**AuthController – Update Authorities Part 1**

**Add this code below in AuthController**

@PutMapping(value = "/users/update-authorities/{user\_id}", produces = "application/json", consumes = "application/json")

    @Operation(summary = "Update authorities")

    @ApiResponse(responseCode = "200", description = "Update authorities")

    @ApiResponse(responseCode = "401", description = "Token missing")

    @ApiResponse(responseCode = "403", description = "Token error")

    @SecurityRequirement(name = "studyeasy-demo-api")

    public AccountViewDTO update\_auth(@Valid @RequestBody AuthoritiesDTO authoritiesDTO, @PathVariable long user\_id){

        Optional<Account> optionalAccount = accountService.findById(user\_id);

        if(optionalAccount.isPresent()){

            Account account = optionalAccount.get();

            account.setAuthorities(authoritiesDTO.getAuthorities());

            accountService.save(account);

            AccountViewDTO accountViewDTO = new AccountViewDTO(account.getId(), account.getEmail(), account.getAuthorities());

            return accountViewDTO;

        }

        return null;

    }

**Create a new DTO -> AuthoritiesDTO.java**

package org.studyeasy.SpringRestdemo.payload.auth;

import io.swagger.v3.oas.annotations.media.Schema;

import io.swagger.v3.oas.annotations.media.Schema.RequiredMode;

import jakarta.validation.constraints.NotBlank;

import jakarta.validation.constraints.Size;

import lombok.Getter;

import lombok.Setter;

@Getter

@Setter

public class AuthoritiesDTO {

    @NotBlank

    @Size(min = 6, max = 20)

    @Schema(description = "Authorities", example = "USER", requiredMode = RequiredMode.REQUIRED)

    private String authorities;

}

**Add this in AccountService.java -> below piece of code**

 public Optional<Account> findById(long id) {

        return accountRepository.findById(id);

    }

**In SecurityConfig.java**

package org.studyeasy.SpringRestdemo.security;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.authentication.ProviderManager;

import org.springframework.security.authentication.dao.DaoAuthenticationProvider;

import org.springframework.security.config.Customizer;

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

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

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

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

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

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

import org.springframework.security.oauth2.jwt.JwtDecoder;

import org.springframework.security.oauth2.jwt.JwtEncoder;

import org.springframework.security.oauth2.jwt.NimbusJwtDecoder;

import org.springframework.security.oauth2.jwt.NimbusJwtEncoder;

import org.springframework.security.web.SecurityFilterChain;

import com.nimbusds.jose.JOSEException;

import com.nimbusds.jose.jwk.JWKSet;

import com.nimbusds.jose.jwk.RSAKey;

import com.nimbusds.jose.jwk.source.JWKSource;

import com.nimbusds.jose.proc.SecurityContext;

@Configuration

@EnableWebSecurity

public class SecurityConfig {

    private RSAKey rsaKey;

    @Bean

    public JWKSource<SecurityContext> jwkSource() {

        rsaKey = Jwks.generateRsa();

        JWKSet jwkSet = new JWKSet(rsaKey);

        return (jwkSelector, securityContext) -> jwkSelector.select(jwkSet);

    }

    @Bean

    public static PasswordEncoder passwordEncoder() {

        return new BCryptPasswordEncoder();

    }

    // @Bean

    // public InMemoryUserDetailsManager users() {

    // return new InMemoryUserDetailsManager(

    // User.withUsername("chaand")

    // .password("{noop}password")

    // .authorities("read")

    // .build());

    // }

    @Bean

    public AuthenticationManager authManager(UserDetailsService userDetailsService) {

        var authProvider = new DaoAuthenticationProvider();

        authProvider.setPasswordEncoder(passwordEncoder());

        authProvider.setUserDetailsService(userDetailsService);

        return new ProviderManager(authProvider);

    }

    @Bean

    JwtEncoder jwtEncoder(JWKSource<SecurityContext> jwks) {

        return new NimbusJwtEncoder(jwks);

    }

    @Bean

    JwtDecoder jwtDecoder() throws JOSEException {

        return NimbusJwtDecoder.withPublicKey(rsaKey.toRSAPublicKey()).build();

    }

    @Bean

    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {

        http

                // CSRF configuration

                .csrf(csrf -> csrf

                        .disable()) // Disable CSRF for stateless JWT authentication

                // Frame options for H2 console

                .headers(headers -> headers

                        .frameOptions(frameOptions -> frameOptions.sameOrigin()))

                // Authorization configuration

                .authorizeHttpRequests(authorize -> authorize

                        .requestMatchers("/auth/token").permitAll()

                        .requestMatchers("/auth/users/add").permitAll()

                        .requestMatchers("/auth/users").hasAuthority("SCOPE\_ADMIN")

                        .requestMatchers("/auth/users/update-authorities/\*\*").hasAuthority("SCOPE\_ADMIN")

                        .requestMatchers("/auth/profile").authenticated()

                        .requestMatchers("/auth/profile/update-password").authenticated()

                        .requestMatchers("/swagger-ui/\*\*").permitAll()

                        .requestMatchers("/v3/api-docs/\*\*").permitAll()

                        .requestMatchers("/test").authenticated() // `/test` requires authentication

                )

                // JWT-based authentication

                .oauth2ResourceServer(oauth2 -> oauth2

                        .jwt(Customizer.withDefaults()))

                // Stateless session management

                .sessionManagement(session -> session

                        .sessionCreationPolicy(SessionCreationPolicy.STATELESS))

                .headers(headers -> headers

                        .frameOptions(frameOptions -> frameOptions.sameOrigin())); // Required for H2 console

        return http.build();

    }

}