**Create authentication service that returns JWT**   
  
As part of the first step of the JWT process, the user credentials need to be sent to an authentication service request that generates and returns the JWT.  
  
Ideally when the below curl command is executed that calls the new authentication service, the token should be responded. Kindly note that the credentials are passed using -u option.  
  
**Request**

curl -s -u user:pwd http://localhost:8090/authenticate

**Response**

{"token":"eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJ1c2VyIiwiaWF0IjoxNTcwMzc5NDc0LCJleHAiOjE1NzAzODA2NzR9.t3LRvlCV-hwKfoqZYlaVQqEUiBloWcWn0ft3tgv0dL0"}

This can be incorporated as three major steps:

* · Create authentication controller and configure it in SecurityConfig
* · Read Authorization header and decode the username and password
* · Generate token based on the user retrieved in the previous step

Let's incorporate the above as separate hands on exercises.

### **1. Initial Project Setup**

* Open spring-learn in Eclipse
* Add **Spring Security** to your pom.xml:-

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt</artifactId>

<version>0.9.1</version>

</dependency>

**2. Create SecurityConfig class:-**

@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");

}

@Override

protected void configure(HttpSecurity http) throws Exception {

http.csrf().disable()

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

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

.anyRequest().authenticated()

.and()

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

http.addFilterBefore(jwtRequestFilter, UsernamePasswordAuthenticationFilter.class);

}

@Bean

public PasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

@Autowired

private JwtRequestFilter jwtRequestFilter;

@Bean

@Override

public AuthenticationManager authenticationManagerBean() throws Exception {

return super.authenticationManagerBean();

}

}

3. **Create Models for JWT Authentication:-**

#### **JwtRequest.java**

public class JwtRequest {

private String username;

private String password;

// getters and setters

}

#### **JwtResponse.java**

public class JwtResponse {

private String token;

public JwtResponse(String token) {

this.token = token;

}

// getter

}

4. **JWT Utility Class:-**

@Component

public class JwtUtil {

private String SECRET\_KEY = "secretkey";

public String generateToken(UserDetails userDetails) {

return Jwts.builder()

.setSubject(userDetails.getUsername())

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

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

.signWith(SignatureAlgorithm.HS256, SECRET\_KEY)

.compact();

}

public String extractUsername(String token) {

return Jwts.parser().setSigningKey(SECRET\_KEY).parseClaimsJws(token).getBody().getSubject();

}

public Boolean validateToken(String token, UserDetails userDetails) {

final String username = extractUsername(token);

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

}

private Boolean isTokenExpired(String token) {

final Date expiration = Jwts.parser().setSigningKey(SECRET\_KEY).parseClaimsJws(token).getBody().getExpiration();

return expiration.before(new Date());

}

}

5. **JWT Filter:-**

@Component

public class JwtRequestFilter extends OncePerRequestFilter {

@Autowired

private MyUserDetailsService userDetailsService;

@Autowired

private JwtUtil jwtUtil;

@Override

protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain chain)

throws ServletException, IOException {

final String authorizationHeader = request.getHeader("Authorization");

String username = null;

String jwt = null;

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

jwt = authorizationHeader.substring(7);

username = jwtUtil.extractUsername(jwt);

}

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

UserDetails userDetails = this.userDetailsService.loadUserByUsername(username);

if (jwtUtil.validateToken(jwt, userDetails)) {

UsernamePasswordAuthenticationToken authToken =

new UsernamePasswordAuthenticationToken(userDetails, null, userDetails.getAuthorities());

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

SecurityContextHolder.getContext().setAuthentication(authToken);

}

}

chain.doFilter(request, response);

}

}

**6. Custom UserDetailsService:-**

@Service

public class MyUserDetailsService implements UserDetailsService {

@Override

public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

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

return new User("user", new BCryptPasswordEncoder().encode("pwd"), Collections.singleton(new SimpleGrantedAuthority("ROLE\_USER")));

} else if ("admin".equals(username)) {

return new User("admin", new BCryptPasswordEncoder().encode("pwd"), Collections.singleton(new SimpleGrantedAuthority("ROLE\_ADMIN")));

} else {

throw new UsernameNotFoundException("User not found");

}

}

}

7. **Authentication Controller:-**

@RestController

public class AuthenticationController {

@Autowired

private AuthenticationManager authenticationManager;

@Autowired

private MyUserDetailsService userDetailsService;

@Autowired

private JwtUtil jwtUtil;

@PostMapping("/authenticate")

public ResponseEntity<?> createAuthenticationToken(@RequestBody JwtRequest jwtRequest) throws Exception {

try {

authenticationManager.authenticate(

new UsernamePasswordAuthenticationToken(jwtRequest.getUsername(), jwtRequest.getPassword())

);

} catch (BadCredentialsException e) {

throw new Exception("Incorrect username or password", e);

}

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

final String jwt = jwtUtil.generateToken(userDetails);

return ResponseEntity.ok(new JwtResponse(jwt));

}

}

OUTPUTS:-





