Authentication Admin

Here's a **Spring Boot example project** outline where:

- An **admin** can **register** and **login**.
- All API methods are protected so **only the admin can access them after authentication** using **JWT**.
- It includes **Spring Security**, **JWT**, **role-based access**, and **admin-only endpoints**.

✅ **Project Features**

1. **Admin Registration & Login**
2. **JWT Authentication**
3. **Role-based Authorization** (Only ROLE_ADMIN)
4. **Protected Endpoints (CRUD actions)**

🧱 **Project Structure**

arduino

CopyEdit

src

├── config

│ └── SecurityConfig.java

├── controller

│ └── AdminController.java

├── dto

│ └── AuthRequest.java

│ └── AuthResponse.java

│ └── RegisterRequest.java

├── entity

│ └── Admin.java

├── repository

│ └── AdminRepository.java

├── service

│ └── AdminService.java

├── util

│ └── JwtUtil.java

└── Application.java

🧪 **Sample Endpoints**

Method	Endpoint	Description
POST	/api/auth/register	Register admin
POST	/api/auth/login	Login and get JWT
GET	/api/admin/data	Get admin-only data

```
// Main Application Class
package com.example.adminjwt;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class AdminJwtApplication {
    public static void main(String[] args) {
        SpringApplication.run(AdminJwtApplication.class, args);
    }
}

// Entity
package com.example.adminjwt.entity;
```

```
import jakarta.persistence.*;
import lombok.*;

@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Admin {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;

   private String username;
   private String password;
   private String role = "ROLE_ADMIN";
}

// Repository
package com.example.adminjwt.repository;

import com.example.adminjwt.entity.Admin;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.Optional;

public interface AdminRepository extends JpaRepository<Admin, Long> {
   Optional<Admin> findByUsername(String username);
}
```

```
// DTOs
package com.example.adminjwt.dto;

import lombok.*;

@Data
public class RegisterRequest {
   private String username;
   private String password;
}

@Data
public class AuthRequest {
   private String username;
   private String password;
}

@Data
@AllArgsConstructor
public class AuthResponse {
   private String token;
}

// Service
package com.example.adminjwt.service;

import com.example.adminjwt.dto.*;
import com.example.adminjwt.entity.Admin;
```

```
import com.example.adminjwt.repository.AdminRepository;
import com.example.adminjwt.util.JwtUtil;
import lombok.RequiredArgsConstructor;
import org.springframework.security.authentication.*;
import org.springframework.security.core.userdetails.*;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;

@Service
@RequiredArgsConstructor
public class AdminService {

    private final AdminRepository adminRepository;
    private final PasswordEncoder passwordEncoder;
    private final AuthenticationManager authenticationManager;
    private final JwtUtil jwtUtil;

    public String register(RegisterRequest request) {
        Admin admin = new Admin();
        admin.setUsername(request.getUsername());
        admin.setPassword(passwordEncoder.encode(request.getPassword()));
        adminRepository.save(admin);
        return "Admin registered successfully!";
    }

    public AuthResponse login(AuthRequest request) {
        authenticationManager.authenticate(
            new UsernamePasswordAuthenticationToken(
                request.getUsername(), request.getPassword())
```

```
        );
        UserDetails user = new User(request.getUsername(), "",
            java.util.List.of(() -> "ROLE_ADMIN"));
        String token = jwtUtil.generateToken(user);
        return new AuthResponse(token);
    }
}

// Controller
package com.example.adminjwt.controller;

import com.example.adminjwt.dto.*;
import com.example.adminjwt.service.AdminService;
import lombok.RequiredArgsConstructor;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api")
@RequiredArgsConstructor
public class AdminController {

    private final AdminService adminService;

    @PostMapping("/auth/register")
    public String register(@RequestBody RegisterRequest request) {
        return adminService.register(request);
    }

    @PostMapping("/auth/login")
```

```
    public AuthResponse login(@RequestBody AuthRequest request) {
        return adminService.login(request);
    }

    @GetMapping("/admin/data")
    public String adminData() {
        return "This is confidential admin data.";
    }
}

// Security Config
package com.example.adminjwt.config;

import com.example.adminjwt.util.JwtFilter;
import lombok.RequiredArgsConstructor;
import org.springframework.context.annotation.*;
import org.springframework.security.authentication.*;
import org.springframework.security.config.annotation.authentication.configuration.*;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
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.SecurityFilterChain;
import
org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

@Configuration
@RequiredArgsConstructor
public class SecurityConfig {
```

```
    private final JwtFilter jwtFilter;

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

    @Bean
    public AuthenticationManager authManager(AuthenticationConfiguration config) throws
Exception {
        return config.getAuthenticationManager();
    }

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http.csrf().disable()
            .authorizeHttpRequests()
            .requestMatchers("/api/auth/**").permitAll()
            .anyRequest().hasRole("ADMIN")
            .and()
            .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
            .and()
            .addFilterBefore(jwtFilter, UsernamePasswordAuthenticationFilter.class);

        return http.build();
    }
}

// JWT Utility
package com.example.adminjwt.util;
```

```
import io.jsonwebtoken.*;
import io.jsonwebtoken.security.Keys;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.stereotype.Component;

import javax.crypto.SecretKey;
import java.util.Date;

@Component
public class JwtUtil {
    private final SecretKey key = Keys.secretKeyFor(SignatureAlgorithm.HS256);
    private final long EXPIRATION_TIME = 1000 * 60 * 60; // 1 hour

    public String generateToken(UserDetails user) {
        return Jwts.builder()
                .setSubject(user.getUsername())
                .claim("role", "ROLE_ADMIN")
                .setIssuedAt(new Date())
                .setExpiration(new Date(System.currentTimeMillis() + EXPIRATION_TIME))
                .signWith(key)
                .compact();
    }

    public String extractUsername(String token) {
        return Jwts.parserBuilder().setSigningKey(key).build()
                .parseClaimsJws(token).getBody().getSubject();
    }
```

```
    public boolean validateToken(String token, UserDetails userDetails) {
        String username = extractUsername(token);
        return username.equals(userDetails.getUsername()) && !isTokenExpired(token);
    }

    private boolean isTokenExpired(String token) {
        Date expiration = Jwts.parserBuilder().setSigningKey(key).build()
                .parseClaimsJws(token).getBody().getExpiration();
        return expiration.before(new Date());
    }
}

// JWT Filter
package com.example.adminjwt.util;

import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.*;
import lombok.RequiredArgsConstructor;
import org.springframework.security.authentication.*;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.*;
import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;

import java.io.IOException;

@Component
```

```
@RequiredArgsConstructor
public class JwtFilter extends OncePerRequestFilter {

    private final JwtUtil jwtUtil;

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

        final String authHeader = request.getHeader("Authorization");
        String username = null;
        String token = null;

        if (authHeader != null && authHeader.startsWith("Bearer ")) {
            token = authHeader.substring(7);
            username = jwtUtil.extractUsername(token);
        }

        if (username != null && SecurityContextHolder.getContext().getAuthentication() == null)
{
            UserDetails userDetails = new User(username, "", java.util.List.of(() ->
"ROLE_ADMIN"));
            if (jwtUtil.validateToken(token, userDetails)) {
                UsernamePasswordAuthenticationToken authToken =
                    new UsernamePasswordAuthenticationToken(userDetails, null,
userDetails.getAuthorities());
                authToken.setDetails(new
WebAuthenticationDetailsSource().buildDetails(request));
                SecurityContextHolder.getContext().setAuthentication(authToken);
            }
```

```
        }
        filterChain.doFilter(request, response);
    }
}
```

The full **Spring Boot Admin + JWT Authentication project** code has been generated. This includes registration, login, secure admin-only routes, and JWT token validation.

To run the project:

1. Set up your DB (H2 or MySQL) in application.properties.
2. Hit POST /api/auth/register to register an admin.
3. Hit POST /api/auth/login to get a JWT token.
4. Use that token (in Authorization: Bearer <token>) to access /api/admin/data.