# 📘 Spring Boot Security with JWT – Complete Guide

## 1. Project Structure

Here’s the basic structure of your Spring Boot application for JWT authentication:

src

└── main

├── java

│ └── com

│ └── example

│ └── jwtsecurity

│ ├── config

│ │ └── SecurityConfig.java # Spring Security Configuration

│ ├── controller

│ │ ├── AuthController.java # Login & Registration APIs

│ │ └── TestController.java # Sample protected endpoints

│ ├── dto

│ │ ├── AuthRequest.java # DTO for login/register request

│ │ └── AuthResponse.java # DTO for JWT token response

│ ├── model

│ │ └── User.java # User entity

│ ├── repository

│ │ └── UserRepository.java # JPA Repository for User

│ ├── security

│ │ └── JwtFilter.java # JWT filter to intercept and validate requests

│ └── service

│ ├── JwtService.java # Service to generate/validate JWT tokens

│ └── UserDetailsServiceImpl.java # Loads user from DB for Spring Security

└── resources

└── application.properties # App configuration (DB, security, etc.)

## 2. Code Implementation

### 2.1 User Entity

package com.example.jwtsecurity.model;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import lombok.AllArgsConstructor;

import lombok.Builder;

import lombok.Data;

import lombok.NoArgsConstructor;

@Entity

@Data

@NoArgsConstructor

@AllArgsConstructor

@Builder

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String username;

private String password;

private String role; // e.g., ROLE\_USER or ROLE\_ADMIN

}

**2.2 User Repository**

package com.example.jwtsecurity.repository;

import com.example.jwtsecurity.model.User;

import org.springframework.data.jpa.repository.JpaRepository;

import java.util.Optional;

public interface UserRepository extends JpaRepository<User, Long> {

Optional<User> findByUsername(String username);

}

### 2.3 DTOs for Authentication

#### AuthRequest.java

package com.example.jwtsecurity.dto;

import lombok.Data;

@Data

public class AuthRequest {

private String username;

private String password;

}

#### AuthResponse.java

package com.example.jwtsecurity.dto;

import lombok.AllArgsConstructor;

import lombok.Data;

@Data

@AllArgsConstructor

public class AuthResponse {

private String token;

}

#### 2.4 JWT Service

package com.example.jwtsecurity.service;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

import org.springframework.stereotype.Service;

import java.util.Date;

@Service

public class JwtService {

private final String secretKey = "mySecretKey12345";

private final long expirationMs = 3600000; // 1 hour

public String generateToken(String username, String role) {

return Jwts.builder()

.setSubject(username)

.claim("role", role)

.setIssuedAt(new Date())

.setExpiration(new Date(System.currentTimeMillis() + expirationMs))

.signWith(SignatureAlgorithm.HS256, secretKey)

.compact();

}

public String getUsername(String token) {

return Jwts.parser().setSigningKey(secretKey)

.parseClaimsJws(token)

.getBody().getSubject();

}

public boolean isTokenValid(String token) {

try {

Jwts.parser().setSigningKey(secretKey).parseClaimsJws(token);

return true;

} catch (Exception e) {

return false;

}

}

}

### 2.5 AuthController (Login & Registration)

package com.example.jwtsecurity.controller;

import com.example.jwtsecurity.dto.AuthRequest;

import com.example.jwtsecurity.dto.AuthResponse;

import com.example.jwtsecurity.model.User;

import com.example.jwtsecurity.repository.UserRepository;

import com.example.jwtsecurity.service.JwtService;

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

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

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

import org.springframework.web.bind.annotation.\*;

@RestController

@RequestMapping("/auth")

public class AuthController {

@Autowired

private UserRepository userRepo;

@Autowired

private JwtService jwtService;

@Autowired

private PasswordEncoder passwordEncoder;

@PostMapping("/login")

public ResponseEntity<AuthResponse> login(@RequestBody AuthRequest request) {

User user = userRepo.findByUsername(request.getUsername())

.orElseThrow(() -> new RuntimeException("User not found"));

if (!passwordEncoder.matches(request.getPassword(), user.getPassword())) {

return ResponseEntity.badRequest().body(new AuthResponse("Invalid credentials"));

}

String token = jwtService.generateToken(user.getUsername(), user.getRole());

return ResponseEntity.ok(new AuthResponse(token));

}

@PostMapping("/register")

public ResponseEntity<String> register(@RequestBody AuthRequest request) {

if (userRepo.findByUsername(request.getUsername()).isPresent()) {

return ResponseEntity.badRequest().body("Username already taken.");

}

String encodedPassword = passwordEncoder.encode(request.getPassword());

User user = new User(null, request.getUsername(), encodedPassword, "ROLE\_USER");

userRepo.save(user);

return ResponseEntity.ok("User registered successfully!");

}

}

### 2.6 JWT Filter

package com.example.jwtsecurity.security;

import com.example.jwtsecurity.service.JwtService;

import com.example.jwtsecurity.service.UserDetailsServiceImpl;

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

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

import org.springframework.stereotype.Component;

import org.springframework.web.filter.OncePerRequestFilter;

import javax.servlet.Filter;

import javax.servlet.FilterChain;

import javax.servlet.FilterConfig;

import javax.servlet.ServletException;

import javax.servlet.ServletRequest;

import javax.servlet.ServletResponse;

import javax.servlet.annotation.WebFilter;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import java.io.IOException;

@Component

public class JwtFilter extends OncePerRequestFilter {

private final JwtService jwtService;

private final UserDetailsServiceImpl userDetailsService;

public JwtFilter(JwtService jwtService, UserDetailsServiceImpl userDetailsService) {

this.jwtService = jwtService;

this.userDetailsService = userDetailsService;

}

@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 jwt = authHeader.substring(7);

if (jwtService.isTokenValid(jwt)) {

String username = jwtService.getUsername(jwt);

var userDetails = userDetailsService.loadUserByUsername(username);

var authToken = new UsernamePasswordAuthenticationToken(userDetails, null, userDetails.getAuthorities());

authToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));

SecurityContextHolder.getContext().setAuthentication(authToken);

}

}

filterChain.doFilter(request, response);

}

}

### 2.7 UserDetailsService Implementation

package com.example.jwtsecurity.service;

import com.example.jwtsecurity.model.User;

import com.example.jwtsecurity.repository.UserRepository;

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

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

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

import org.springframework.stereotype.Service;

@Service

public class UserDetailsServiceImpl implements UserDetailsService {

private final UserRepository userRepo;

public UserDetailsServiceImpl(UserRepository userRepo) {

this.userRepo = userRepo;

}

@Override

public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

User user = userRepo.findByUsername(username)

.orElseThrow(() -> new UsernameNotFoundException("User not found"));

return new org.springframework.security.core.userdetails.User(

user.getUsername(),

user.getPassword(),

List.of(new SimpleGrantedAuthority(user.getRole()))

);

}

}

### 2.8 Spring Security Configuration

package com.example.jwtsecurity.config;

import com.example.jwtsecurity.security.JwtFilter;

import com.example.jwtsecurity.service.UserDetailsServiceImpl;

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.core.userdetails.UserDetailsService;

@Configuration

@EnableWebSecurity

public class SecurityConfig extends WebSecurityConfigurerAdapter {

private final JwtFilter jwtFilter;

private final UserDetailsService userDetailsService;

public SecurityConfig(JwtFilter jwtFilter, UserDetailsService userDetailsService) {

this.jwtFilter = jwtFilter;

this.userDetailsService = userDetailsService;

}

@Override

protected void configure(HttpSecurity http) throws Exception {

http.csrf().disable()

.authorizeRequests()

.antMatchers("/auth/login", "/auth/register").permitAll()

.antMatchers("/api/user").hasAnyRole("USER", "ADMIN")

.antMatchers("/api/admin").hasRole("ADMIN")

.anyRequest().authenticated()

.and()

.addFilterBefore(jwtFilter, UsernamePasswordAuthenticationFilter.class)

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

}

@Override

protected void configure(AuthenticationManagerBuilder auth) throws Exception {

auth.userDetailsService(userDetailsService);

}

@Bean

@Override

public AuthenticationManager authenticationManagerBean() throws Exception {

return super.authenticationManagerBean();

}

}

### 2.9 Sample API Controller with Role-Based Access

package com.example.jwtsecurity.controller;

import org.springframework.web.bind.annotation.\*;

@RestController

@RequestMapping("/api")

public class TestController {

@GetMapping("/public")

public String publicEndpoint() {

return "This is a public endpoint.";

}

@GetMapping("/user")

public String userEndpoint() {

return "Hello USER or ADMIN!";

}

@GetMapping("/admin")

public String adminEndpoint() {

return "Hello ADMIN!";

}

}

# Spring Boot JWT Authentication - API Documentation

## 1. User Registration API

Endpoint: POST /auth/register

Description: Registers a new user with username and password.

### Request Body Example:

{  
 "username": "john",  
 "password": "password123"  
}

### Response Example:

"User registered successfully!"

## 2. User Login API

Endpoint: POST /auth/login

Description: Authenticates user and returns JWT token.

### Request Body Example:

{  
 "username": "john",  
 "password": "password123"  
}

### Response Example:

{  
 "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6..."  
}

## 3. Auth Protected API - User

Endpoint: GET /api/user

Description: Accessible by users with role USER or ADMIN. Requires JWT token in header.

### Authorization Header Example:

Authorization: Bearer <JWT\_TOKEN>

### Response Example:

"Hello USER or ADMIN!"

## 4. Auth Protected API - Admin

Endpoint: GET /api/admin

Description: Accessible only by users with role ADMIN. Requires JWT token in header.

### Authorization Header Example:

Authorization: Bearer <JWT\_TOKEN>

### Response Example:

"Hello ADMIN!"