**import** org.springframework.security.authentication.AuthenticationManager;

@RestController  
@RequestMapping(**"/api"**)  
**public class** UserJWTController {  
 **public** UserJWTController(TokenProvider tokenProvider, AuthenticationManager authenticationManager) {….}  
 @PostMapping(**"/authenticate"**)  
 @Timed  
 **public** ResponseEntity<JWTToken> authorize(@Valid @RequestBody LoginVM loginVM) {  
  
 UsernamePasswordAuthenticationToken authenticationToken =  
 **new** UsernamePasswordAuthenticationToken(loginVM.getUsername(), loginVM.getPassword());  
 Authentication authentication = **this**.**authenticationManager**.authenticate(authenticationToken);  
 SecurityContextHolder.*getContext*().setAuthentication(authentication);  
 **boolean** rememberMe = (loginVM.isRememberMe() == **null**) ? **false** : loginVM.isRememberMe();  
 String jwt = **tokenProvider**.createToken(authentication, rememberMe);  
 HttpHeaders httpHeaders = **new** HttpHeaders();  
 httpHeaders.add(JWTFilter.***AUTHORIZATION\_HEADER***, **"Bearer "** + jwt);  
 **return new** ResponseEntity<>(**new** JWTToken(jwt), httpHeaders, HttpStatus.***OK***);  
 }-------------------------------------------------------------------

@Configuration  
@EnableWebSecurity  
@EnableGlobalMethodSecurity(prePostEnabled = **true**, securedEnabled = **true**)  
@Import(SecurityProblemSupport.**class**)  
**public class** SecurityConfiguration **extends** WebSecurityConfigurerAdapter {  
 **public** SecurityConfiguration(AuthenticationManagerBuilder authenticationManagerBuilder, UserDetailsService userDetailsService, TokenProvider tokenProvider, CorsFilter corsFilter, SecurityProblemSupport problemSupport) {...}  
 @PostConstruct  
 **public void** init() {  
 **try** {  
 **authenticationManagerBuilder**.userDetailsService(**userDetailsService**)  
 .passwordEncoder(passwordEncoder());  
 } **catch** (Exception e) {  
 **throw new** BeanInitializationException(**"Security configuration failed"**, e);  
 }  
 }  
 @Override @Bean  
 **public** AuthenticationManager authenticationManagerBean() **throws** Exception {  
 **return super**.authenticationManagerBean();  
 }  
 @Bean  
 **public** PasswordEncoder passwordEncoder() {  
 **return new** BCryptPasswordEncoder();  
 }@Override  
 **public void** configure(WebSecurity web) **throws** Exception {  
 web.ignoring()  
 .antMatchers(HttpMethod.***OPTIONS***, **"/\*\*"**).antMatchers(**"/app/\*\*/\*.{js,html}"**)  
 .antMatchers(**"/i18n/\*\*"**).antMatchers(**"/content/\*\*"**).antMatchers(**"/h2-console/\*\*"**)  
 .antMatchers(**"/swagger-ui/index.html"**).antMatchers(**"/test/\*\*"**);  
 }@Override  
 **public void** configure(HttpSecurity http) **throws** Exception {  
 http  
 .csrf().disable()  
 .addFilterBefore(**corsFilter**, UsernamePasswordAuthenticationFilter.**class**)  
 .exceptionHandling().authenticationEntryPoint(**problemSupport**)  
 .accessDeniedHandler(**problemSupport**)  
 .and().headers().frameOptions().disable()  
 .and().sessionManagement().sessionCreationPolicy(SessionCreationPolicy.***STATELESS***)  
 .and().authorizeRequests()  
 .antMatchers(**"/api/register"**).permitAll().antMatchers(**"/api/\*\*"**).authenticated()  
 .antMatchers(**"/management/\*\*"**).hasAuthority(AuthoritiesConstants.***ADMIN***)  
 .and().apply(securityConfigurerAdapter());  
 }  
 **private** JWTConfigurer securityConfigurerAdapter() {  
 **return new** JWTConfigurer(**tokenProvider**);  
 }-------------------------------------------------------------------

@Component(**"userDetailsService"**)  
**public class** DomainUserDetailsService **implements** UserDetailsService {  
 **private final** Logger **log** = LoggerFactory.getLogger(DomainUserDetailsService.**class**);  
 **private final** UserRepository **userRepository**;  
 **public** DomainUserDetailsService(UserRepository userRepository) {…}  
 @Override  
 @Transactional  
 **public** UserDetails loadUserByUsername(**final** String login) {  
 **log**.debug(**"Authenticating {}"**, login);  
 **if** (**new** EmailValidator().isValid(login, **null**)) {  
 **return userRepository**.findOneWithAuthoritiesByEmail(login)  
 .map(user -> createSpringSecurityUser(login, user))  
 .orElseThrow(() -> **new** UsernameNotFoundException(**"User with email "** + login + **" was not found in the database"**));  
 }  
 String lowercaseLogin = login.toLowerCase(Locale.ENGLISH);  
 **return userRepository**.findOneWithAuthoritiesByLogin(lowercaseLogin)  
 .map(user -> createSpringSecurityUser(lowercaseLogin, user))  
 .orElseThrow(() -> **new** UsernameNotFoundException(**"User "** + lowercaseLogin + **" was not found in the database"**));  
 }  
 **private** org.springframework.security.core.userdetails.User createSpringSecurityUser(String lowercaseLogin, User user) {  
 **if** (!user.getActivated()) {  
 **throw new** UserNotActivatedException(**"User "** + lowercaseLogin + **" was not activated"**);  
 }  
 List<GrantedAuthority> grantedAuthorities = user.getAuthorities().stream()  
 .map(authority -> **new** SimpleGrantedAuthority(authority.getName()))  
 .collect(Collectors.toList());  
 **return new** org.springframework.security.core.userdetails.User(user.getLogin(),  
 user.getPassword(),  
 grantedAuthorities);  
 }  
}---------------------------------------

**public class** JWTFilter **extends** GenericFilterBean {  
  
 **public static final** String ***AUTHORIZATION\_HEADER*** = **"Authorization"**;  
 **private** TokenProvider **tokenProvider**;  
 **public** JWTFilter(TokenProvider tokenProvider) {  
 **this**.**tokenProvider** = tokenProvider;  
 }  
  
 @Override  
 **public void** doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain)  
 **throws** IOException, ServletException {  
 HttpServletRequest httpServletRequest = (HttpServletRequest) servletRequest;  
 String jwt = resolveToken(httpServletRequest);  
 **if** (StringUtils.*hasText*(jwt) && **this**.**tokenProvider**.validateToken(jwt)) {  
 Authentication authentication = **this**.**tokenProvider**.getAuthentication(jwt);  
 SecurityContextHolder.*getContext*().setAuthentication(authentication);  
 }  
 filterChain.doFilter(servletRequest, servletResponse);  
 }  
  
 **private** String resolveToken(HttpServletRequest request){  
 String bearerToken = request.getHeader(***AUTHORIZATION\_HEADER***);  
 **if** (StringUtils.*hasText*(bearerToken) && bearerToken.startsWith(**"Bearer "**)) {  
 **return** bearerToken.substring(7);  
 }  
 **return null**;  
 }-------------------------------------------

@Component  
**public class** TokenProvider {  
 **private static final** String ***AUTHORITIES\_KEY*** = **"auth"**;**private** Key **key**;  
 **private long tokenValidityInMilliseconds**;**private long tokenValidityInMillisecondsForRememberMe**;  
 **private final** JHipsterProperties **jHipsterProperties**;  
 **public** TokenProvider(JHipsterProperties jHipsterProperties) {..}  
 @PostConstruct  
 **public void** init() {  
 **byte**[] keyBytes;  
 String secret = **jHipsterProperties**.getSecurity().getAuthentication().getJwt().getSecret();  
 **if** (!StringUtils.*isEmpty*(secret)) {  
 **log**.warn(**"Warning: the JWT key used is not Base64-encoded. "** );  
 keyBytes = secret.getBytes(StandardCharsets.UTF\_8);  
 } **else** {  
 **log**.debug(**"Using a Base64-encoded JWT secret key"**);  
 keyBytes = Decoders.***BASE64***.decode(**jHipsterProperties**.getSecurity().getAuthentication().getJwt().getBase64Secret());  
 }  
 **this**.**key** = Keys.*hmacShaKeyFor*(keyBytes);  
 **this**.**tokenValidityInMilliseconds** =  
 1000 \* **jHipsterProperties**.getSecurity().getAuthentication().getJwt().getTokenValidityInSeconds();  
 **this**.**tokenValidityInMillisecondsForRememberMe** =  
 1000 \* **jHipsterProperties**.getSecurity().getAuthentication().getJwt()  
 .getTokenValidityInSecondsForRememberMe();  
 }  
  
 **public** String createToken(Authentication authentication, **boolean** rememberMe) {  
 String authorities = authentication.getAuthorities().stream()  
 .map(GrantedAuthority::getAuthority)  
 .collect(Collectors.joining(**","**));  
 **long** now = (**new** Date()).getTime();  
 Date validity;  
 **if** (rememberMe) {  
 validity = **new** Date(now + **this**.**tokenValidityInMillisecondsForRememberMe**);  
 } **else** {  
 validity = **new** Date(now + **this**.**tokenValidityInMilliseconds**);  
 }  
 **return** Jwts.*builder*()  
 .setSubject(authentication.getName())  
 .claim(***AUTHORITIES\_KEY***, authorities)  
 .signWith(**key**, SignatureAlgorithm.***HS512***)  
 .setExpiration(validity)  
 .compact();  
 }  
 **public** Authentication getAuthentication(String token) {  
 Claims claims = Jwts.*parser*()  
 .setSigningKey(**key**).parseClaimsJws(token).getBody();  
 Collection<? **extends** GrantedAuthority> authorities =  
 Arrays.stream(claims.get(***AUTHORITIES\_KEY***).toString().split(**","**))  
 .map(SimpleGrantedAuthority::**new**)  
 .collect(Collectors.toList());  
 User principal = **new** User(claims.getSubject(), **""**, authorities);  
 **return new** UsernamePasswordAuthenticationToken(principal, token, authorities);  
 }  
 **public boolean** validateToken(String authToken) {  
 **try** {  
 Jwts.*parser*().setSigningKey(**key**).parseClaimsJws(authToken);  
 **return true**;  
 } **catch** (io.jsonwebtoken.security.SecurityException | MalformedJwtException e) {  
 **log**.info(**"Invalid JWT signature."**);  
 } **catch** (ExpiredJwtException e) {**log**.info(**"Expired JWT token."**);  
 } **catch** (UnsupportedJwtException e) {**log**.info(**"Unsupported JWT token."**);  
 } **catch** (IllegalArgumentException e) {**log**.info(**"JWT token compactinvalid."**);  
 }  
 **return false**;  
 }  
}------------------------------------------------------------------------

**public class** JWTConfigurer **extends** SecurityConfigurerAdapter<DefaultSecurityFilterChain, HttpSecurity> {  
 **private** TokenProvider **tokenProvider**;  
 **public** JWTConfigurer(TokenProvider tokenProvider) {}  
 @Override  
 **public void** configure(HttpSecurity http) **throws** Exception {  
 JWTFilter customFilter = **new** JWTFilter(**tokenProvider**);  
 http.addFilterBefore(customFilter, UsernamePasswordAuthenticationFilter.**class**);  
 }