Okay, let's proceed with Task 2: **Implementing an Authentication and Authorization System.**

Here's a more complete code implementation, incorporating JWT authentication, user roles, and Spring Security.

**1. User Entity and Repository**

import javax.persistence.\*;  
import java.util.Set;  
  
@Entity  
@Table(name = "users")  
public class User {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private Long id;  
  
 @Column(unique = true, nullable = false)  
 private String username;  
  
 @Column(nullable = false)  
 private String password;  
  
 @ElementCollection(fetch = FetchType.EAGER)  
 @CollectionTable(name = "user\_roles", joinColumns = @JoinColumn(name = "user\_id"))  
 @Column(name = "role")  
 private Set<String> roles;  
  
 // Getters and setters  
 public Long getId() {  
 return id;  
 }  
  
 public void setId(Long id) {  
 this.id = id;  
 }  
  
 public String getUsername() {  
 return username;  
 }  
  
 public void setUsername(String username) {  
 this.username = username;  
 }  
  
 public String getPassword() {  
 return password;  
 }  
  
 public void setPassword(String password) {  
 this.password = password;  
 }  
  
 public Set<String> getRoles() {  
 return roles;  
 }  
  
 public void setRoles(Set<String> roles) {  
 this.roles = roles;  
 }  
}

import org.springframework.data.jpa.repository.JpaRepository;  
  
public interface UserRepository extends JpaRepository<User, Long> {  
 User findByUsername(String username);  
}

**2. JWT Utility Class**

import io.jsonwebtoken.Claims;  
import io.jsonwebtoken.Jwts;  
import io.jsonwebtoken.SignatureAlgorithm;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.security.core.userdetails.UserDetails;  
import org.springframework.stereotype.Component;  
  
import java.util.Date;  
import java.util.HashMap;  
import java.util.Map;  
import java.util.function.Function;  
  
@Component  
public class JwtUtil {  
  
 @Value("${jwt.secret}")  
 private String secret;  
  
 @Value("${jwt.expiration}")  
 private Long jwtExpiration;  
  
 public String extractUsername(String token) {  
 return extractClaim(token, Claims::getSubject);  
 }  
  
 public Date extractExpiration(String token) {  
 return extractClaim(token, Claims::getExpiration);  
 }  
  
 public <T> T extractClaim(String token, Function<Claims, T> claimsResolver) {  
 final Claims claims = extractAllClaims(token);  
 return claimsResolver.apply(claims);  
 }  
  
 private Claims extractAllClaims(String token) {  
 return Jwts.parser().setSigningKey(secret).parseClaimsJws(token).getBody();  
 }  
  
 public Boolean isTokenExpired(String token) {  
 return extractExpiration(token).before(new Date());  
 }  
  
 public String generateToken(UserDetails userDetails) {  
 Map<String, Object> claims = new HashMap<>();  
 return createToken(claims, userDetails.getUsername());  
 }  
  
 private String createToken(Map<String, Object> claims, String subject) {  
 return Jwts.builder().setClaims(claims).setSubject(subject)  
 .setIssuedAt(new Date(System.currentTimeMillis()))  
 .setExpiration(new Date(System.currentTimeMillis() + jwtExpiration))  
 .signWith(SignatureAlgorithm.HS256, secret).compact();  
 }  
  
 public Boolean validateToken(String token, UserDetails userDetails) {  
 final String username = extractUsername(token);  
 return (username.equals(userDetails.getUsername()) && !isTokenExpired(token));  
 }  
}

**3. UserDetailsService Implementation**

import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.security.core.userdetails.UserDetails;  
import org.springframework.security.core.userdetails.UserDetailsService;  
import org.springframework.security.core.userdetails.UsernameNotFoundException;  
import org.springframework.stereotype.Service;  
  
import java.util.Collections;  
import java.util.stream.Collectors;  
  
@Service  
public class MyUserDetailsService implements UserDetailsService {  
  
 @Autowired  
 private UserRepository userRepository;  
  
 @Override  
 public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  
 User user = userRepository.findByUsername(username);  
 if (user == null) {  
 throw new UsernameNotFoundException("User not found with username: " + username);  
 }  
 return new org.springframework.security.core.userdetails.User(  
 user.getUsername(),  
 user.getPassword(),  
 user.getRoles().stream()  
 .map(role -> new org.springframework.security.core.authority.SimpleGrantedAuthority("ROLE\_" + role))  
 .collect(Collectors.toList())  
 );  
 }  
}

**4. Security Configuration**

import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.context.annotation.Bean;  
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.config.http.SessionCreationPolicy;  
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;  
import org.springframework.security.crypto.password.PasswordEncoder;  
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;  
  
@EnableWebSecurity  
public class SecurityConfig extends WebSecurityConfigurerAdapter {  
  
 @Autowired  
 private MyUserDetailsService userDetailsService;  
  
 @Autowired  
 private JwtRequestFilter jwtRequestFilter;  
  
 @Override  
 protected void configure(AuthenticationManagerBuilder auth) throws Exception {  
 auth.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder());  
 }  
  
 @Override  
 protected void configure(HttpSecurity http) throws Exception {  
 http.csrf().disable()  
 .authorizeRequests()  
 .antMatchers("/auth/\*\*").permitAll()  
 .antMatchers("/admin/\*\*").hasRole("ADMIN")  
 .antMatchers("/user/\*\*").hasAnyRole("USER", "ADMIN")  
 .anyRequest().authenticated()  
 .and().sessionManagement()  
 .sessionCreationPolicy(SessionCreationPolicy.STATELESS);  
  
 http.addFilterBefore(jwtRequestFilter, UsernamePasswordAuthenticationFilter.class);  
 }  
  
 @Bean  
 public PasswordEncoder passwordEncoder() {  
 return new BCryptPasswordEncoder();  
 }  
  
 @Override  
 @Bean  
 public AuthenticationManager authenticationManagerBean() throws Exception {  
 return super.authenticationManagerBean();  
 }  
}

**5. JWT Request Filter**

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.web.authentication.WebAuthenticationDetailsSource;  
import org.springframework.stereotype.Component;  
import org.springframework.web.filter.OncePerRequestFilter;  
  
import javax.servlet.FilterChain;  
import javax.servlet.ServletException;  
import javax.servlet.http.HttpServletRequest;  
import javax.servlet.http.HttpServletResponse;  
import java.io.IOException;  
  
@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 usernamePasswordAuthenticationToken = new UsernamePasswordAuthenticationToken(  
 userDetails, null, userDetails.getAuthorities());  
 usernamePasswordAuthenticationToken  
 .setDetails(new WebAuthenticationDetailsSource().buildDetails(request));  
 SecurityContextHolder.getContext().setAuthentication(usernamePasswordAuthenticationToken);  
 }  
 }  
 chain.doFilter(request, response);  
 }  
}

**6. Authentication and Registration Controller**

import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.ResponseEntity;  
import org.springframework.security.authentication.AuthenticationManager;  
import org.springframework.security.authentication.BadCredentialsException;  
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;  
import org.springframework.security.core.userdetails.UserDetails;  
import org.springframework.security.crypto.password.PasswordEncoder;  
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 java.util.Collections;  
import java.util.HashSet;  
import java.util.Set;  
  
@RestController  
@RequestMapping("/auth")  
public class AuthController {  
  
 @Autowired  
 private AuthenticationManager authenticationManager;  
  
 @Autowired  
 private MyUserDetailsService userDetailsService;  
  
 @Autowired  
 private JwtUtil jwtUtil;  
  
 @Autowired  
 private UserRepository userRepository;  
  
 @Autowired  
 private PasswordEncoder passwordEncoder;  
  
 @PostMapping("/authenticate")  
 public ResponseEntity<?> createAuthenticationToken(@RequestBody AuthenticationRequest authenticationRequest) throws Exception {  
 try {  
 authenticationManager.authenticate(  
 new UsernamePasswordAuthenticationToken(authenticationRequest.getUsername(), authenticationRequest.getPassword())  
 );  
 } catch (BadCredentialsException e) {  
 throw new Exception("Incorrect username or password", e);  
 }  
  
 final UserDetails userDetails = userDetailsService.loadUserByUsername(authenticationRequest.getUsername());  
 final String jwt = jwtUtil.generateToken(userDetails);  
  
 return ResponseEntity.ok(new AuthenticationResponse(jwt));  
 }  
  
 @PostMapping("/register")  
 public ResponseEntity<?> registerUser(@RequestBody RegistrationRequest registrationRequest) {  
 User user = new User();  
 user.setUsername(registrationRequest.getUsername());  
 user.setPassword(passwordEncoder.encode(registrationRequest.getPassword()));  
 Set<String> roles = new HashSet<>();  
 roles.add("USER"); // Default role  
 user.setRoles(roles);  
 userRepository.save(user);  
 return ResponseEntity.ok("Registration successful");  
 }  
}  
  
class AuthenticationRequest {  
 private String username;  
 private String password;  
  
 public String getUsername() {  
 return username;  
 }  
  
 public void setUsername(String username) {  
 this.username = username;  
 }  
  
 public String getPassword() {  
 return password;  
 }  
  
 public void setPassword(String password) {  
 this.password = password;  
 }  
}  
  
class AuthenticationResponse {  
 private final String jwt;  
  
 public AuthenticationResponse(String jwt) {  
 this.jwt = jwt;  
 }  
  
 public String getJwt() {  
 return jwt;  
 }  
}  
  
class RegistrationRequest {  
 private String username;  
 private String password;  
  
 public String getUsername() {  
 return username;  
 }  
  
 public void setUsername(String username) {  
 this.username = username;  
 }  
  
 public String getPassword() {  
 return password;  
 }  
  
 public void setPassword() {  
 this.password = password;  
 }  
}

**7. Example Secured Controller**

import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
@RestController  
public class ExampleController {  
  
 @GetMapping("/user/profile")  
 public String userEndpoint() {  
 return "User profile data";  
 }  
  
 @GetMapping("/admin/dashboard")  
 public String adminEndpoint() {  
 return "Admin dashboard data";  
 }  
}

**Key Improvements and Explanations**

* **Role Prefix:** In MyUserDetailsService, roles are now prefixed with "ROLE\_" before being granted. Spring Security expects roles to be in this format (e.g., "ROLE\_USER", "ROLE\_ADMIN").
* **Authorization Rules:**
  + /auth/\*\* is permitted for all (registration and authentication).
  + /admin/\*\* requires the "ADMIN" role.
  + /user/\*\* requires either "USER" or "ADMIN" role.
  + All other endpoints require authentication.
* **Authentication and Registration Controllers:**
  + Cleaned up request and response classes.
  + Registration now assigns a default role "USER".
* **Example Secured Controller:** Demonstrates how to secure endpoints based on roles.
* **Security Configuration:** Updated to use the correct role format and demonstrate role-based authorization.

This revised code provides a more robust and accurate implementation of authentication and authorization using JWT and Spring Security.