disable()

.authorizeRequests()

.antMatchers("/countries").hasRole("USER")

.antMatchers("/authenticate").hasAnyRole("USER", "ADMIN")

.anyRequest().authenticated()

.and()

.httpBasic();

}

Create authentication controller and conflgure it in SecurityConflg

AuthenticationController.java

package com.example.controller;

import java.util.HashMap; import java.util.Map;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

@RestController

public class AuthenticationController {

private static final Logger LOGGER =

LoggerFactory.getLogger(AuthenticationController.class);

@GetMapping("/authenticate")

public Map<String, String> authenticate(@RequestHeader("Authorization") String authHeader) {

LOGGER.info("Start of /authenticate");

LOGGER.debug("Authorization Header: {}", authHeader);

Map<String, String> map = new HashMap<>();

map.put("token", ""); // Empty token as per the current requirement

LOGGER.info("End of /authenticate"); return map;

}

}

SecurityConflg.java

import org.springframework.context.annotation.Bean; import

org.springframework.security.config.annotation.authentication.builders.AuthenticationMana gerBuilder;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import

org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAd apter;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder; import org.springframework.security.crypto.password.PasswordEncoder;

public class SecurityConfig extends WebSecurityConfigurerAdapter {

@Override

protected void configure(AuthenticationManagerBuilder auth) throws Exception { auth.inMemoryAuthentication()

.withUser("admin").password(passwordEncoder().encode("pwd")).roles("ADMIN")

.and()

.withUser("user").password(passwordEncoder().encode("pwd")).roles("USER");

}

@Bean

public PasswordEncoder passwordEncoder() { return new BCryptPasswordEncoder();

}

@Override

protected void configure(HttpSecurity httpSecurity) throws Exception { httpSecurity.csrf().disable().httpBasic().and()

.authorizeRequests()

.antMatchers("/countries").hasRole("USER")

.antMatchers("/authenticate").hasAnyRole("USER", "ADMIN")

.anyRequest().authenticated();

}

}

# Read Authorization header and decode the username and password

AuthenticationController.java

package com.example.controller;

import java.util.Base64; import java.util.HashMap; import java.util.Map;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

@RestController

public class AuthenticationController {

private static final Logger LOGGER = LoggerFactory.getLogger(AuthenticationController.class);

@GetMapping("/authenticate")

public Map<String, String> authenticate(@RequestHeader("Authorization") String authHeader)

{

LOGGER.info("Start of /authenticate"); LOGGER.debug("Authorization Header: {}", authHeader);

String username = getUser(authHeader); LOGGER.debug("Decoded Username: {}", username);

Map<String, String> map = new HashMap<>(); map.put("token", ""); // Placeholder token LOGGER.info("End of /authenticate");

return map;

}

private String getUser(String authHeader) { LOGGER.debug("Start of getUser()");

// Strip "Basic " prefix

String encodedCredentials = authHeader.substring("Basic ".length());

LOGGER.debug("Encoded credentials: {}", encodedCredentials);

// Decode using Base64

byte[] decodedBytes = Base64.getDecoder().decode(encodedCredentials);

String decodedCredentials = new String(decodedBytes); LOGGER.debug("Decoded credentials: {}", decodedCredentials);

// Extract username (before colon)

String username = decodedCredentials.split(":", 2)[0]; LOGGER.debug("Extracted username: {}", username);

LOGGER.debug("End of getUser()"); return username;

}

}

# Generate token based on the user

AuthenticationController.java

package com.example.controller;

import java.util.Base64; import java.util.Date; import java.util.HashMap; import java.util.Map;

import io.jsonwebtoken.JwtBuilder; import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

@RestController

public class AuthenticationController {

private static final Logger LOGGER =

LoggerFactory.getLogger(AuthenticationController.class);

@GetMapping("/authenticate")

public Map<String, String> authenticate(@RequestHeader("Authorization") String authHeader) {

LOGGER.info("Start of /authenticate");

LOGGER.debug("Authorization Header: {}", authHeader);

String username = getUser(authHeader);

LOGGER.debug("Decoded Username: {}", username);

String token = generateJwt(username); LOGGER.debug("Generated JWT: {}", token);

Map<String, String> map = new HashMap<>(); map.put("token", token);

LOGGER.info("End of /authenticate"); return map;

}

private String getUser(String authHeader) { LOGGER.debug("Start of getUser()");

String encodedCredentials = authHeader.substring("Basic ".length()); LOGGER.debug("Encoded credentials: {}", encodedCredentials);

byte[] decodedBytes = Base64.getDecoder().decode(encodedCredentials); String decodedCredentials = new String(decodedBytes);

LOGGER.debug("Decoded credentials: {}", decodedCredentials);

String username = decodedCredentials.split(":", 2)[0]; LOGGER.debug("Extracted username: {}", username);

LOGGER.debug("End of getUser()"); return username;

}

private String generateJwt(String user) {

LOGGER.debug("Start of generateJwt() for user: {}", user);

JwtBuilder builder = Jwts.builder(); builder.setSubject(user);

builder.setIssuedAt(new Date());

builder.setExpiration(new Date((new Date()).getTime() + 1200000)); // 20 minutes builder.signWith(SignatureAlgorithm.HS256, "secretkey");

String token = builder.compact();

LOGGER.debug("JWT Token generated: {}", token);

return token;

}

}

# Authorize based on JWT

JwtAuthorizationFilter.java

package com.cognizant.springlearn.security;

import java.io.IOException; import java.util.ArrayList;

import javax.servlet.FilterChain; import javax.servlet.ServletException;

import javax.servlet.http.HttpServletRequest; import javax.servlet.http.HttpServletResponse;

import io.jsonwebtoken.Claims; import io.jsonwebtoken.Jws;

import io.jsonwebtoken.JwtException; import io.jsonwebtoken.Jwts;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.security.authentication.AuthenticationManager;

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

import org.springframework.security.web.authentication.www.BasicAuthenticationFilter;

public class JwtAuthorizationFilter extends BasicAuthenticationFilter {

private static final Logger LOGGER = LoggerFactory.getLogger(JwtAuthorizationFilter.class);

public JwtAuthorizationFilter(AuthenticationManager authenticationManager) { super(authenticationManager);

LOGGER.info("JwtAuthorizationFilter Constructor - Start");

LOGGER.debug("AuthenticationManager: {}", authenticationManager);

}

@Override

protected void doFilterInternal(HttpServletRequest req, HttpServletResponse res, FilterChain chain) throws IOException, ServletException {

LOGGER.info("Start of doFilterInternal()"); String header = req.getHeader("Authorization");

LOGGER.debug("Authorization Header: {}", header);

if (header == null || !header.startsWith("Bearer ")) { chain.doFilter(req, res);

return;

}

UsernamePasswordAuthenticationToken authentication = getAuthentication(req);

SecurityContextHolder.getContext().setAuthentication(authentication); chain.doFilter(req, res);

LOGGER.info("End of doFilterInternal()");

}

private UsernamePasswordAuthenticationToken getAuthentication(HttpServletRequest request) {

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

if (token != null) { try {

Jws<Claims> jws = Jwts.parser()

.setSigningKey("secretkey")

.parseClaimsJws(token.replace("Bearer ", "")); String user = jws.getBody().getSubject();

LOGGER.debug("Decoded user from token: {}", user); if (user != null) {

return new UsernamePasswordAuthenticationToken(user, null, new ArrayList<>());

}

} catch (JwtException ex) {

LOGGER.error("Invalid JWT token", ex); return null;

}

}

return null;

}

}

SecurityConflg.java

@Override

protected void configure(HttpSecurity httpSecurity) throws Exception { httpSecurity.csrf().disable()

.httpBasic().and()

.authorizeRequests()

.antMatchers("/authenticate").hasAnyRole("USER", "ADMIN")

.anyRequest().authenticated()

.and()

.addFilter(new JwtAuthorizationFilter(authenticationManager()));

}