

# Spring Boot JWT Security + MySQL

pom.xml

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
</dependency>

<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.46</version><!--$NO-MVN-MAN-VER$ -->
</dependency>
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.9.1</version>
</dependency>

<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <scope>provided</scope>
</dependency>
```

### application.properties

```
#Server
server.port=9966
app.secret=raghu

#DataSource
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/boot
spring.datasource.username=root
spring.datasource.password=root
#JPA
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
spring.jpa.database-platform=org.hibernate.dialect.MySQL55Dialect
spring.jpa.properties.hibernate.format_sql=true
```

### 1. JwtUtil

```
package in.nit.raghu.util;

import java.util.Date;
import java.util.HashMap;
import java.util.Map;
import java.util.concurrent.TimeUnit;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.SignatureAlgorithm;

@Component
public class JwtUtil {

    @Value("${app.secret}")
    private String secret;

    //7. validate token user name and request user also expDate
    public boolean validateToken(String token, String username) {
        String usernameInToken = getUsername(token);
```

```
        return (usernameInToken.equals(username) && !isTokenExpired(token));
    }

    //6. Check Current and Exp Date
    private boolean isTokenExpired(String token) {
        final Date expiration = getExpDate(token);
        return expiration.before(new Date());
    }

    //5. Generate Token with Empty Claims
    public String generateToken(String username) {
        Map<String, Object> claims = new HashMap<>();
        return generateToken(claims, username);
    }

    //4. Read username
    public String getUsername(String token) {
        return getClaims(token).getSubject();
    }

    //3. read ExpDate
    public Date getExpDate(String token) {
        return getClaims(token).getExpiration();
    }

    //2. Read Claim
    private Claims getClaims(String token) {
        return Jwts.parser()
            .setSigningKey(secret)
            .parseClaimsJws(token)
            .getBody();
    }

    //1. generate token
    private String generateToken(Map<String, Object> claims, String subject) {
        return Jwts.builder()
            .setClaims(claims)
            .setSubject(subject)
            .setIssuer("RAGHU")
            .setIssuedAt(new Date(System.currentTimeMillis()))
            .setExpiration(new Date(System.currentTimeMillis() +
                TimeUnit.MINUTES.toMillis(30)))
            .signWith(SignatureAlgorithm.HS512, secret)
            .compact();
    }
}
```

```
}
```

## 2. Entity classes

a. User

```
package in.nit.raghu.entity;

import java.util.List;

import javax.persistence.ElementCollection;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.Table;

import lombok.Data;

@Entity
@Table(name="app_user_tab")
@Data
public class User {
    @Id
    @GeneratedValue
    private Integer id;
    private String name;
    private String username;
    private String password;
    @ElementCollection
    private List<String> roles;
}
```

b. UserRequest

```
package in.nit.raghu.entity;

import lombok.Data;

@Data
public class UserRequest {

    private String username;
```

```
        private String password;  
    }
```

c. UserResponse

```
package in.nit.raghu.entity;  
  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
public class UserResponse {  
  
    private String token;  
    private String note;  
}
```

### **3. Repository interface**

```
package in.nit.raghu.repository;  
  
import java.util.Optional;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
  
import in.nit.raghu.entity.User;  
  
public interface UserRepository extends JpaRepository<User, Integer> {  
  
    Optional<User> findByUsername(String username);  
}
```

### **4. Service Interface**

```
package in.nit.raghu.service;  
  
import in.nit.raghu.entity.User;  
  
public interface IUserService {
```

```
    public Integer saveUser(User user);  
    public User findByUsername(String username);  
}
```

### **5. UserServiceImpl**

```
package in.nit.raghu.service.impl;  
  
import java.util.List;  
import java.util.Optional;  
import java.util.stream.Collectors;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.security.core.GrantedAuthority;  
import org.springframework.security.core.authority.SimpleGrantedAuthority;  
import org.springframework.security.core.userdetails.UserDetails;  
import org.springframework.security.core.userdetails.UserDetailsService;  
import org.springframework.security.core.userdetails.UsernameNotFoundException;  
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
  
import in.nit.raghu.entity.User;  
import in.nit.raghu.repository.UserRepository;  
import in.nit.raghu.service.IUserService;  
  
@Service  
public class UserServiceImpl implements IUserService, UserDetailsService {  
    @Autowired  
    private UserRepository repository;  
    @Autowired  
    private BCryptPasswordEncoder pwdEncoder;  
  
    @Transactional  
    public Integer saveUser(User user) {  
        user.setPassword(  
            pwdEncoder.encode(user.getPassword())  
        );  
        return repository.save(user).getId();  
    }  
  
    @Transactional(readOnly = true)  
    public User findByUsername(String username) {  
        Optional<User> user = repository.findByUsername(username);  
        if (user.isPresent())
```

```
        return user.get();
    }
    return null;
}

@Transactional(readOnly = true)
public UserDetails loadUserByUsername(String username)
    throws UsernameNotFoundException
{
    User user=findByUsername(username);
    if(user==null)
        throw new UsernameNotFoundException(
            new StringBuffer()
                .append("User name ")
                .append(username)
                .append(" not found!")
                .toString()
        );

    List<GrantedAuthority> authorities=
        user.getRoles()
            .stream()
            .map(
                role->new SimpleGrantedAuthority(role)
            )
            .collect(Collectors.toList());

    return new org.springframework.security.core.userdetails.User(
        username,
        user.getPassword(),
        authorities);
}
}
```

## **6. User Controller**

```
package in.nit.raghu.controller;

import java.security.Principal;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
```

```
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import in.nit.raghu.entity.User;
import in.nit.raghu.entity.UserRequest;
import in.nit.raghu.entity.UserResponse;
import in.nit.raghu.service.IUserService;
import in.nit.raghu.util.JwtUtil;

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

    @Autowired
    private AuthenticationManager authenticationManager;

    @Autowired
    private IUserService service;

    @Autowired
    private JwtUtil jwtUtil;

    @PostMapping("/save")
    public ResponseEntity<String> saveUser(@RequestBody User user) {
        Integer id=service.saveUser(user);
        return ResponseEntity.ok("User saved with id"+id);
    }

    @PostMapping("/login")
    public ResponseEntity<UserResponse> loginUser(@RequestBody UserRequest
userRequest)
    {

        authenticationManager.authenticate(
            new UsernamePasswordAuthenticationToken(
                userRequest.getUsername(),
                userRequest.getPassword()
            )
        );

        String token=jwtUtil.generateToken(userRequest.getUsername());

        return ResponseEntity.ok(new UserResponse(token,"GENERATED BY MR.RAGHU
-NIT"));
    }
}
```



```
}

@PostMapping("/welcome")
public ResponseEntity<String> accessUserData(Principal p) {
    return ResponseEntity.ok("Hello user:"+p.getName());
}

}
```

### **7. AppConfig**

```
package in.nit.raghu.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

@Configuration
public class AppConfig {

    @Bean
    public BCryptPasswordEncoder pwdEncoder() {
        return new BCryptPasswordEncoder();
    }

}
```

### **8. Exception Entry Point**

```
package in.nit.raghu.config;

import java.io.IOException;

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

import org.springframework.security.core.AuthenticationException;
import org.springframework.security.web.AuthenticationEntryPoint;
import org.springframework.stereotype.Component;
```

```
@Component
public class InvalidUserAuthenticationEntryPoint implements
AuthenticationEntryPoint {

    @Override
    public void commence(HttpServletRequest request, HttpServletResponse
response,
        AuthenticationException authException) throws IOException, ServletException
    {
        response.sendError(HttpServletResponse.SC_UNAUTHORIZED, "Unauthorized User!");
    }
}
```

### **9. SecurityConfig**

```
package in.nit.raghu.config;

import org.springframework.beans.factory.annotation.Autowired;
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.Authentica
tionManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecurit
y;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConf
igurerAdapter;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.AuthenticationEntryPoint;
import
org.springframework.security.web.authentication.UsernamePasswordAuthenticationFil
ter;

import in.nit.raghu.filter.SecurityFilter;
```

```
@EnableWebSecurity
@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter{
    @Autowired
    private UserDetailsService service;
    @Autowired
    private PasswordEncoder passwordEncoder;
    @Autowired
    private AuthenticationEntryPoint authenticationEntryPoint;
    @Autowired
    private SecurityFilter securityFilter;

    @Override
    @Bean
    public AuthenticationManager authenticationManagerBean() throws Exception {
        return super.authenticationManagerBean();
    }

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception
    {
        auth.userDetailsService(service).passwordEncoder(passwordEncoder);
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.csrf().disable()
            .authorizeRequests()
            .antMatchers("/user/save", "/user/login").permitAll()
            .anyRequest().authenticated()
            .and()
            .exceptionHandling()
            .authenticationEntryPoint(authenticationEntryPoint)
            .and()

            .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
            .and()
            .addFilterBefore(securityFilter,
UsernamePasswordAuthenticationFilter.class);
    }
}
```

## **10. SecurityFilter**

```
package in.nit.raghu.filter;

import java.io.IOException;

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

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;

import in.nit.raghu.util.JwtUtil;

@Component
public class SecurityFilter extends OncePerRequestFilter {

    @Autowired
    private JwtUtil jwtUtil;
    @Autowired
    private UserDetailsService userDetailsService;

    @Override
    protected void doFilterInternal(
        HttpServletRequest request,
        HttpServletResponse response,
        FilterChain filterChain)
        throws ServletException, IOException {

        String token=request.getHeader("Authorization");
        if(token!=null) {

            String username=jwtUtil.getUsername(token);

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

                UserDetails user=userDetailsService.loadUserByUsername(username);
```

```
boolean isValid=jwtUtil.validateToken(token, user.getUsername());

if(isValid) {
    UsernamePasswordAuthenticationToken authToken=new
UsernamePasswordAuthenticationToken(username, user.getPassword(),
user.getAuthorities());

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

    SecurityContextHolder.getContext().setAuthentication(authToken);
    }
    }
    filterChain.doFilter(request, response);
}
}
```