Security + JWT

Installation

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

Configuration

```
@Configuration
@EnableWebSecurity
@EnableMethodSecurity
class SecurityConfig {

    @Autowired
    lateinit var rsaKeyConfigProperties: RsaKeyConfigProperties

    @Autowired
    lateinit var userDetailsService: JpaUserDetailsService

    @Value("\${cors.allowedOrigins}")
    private lateinit var allowedOrigins: String


    @Bean
    fun authManager(): AuthenticationManager {
        val authProvider = DaoAuthenticationProvider()
        authProvider.setUserDetailsService(userDetailsService)
        authProvider.setPasswordEncoder(passwordEncoder())
        return ProviderManager(authProvider)
    }


    @Bean
    @Throws(Exception::class)
    fun filterChain(http: HttpSecurity, introspector: HandlerMappingIntrospector?):
SecurityFilterChain {
        return http
            .csrf { csrf: CsrfConfigurer<HttpSecurity> ->
                csrf.disable()
            }
            .cors(Customizer.withDefaults())
            .authorizeHttpRequests { auth ->
                auth.requestMatchers("/error/**").permitAll()
                auth.requestMatchers("/api/auth/**").permitAll()
```

```
                auth.requestMatchers("/h2-console/**").permitAll()
                auth.requestMatchers("/swagger-ui/**").permitAll()
                auth.requestMatchers("/api-docs/**").permitAll()
                auth.requestMatchers("/uploads/**").permitAll()
                auth.requestMatchers("/images/**").permitAll()


                auth.requestMatchers("/api/**").authenticated()

                auth.anyRequest().authenticated()
            }.headers { headers ->
                headers.frameOptions { it.disable() }
            }
            .sessionManagement { s: SessionManagementConfigurer<HttpSecurity?> ->
                s.sessionCreationPolicy(
                    SessionCreationPolicy.STATELESS
                )
            }
            .oauth2ResourceServer { oauth2:
OAuth2ResourceServerConfigurer<HttpSecurity?> ->
                oauth2.jwt { jwt ->
                    jwt.decoder(
                        jwtDecoder()
                    )
                }
            }
            .userDetailsService(userDetailsService)
            .httpBasic(Customizer.withDefaults())
            .build()
    }

    @Bean
    fun jwtDecoder(): JwtDecoder {
        return
NimbusJwtDecoder.withPublicKey(rsaKeyConfigProperties.publicKey).build()
    }

    @Bean
    fun jwtEncoder(): JwtEncoder {
        val jwk: JWK =
RSAKey.Builder(rsaKeyConfigProperties.publicKey).privateKey(rsaKeyConfigProperties.
privateKey).build()

        val jwks: JWKSource<SecurityContext> = ImmutableJWKSet(JWKSet(jwk))
        return NimbusJwtEncoder(jwks)
    }

    @Bean
    fun passwordEncoder(): PasswordEncoder {
        return BCryptPasswordEncoder()
    }

    @Bean
    fun corsConfigurationSource(): CorsConfigurationSource {
        val source = UrlBasedCorsConfigurationSource()
        val config = CorsConfiguration()
```

```
        if (activeProfile == "dev") {
            config.allowedOrigins = allowedOrigins.split(",")
            config.allowedMethods = listOf("GET", "POST", "PUT", "DELETE",
"OPTIONS", "PATCH", "HEAD")
            config.allowedHeaders = listOf("*")
            config.allowCredentials = true
            source.registerCorsConfiguration("/api/**", config)
        }
        return source
    }


    companion object {

        private val log: Logger =
LoggerFactory.getLogger(SecurityConfig::class.java)
    }
}
```