Since you're using Spring Boot with MongoDB, I'll provide an example of how to implement JWT authentication and authorization in your project. Here's the code you need:

1. Add the required dependencies to your **pom.xml** file:

xml

EditFull ScreenCopy code

1<dependency>

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

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

4</dependency>

5<dependency>

6 <groupId>io.jsonwebtoken</groupId>

7 <artifactId>jjwt</artifactId>

8 <version>0.9.1</version>

9</dependency>

1. Create a **SecurityConfig** class that extends **WebSecurityConfigurerAdapter** to configure Spring Security:

java

EditFull ScreenCopy code

1import org.springframework.beans.factory.annotation.Autowired;

2import org.springframework.context.annotation.Bean;

3import org.springframework.context.annotation.Configuration;

4import org.springframework.security.authentication.AuthenticationManager;

5import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;

6import org.springframework.security.config.annotation.web.builders.HttpSecurity;

7import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

8import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

9import org.springframework.security.config.http.SessionCreationPolicy;

10import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

11import org.springframework.security.crypto.password.PasswordEncoder;

12import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

13

14@Configuration

15@EnableWebSecurity

16public class SecurityConfig extends WebSecurityConfigurerAdapter {

17

18 @Autowired

19 private MyUserDetailsService userDetailsService;

20

21 @Autowired

22 private JwtRequestFilter jwtRequestFilter;

23

24 @Override

25 protected void configure(AuthenticationManagerBuilder auth) throws Exception {

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

27 }

28

29 @Bean

30 public PasswordEncoder passwordEncoder() {

31 return new BCryptPasswordEncoder();

32 }

33

34 @Bean

35 @Override

36 public AuthenticationManager authenticationManagerBean() throws Exception {

37 return super.authenticationManagerBean();

38 }

39

40 @Override

41 protected void configure(HttpSecurity http) throws Exception {

42 http.csrf().disable()

43 .authorizeRequests().antMatchers("/auth/\*\*").permitAll()

44 .anyRequest().authenticated()

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

46

47 http.addFilterBefore(jwtRequestFilter, UsernamePasswordAuthenticationFilter.class);

48 }

49}

1. Create a **MyUserDetailsService** class that implements **UserDetailsService**:

java

EditFull ScreenCopy code

1import org.springframework.beans.factory.annotation.Autowired;

2import org.springframework.security.core.userdetails.UserDetails;

3import org.springframework.security.core.userdetails.UserDetailsService;

4import org.springframework.security.core.userdetails.UsernameNotFoundException;

5import org.springframework.stereotype.Service;

6

7@Service

8public class MyUserDetailsService implements UserDetailsService {

9

10 @Autowired

11 private UserRepository userRepository;

12

13 @Override

14 public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

15 User user = userRepository.findByUsername(username);

16 if (user == null) {

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

18 }

19 return new org.springframework.security.core.userdetails.User(user.getUsername(), user.getPassword(), new ArrayList<>());

20 }

21}

1. Create a **JwtRequestFilter** class that extends **OncePerRequestFilter** to handle JWT tokens:

java

EditFull ScreenCopy code

1import org.springframework.security.authentication.AuthenticationManager;

2import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

3import org.springframework.security.core.context.SecurityContextHolder;

4import org.springframework.security.core.userdetails.UserDetails;

5import org.springframework.security.core.userdetails.UserDetailsService;

6import org.springframework.security.web.authentication.www.BasicAuthenticationFilter;

7

8import javax.servlet.FilterChain;

9import javax.servlet.ServletException;

10import javax.servlet.http.HttpServletRequest;

11import javax.servlet.http.HttpServletResponse;

12import java.io.IOException;

13

14public class JwtRequestFilter extends BasicAuthenticationFilter {

15

16 private UserDetailsService userDetailsService;

17 private JwtTokenUtil jwtTokenUtil;

18

19 public JwtRequestFilter(AuthenticationManager authManager, UserDetailsService userDetailsService, JwtTokenUtil jwtTokenUtil) {

20 super(authManager);

21 this.userDetailsService = userDetailsService;

22 this.jwtTokenUtil = jwtTokenUtil;

23 }

24

25 @Override

26 protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain chain) throws IOException, ServletException {

27 String header = request.getHeader(JwtTokenUtil.HEADER\_STRING);

28 if (header == null || !header.startsWith(JwtTokenUtil.TOKEN\_PREFIX)) {

29 chain.doFilter(request, response);

30 return;

31 }

32 String token = header.replace(JwtTokenUtil.TOKEN\_PREFIX, "");

33 String username = jwtTokenUtil.getUsernameFromToken(token);

34

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

36 UserDetails userDetails = userDetailsService.loadUserByUsername(username);

37 if (jwtTokenUtil.validateToken(token, userDetails)) {

38 UsernamePasswordAuthenticationToken authentication = new UsernamePasswordAuthenticationToken(userDetails, null, userDetails.getAuthorities());

39 SecurityContextHolder.getContext().setAuthentication(authentication);

40 }

41 }

42 chain.doFilter(request, response);

43 }

44}

1. Create a **JwtTokenUtil** class that handles JWT tokens:

java

EditFull ScreenCopy code

1import io.jsonwebtoken.Claims;

2import io.jsonwebtoken.Jwts;

3import io.jsonwebtoken.SignatureAlgorithm;

4import org.springframework.security.core.userdetails.UserDetails;

5import org.springframework.stereotype.Component;

6

7import java.util.Date;

8import java.util.HashMap;

9import java.util.Map;

10import java.util.function.Function;

11

12@Component

13public class JwtTokenUtil {

14

15 public static final String SECRET = "your-secret-key";

16 public static final String HEADER\_STRING = "Authorization";

17 public static final String TOKEN\_PREFIX = "Bearer ";

18 public static final long EXPIRATION\_TIME = 86400000; // 1 day

19

20 public String getUsernameFromToken(String token) {

21 return getClaimFromToken(token, Claims::getSubject);

22 }

23

24 public Date getExpirationDateFromToken(String token) {

25 return getClaimFromToken(token, Claims::getExpiration);

26 }

27

28 public <T> T getClaimFromToken(String token, Function<Claims, T> claimsResolver) {

29 final Claims claims = getAllClaimsFromToken(token);

30 return claimsResolver.apply(claims);

31 }

32

33 private Claims getAllClaimsFromToken(String token) {

34 return Jwts.parser().setSigningKey(SECRET).parseClaimsJws(token).getBody();

35 }

36

37 public String generateToken(UserDetails userDetails) {

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

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

40 }

41

42 private String doGenerateToken(Map<String, Object> claims, String subject) {

43 return Jwts.builder().setClaims(claims).setSubject(subject).setIssuedAt(new Date(System.currentTimeMillis()))

44 .setExpiration(new Date(System.currentTimeMillis() + EXPIRATION\_TIME))

45 .signWith(SignatureAlgorithm.HS512, SECRET).compact();

46 }

47

48 public Boolean validateToken(String token, UserDetails userDetails) {

49 final String username = getUsernameFromToken(token);

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

51 }

52

53 private Boolean isTokenExpired(String token) {

54 final Date expiration = getExpirationDateFromToken(token);

55 return expiration.before(new Date());}

56}

1. Create a **User** entity that represents a participant:

java

EditFull ScreenCopy code

1import org.springframework.data.annotation.Id;

2import org.springframework.data.mongodb.core.mapping.Document;

3import org.springframework.security.core.GrantedAuthority;

4import org.springframework.security.core.authority.SimpleGrantedAuthority;

5import org.springframework.security.core.userdetails.UserDetails;

6

7import java.util.Collection;

8import java.util.List;

9

10@Document(collection = "users")

11public class User implements UserDetails {

12

13 @Id

14 private String id;

15 private String username;

16 private String password;

17 private List<String> roles;

18

19 public User(String id, String username, String password, List<String> roles) {

20 this.id = id;

21 this.username = username;

22 this.password = password;

23 this.roles = roles;

24 }

25

26 @Override

27 public Collection<? extends GrantedAuthority> getAuthorities() {

28 return roles.stream().map(SimpleGrantedAuthority::new).collect(Collectors.toList());

29 }

30

31 @Override

32 public String getPassword() {

33 return password;

34 }

35

36 @Override

37 public String getUsername() {

38 return username;

39 }

40

41 @Override

42 public boolean isAccountNonExpired() {

43 return true;

44 }

45

46 @Override

47 public boolean isAccountNonLocked() {

48 return true;

49 }

50

51 @Override

52 public boolean isCredentialsNonExpired() {

53 return true;

54 }

55

56 @Override

57 public boolean isEnabled() {

58 return true;

59 }

60

61 public String getId() {

62 return id;

63 }

64

65 public void setId(String id) {

66 this.id = id;

67 }

68

69 public void setUsername(String username) {

70 this.username = username;

71 }

72

73 public void setPassword(String password) {

74 this.password = password;

75 }

76

77 public void setRoles(List<String> roles) {

78 this.roles = roles;

79 }

80}

1. Create a **UserRepository** interface that extends **MongoRepository**:

java

EditFull ScreenCopy code

1import org.springframework.data.mongodb.repository.MongoRepository;

2import org.springframework.stereotype.Repository;

3

4@Repository

5public interface UserRepository extends MongoRepository<User, String> {

6

7 User findByUsername(String username);

8}

1. Create a **UserController** class that handles user authentication:

java

EditFull ScreenCopy code

1import org.springframework.beans.factory.annotation.Autowired;

2import org.springframework.http.ResponseEntity;

3import org.springframework.security.authentication.AuthenticationManager;

4import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

5import org.springframework.security.core.userdetails.UserDetails;

6import org.springframework.security.core.userdetails.UserDetailsService;

7import org.springframework.web.bind.annotation.PostMapping;

8import org.springframework.web.bind.annotation.RequestBody;

9import org.springframework.web.bind.annotation.RequestMapping;

10import org.springframework.web.bind.annotation.RestController;

11

12import javax.servlet.http.HttpServletRequest;

13

14@RestController

15@RequestMapping("/auth")

16public class UserController {

17

18 @Autowired

19 private AuthenticationManager authenticationManager;

20

21 @Autowired

22 private UserDetailsService userDetailsService;

23

24 @Autowired

25 private JwtTokenUtil jwtTokenUtil;

26

27 @PostMapping("/login")

28 public ResponseEntity<?> createAuthenticationToken(@RequestBody User user, HttpServletRequest request) {

29 authenticate(user.getUsername(), user.getPassword());

30 final UserDetails userDetails = userDetailsService.loadUserByUsername(user.getUsername());

31 String token = jwtTokenUtil.generateToken(userDetails);

32 return ResponseEntity.ok(new JwtResponse(token));

33 }

34

35 private void authenticate(String username, String password) {

36 try {

37 authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(username, password));

38 } catch (Exception e) {

39 throw new RuntimeException("Invalid username or password");

40 }

41 }

42}

1. Create a **JwtResponse** class that represents the response when a user logs in:

java

EditFull ScreenCopy code

1public class JwtResponse {

2

3 private String token;

4

5 public JwtResponse(String token) {

6 this.token = token;

7 }

8

9 public String getToken() {

10 return token;

11 }

12

13 public void setToken(String token) {

14 this.token = token;

15 }

16}

That's it! With these classes, you should be able to implement JWT authentication and authorization in your Spring Boot with MongoDB project. Note that you will need to modify the code to fit your specific use case.