How to Implement JWT Authentication & Authorization using SQL

sahu-himanshu

jwt.txt

What is ‘JWT’? {

JWT stands for JSON Web Token. It is an open standard (RFC 7519) used for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed.

}

in sahu-himanshu

steps.txt

Steps to ‘Implement’ {

01 Write the Entity classes for User and Role

02 Setup SQL & Create Role/User Repo

03 Create Controller and Services for User and Roles

04 Create Security Configuration Files

}

Java (Spring Boot)

sahu-himanshu

steps.txt

Steps to ‘Implement’ {

05 Implement UserDetailsService

06 Implement AuthenticationEntryPoint and AuthenticationFilter

07 Create JWT Classes: JWTResponse, JWTRequest, JWTHelper

}

sahu-himanshu

User ‘Class’;

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
```

```java
@Getter
@Setter
@AllArgsConstructor
@Entity
@Table(name = "users")
public class User implements UserDetails {
    @Id
    @Column(name = "id")
    private String userId;
    private String name;
    private String email;
    private String password;
    @ManyToMany(fetch = FetchType.EAGER, cascade = CascadeType.ALL)
    @JoinTable(name = "employee_roles",
            joinColumns = @JoinColumn(name = "employee_id", referencedColumnName = "id"),
            inverseJoinColumns = @JoinColumn(name = "role_id", referencedColumnName =
"id")
    )
    @JsonManagedReference
    private List<Role> roles;
}
```

User.java

Role.java

Role ‘Class’;

```
@Getter
@Setter
@AllArgsConstructor
@Entity
@Table(name = "role")
public class Role {
    @Id
    @Column(name = "id")
    private String roleId;

    private String name;

    @ManyToMany(mappedBy = "roles")
    @JsonBackReference
    private List<User> users;
}
```


sahu-himanshu

UserRepository.java

RoleRepository.java

User & RoleRepository ‘Class’;

```
# UserRepository.java
@Repository
public interface UserRepository extends JpaRepository<User, String> {
    Optional<User> findByEmail(String email);
}

# RoleRepository.java
@Repository
public interface RoleRepository extends JpaRepository<Role, String> {
    Optional<Role> findById(String id);
}
```

AdminController.java

AuthController.java

AdminController ‘Class’;

```java
@RestController
@RequestMapping("/admin")
public class AdminController {

    @Autowired
    private UserService userService;

    @PostMapping("/create")
    public String user(@RequestBody User user) {
        return userService.createUser(user);
    }
}
```


sahu-himanshu

AdminController.java

AuthController.java

AuthController ‘Class’;

```java
@RestController
@RequestMapping("/auth")
public class AuthController {

    @Autowired
    private UserDetailsService userDetailsService;
    @Autowired
    private AuthenticationManager manager;
    @Autowired
    private JwtHelper helper;

    private Logger logger = LoggerFactory.getLogger(AuthController.class);

    @PostMapping("/login")
    public ResponseEntity<JwtResponse> login(@RequestBody JwtRequest request) {

        this.doAuthenticate(request.getEmail(), request.getPassword());
        UserDetails userDetails = userDetailsService.loadUserByUsername(request.getEmail());
        String token = this.helper.generateToken(userDetails);
        JwtResponse response = JwtResponse.builder()
                .jwtToken(token)
                .username(userDetails.getUsername())
                .build();
        return new ResponseEntity<>(response, HttpStatus.OK);
    }

    private void doAuthenticate(String email, String password) {

        UsernamePasswordAuthenticationToken authentication = new UsernamePasswordAuthenticationToken(email, password);
        try {
            manager.authenticate(authentication);


        } catch (BadCredentialsException e) {
            throw new BadCredentialsException(" Invalid Username or Password  !!");
        }

    }

    @ExceptionHandler(BadCredentialsException.class)
    public String exceptionHandler() {
        return "Credentials Invalid !!";
    }

}
```

Java (Spring Boot)

sahu-himanshu

UserController.java

UserController ‘Class’;

```java
@RestController
@RequestMapping("/user")
public class UserController {

    @Autowired
    private UserService userService;

    @RequestMapping("/getevents")
    private String getEvents() {
        return "Events";
    }

    @GetMapping("/all")
    public List<User> user() {
        return this.userService.getAllUsers();
    }
}
```

Java (Spring Boot)

sahu-himanshu

AuthController.java

UserService.java

UserService ‘Class’;

```
@Service
public class UserService {

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private RoleRepository roleRepository;

    @Autowired
    private PasswordEncoder passwordEncoder;

    public List<User> getAllUsers() {
        return userRepository.findAll();
    }
```

```
public String createUser(User user) {

user.setUserId(UUID.randomUUID().toString());

user.setPassword(passwordEncoder.encode(user.get
Password()));

        List<Role> roles = user.getRoles();
        for (Role role: roles) {
            Role r = new Role();

r.setRoleId(UUID.randomUUID().toString());
            r.setName(role.getName());
//            roleRepository.save(r);
        }
        userRepository.save(user);
        return user.getUserId();
    }

}
```

Java (Spring Boot)

sahu-himanshu

AppConfig ‘Class’;

```
@Configuration
public class AppConfig {

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}
```


in sahu-himanshu

AppConfig.java

Security.java

SecurityConfig ‘Class’;

```
@Configuration
public class SecurityConfig {

    @Autowired
    private JWTAuthentcationEntryPoint point;
    @Autowired
    private JwtAuthenticationFilter filter;
    @Autowired
    private CustomUserDetailsService userDetailsService;

    @Autowired
    private PasswordEncoder passwordEncoder;

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity
http) throws Exception {
        http.csrf(csrf → csrf.disable())
                .authorizeHttpRequests(auth → auth
                        .requestMatchers("/auth/login").permitAll()

.requestMatchers("/admin/create").permitAll()

.requestMatchers("/user/all").hasAuthority("USER")
                        .anyRequest().hasAnyAuthority("ADMIN",
"USER"))
                .exceptionHandling(ex →
ex.authenticationEntryPoint(point))
                .sessionManagement(session →
session.sessionCreationPolicy(SessionCreationPolicy.STATELESS));
        http.addFilterBefore(filter,
UsernamePasswordAuthenticationFilter.class);
        return http.build();
    }
```

```
 @Bean
    public DaoAuthenticationProvider daoAuthenticationProvider() {
        DaoAuthenticationProvider daoAuthenticationProvider = new
DaoAuthenticationProvider();

daoAuthenticationProvider.setUserDetailsService(userDetailsService);

daoAuthenticationProvider.setPasswordEncoder(passwordEncoder);
        return daoAuthenticationProvider;
    }

    @Bean
    public AuthenticationManager
authenticationManager(AuthenticationConfiguration configuration)
throws Exception {
        return configuration.getAuthenticationManager();
    }
}
```

Java (Spring Boot)

sahu-himanshu

CustomUserDetailsService.java

UserService.java

UserDetailsService ‘Class’;

```
@Service
public class CustomUserDetailsService implements UserDetailsService {

    @Autowired
    private UserRepository userRepository;

    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        User user = userRepository.findByEmail(username).orElseThrow(() → new
RuntimeException("User Not found!"));
        return user;
    }
}
```

JWTAuthenticationEntryPoint ‘Class’;

```java
@Component
public class JWTAuthentcationEntryPoint implements AuthenticationEntryPoint {
    @Override
    public void commence(HttpServletRequest request, HttpServletResponse response,
AuthenticationException authException) throws IOException, ServletException {
        response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
        PrintWriter printWriter = response.getWriter();
        printWriter.println("Access Denied!!!" + authException.getMessage());
    }
}
```


in sahu-himanshu

JWTAuthenticationEntryPoint.java

JWTAuthenticationFilter.java

JWTAuthenticationFilter ‘Class’;

```java
@Component
public class JwtAuthenticationFilter extends OncePerRequestFilter {

    private Logger logger =
LoggerFactory.getLogger(OncePerRequestFilter.class);
    @Autowired
    private JwtHelper jwtHelper;
    @Autowired
    private UserDetailsService userDetailsService;
    @Override
    protected void doFilterInternal(HttpServletRequest request,
HttpServletResponse response, FilterChain filterChain) throws ServletException,
IOException {
        String requestHeader = request.getHeader("Authorization");
        logger.info(" Header :  {}", requestHeader);
        String username = null;
        String token = null;
        if (requestHeader != null && requestHeader.startsWith("Bearer")) {
            token = requestHeader.substring(7);
            try {
                username = this.jwtHelper.getUsernameFromToken(token);
            } catch (IllegalArgumentException e) {
                logger.info("Illegal Argument while fetching the username !!");
                e.printStackTrace();
            } catch (ExpiredJwtException e) {
                logger.info("Given jwt token is expired !!");
                e.printStackTrace();
            } catch (MalformedJwtException e) {
                logger.info("Some changed has done in token !! Invalid Token");
                e.printStackTrace();
            } catch (Exception e) {
                e.printStackTrace();
            }
          else {
            logger.info("Invalid Header Value !! ");
          }
```

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

            UserDetails userDetails =
this.userDetailsService.loadUserByUsername(username);
            Boolean validateToken = this.jwtHelper.validateToken(token,
userDetails);
            if (validateToken) {

                UsernamePasswordAuthenticationToken authentication = new
UsernamePasswordAuthenticationToken(userDetails, null,
userDetails.getAuthorities());
                authentication.setDetails(new
WebAuthenticationDetailsSource().buildDetails(request));


SecurityContextHolder.getContext().setAuthentication(authentication);


            } else {
                logger.info("Validation fails !!");
            }


        }

        filterChain.doFilter(request, response);


    }
}
```

Java (Spring Boot)

in sahu-himanshu

JWTResponse ‘Class’;

```
@Getter
@Setter
@AllArgsConstructor
@Builder
public class JwtResponse {

    private String jwtToken;
    private String username;
}
```


sahu-himanshu

JWTRequest ‘Class’;

```java
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Builder
@ToString
public class JwtRequest {
    private String email;
    private String password;
}
```

sahu-himanshu

JWTHelper.java

JWTHelper ‘Class’;

```
@Component
public class JwtHelper {
    public static final long JWT_TOKEN_VALIDITY = 5 * 60 * 60;
    private String secret =
"afafasfafafasfasfasfafacasdasfasxASFACASDFACASDFASFASFDAFASFASDAAD
SCSDFADCVSGCFVADXCcadwavfsfarvf";

    public String getUsernameFromToken(String token) {
        return getClaimFromToken(token, Claims::getSubject);
    }
    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);
    }
    private Claims getAllClaimsFromToken(String token) {
        return
Jwts.parser().setSigningKey(secret).parseClaimsJws(token).getBody()
;
    }
    private Boolean isTokenExpired(String token) {
        final Date expiration = getExpirationDateFromToken(token);
        return expiration.before(new Date());
    }
    public String generateToken(UserDetails userDetails) {
        Map<String, Object> claims = new HashMap<>();
        return doGenerateToken(claims, userDetails.getUsername());
    }
    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();
    }
    public Boolean validateToken(String token, UserDetails userDetails) {
        final String username = getUsernameFromToken(token);
        return (username.equals(userDetails.getUsername()) &&
!isTokenExpired(token));
    }
}
```


sahu-himanshu

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
```

Thanks {

Guys;

}

in sahu-himanshu