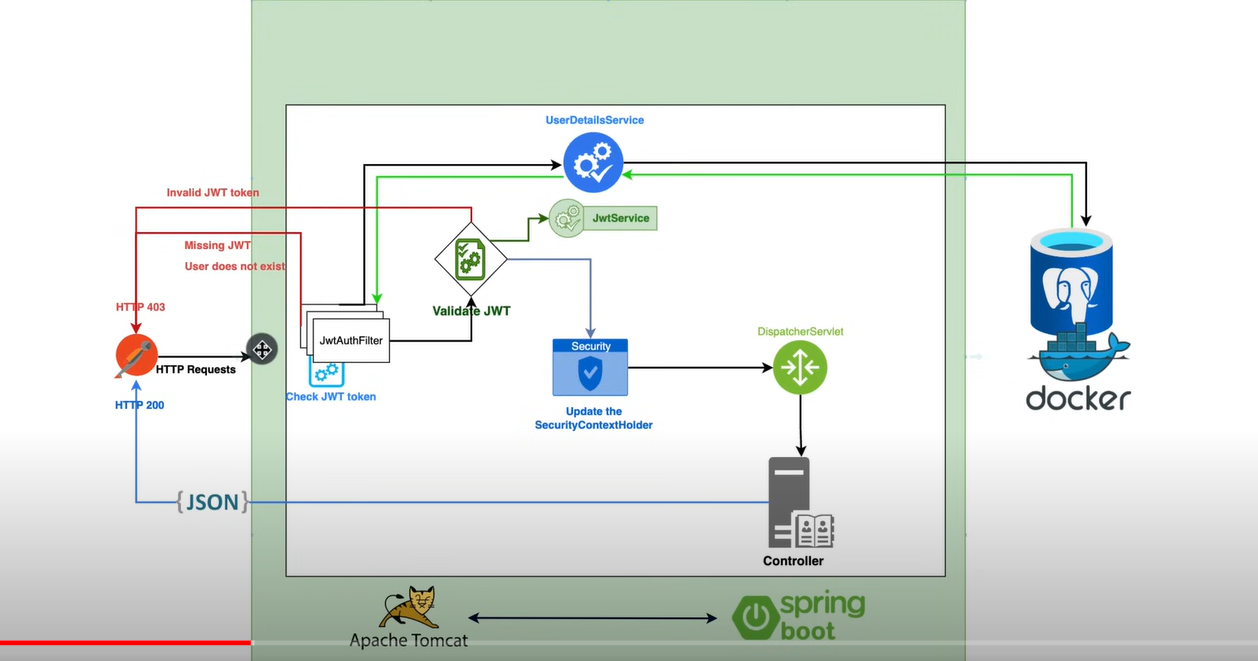
JWT implementation in Spring boot :



1. The firstly we have to make a filter JwtAuthenticationFilter which will extend OncePerRequestFilter.

Code :

@Override  
 protected void doFilterInternal( HttpServletRequest request, HttpServletResponse response, FilterChain filterChain) throws ServletException, IOException, IOException {  
  
 final String auth = request.getHeader("Authorization");  
 String jwt;  
 System.*out*.println("The value of auth is " + auth);  
 if (auth == null || !auth.startsWith("Bearer ")) {  
 filterChain.doFilter(request, response);  
 return;  
 }  
 jwt = auth.substring(7);  
 String username = jwtService.extractUsername(jwt);  
 System.*out*.println("The value of username is " + username);  
 if (username != null && SecurityContextHolder.*getContext*().getAuthentication() == null) {  
 UserDetails userDetails = userservice.loadUserByUsername(username);  
 if (jwtService.isTokenValid(jwt, userDetails)) {  
 System.*out*.println("This logger will be printed when the token is valid for username " + username);  
 // IMPORTANT in the object below we have to mention credentials as null otherwise will get 403.  
 UsernamePasswordAuthenticationToken usernamePasswordAuthenticationToken = new UsernamePasswordAuthenticationToken(  
 userDetails,  
 null,  
 userDetails.getAuthorities()  
 );  
 usernamePasswordAuthenticationToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));  
 SecurityContextHolder.*getContext*().setAuthentication(usernamePasswordAuthenticationToken);  
 }  
 }  
 // hello world  
 filterChain.doFilter(request, response);  
 }  
}

1. In this code we see wether the jwt token is valid for not.if valid then we se the SecurityContextHolderConfiguration with the request and the payload details.
2. For all of this we have to make jwtservice which is responsible for extracting information from token as well as generating new token.
3. One more thing we need is AuthenticationProvider asa well as AuthenticationManager.

Code :

@Bean  
public AuthenticationProvider getAuthenticationProvider() {  
 DaoAuthenticationProvider daoAuthenticationProvider = new DaoAuthenticationProvider();  
 daoAuthenticationProvider.setUserDetailsService(userService);  
 daoAuthenticationProvider.setPasswordEncoder(getPasswordEncoder());  
 return daoAuthenticationProvider;  
}  
  
@Bean  
public AuthenticationManager getAuthenticationManager(@Autowired AuthenticationConfiguration authenticationConfiguration) throws Exception {  
 return authenticationConfiguration.getAuthenticationManager();  
}  
  
@Bean  
public PasswordEncoder getPasswordEncoder() {  
 return new BCryptPasswordEncoder();  
}

1. Now we can bind this together using ConfigurationClass

Code:

package com.security.jwtservice.config;  
  
import lombok.RequiredArgsConstructor;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.security.authentication.AuthenticationProvider;  
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.web.SecurityFilterChain;  
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;  
import org.springframework.security.web.util.matcher.AntPathRequestMatcher;  
  
@Configuration  
@EnableWebSecurity  
@RequiredArgsConstructor  
public class SecurityConfiguration {  
  
 private final AuthenticationProvider autheniticationProvider;  
 private final JWTAuthenticationFilter jwtAuthenticationFilter;  
  
 @Bean  
 public SecurityFilterChain getSecurityFilterChain(HttpSecurity http) throws Exception {  
 http  
 .csrf()  
 .disable()  
 .authorizeHttpRequests()  
 .requestMatchers(new AntPathRequestMatcher("/v1/auth/\*\*"))  
 .permitAll().anyRequest().authenticated().and()  
 .sessionManagement()  
 .sessionCreationPolicy(SessionCreationPolicy.*STATELESS*)  
 .and().authenticationProvider(autheniticationProvider)  
 .addFilterBefore(jwtAuthenticationFilter, UsernamePasswordAuthenticationFilter.class);  
 return http.build();  
 }  
}

1. Here basically we are filtering the requests for which filters should be applied and for which requests filter should not be applied (register(sign up), authenticate(sign in))
2. Then apply sessioncorrection policy
3. Then we apply Authenticaton Provider which we created above.
4. Then we apply JWTAuthenticationtoken which we make at the first step before UsernamePasswordAunthenticationFilter.class.
5. This Configuration class is @EnableWebSecurity.

Other implementation (AuthenticationService) :

This service is for register process and authentication process which will generate the jwt token.

package com.security.jwtservice;  
  
import lombok.RequiredArgsConstructor;  
import org.springframework.http.ResponseEntity;  
import org.springframework.security.authentication.AuthenticationManager;  
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;  
import org.springframework.security.core.userdetails.UserDetails;  
import org.springframework.security.crypto.password.PasswordEncoder;  
import org.springframework.stereotype.Service;  
  
@Service  
@RequiredArgsConstructor  
public class AuthenticationService {  
  
 private final JwtService jwtService;  
 private final UsesRepository userRepository;  
 private final AuthenticationManager authManager;  
 private final PasswordEncoder passwordEncoder;  
 public ResponseEntity<AutthenticationResponse> register(RegisterRequest registerRequest) {  
 User user = new User();  
 user.setFirstname(registerRequest.getFirstName());  
 user.setLastname(registerRequest.getLastName());  
 user.setPassword(passwordEncoder.encode(registerRequest.getPassword()));  
 user.setUsername(registerRequest.getUsername());  
 user.setEmailId(registerRequest.getEmailId());  
 user.setRole(Role.*USER*);  
 userRepository.save(user);  
 String jwtToken = jwtService.generateTokenOnlyWithUserDetails(user);  
 AutthenticationResponse authenticationResponse = new AutthenticationResponse();  
 authenticationResponse.setToken(jwtToken);  
 return ResponseEntity.*ok*(authenticationResponse);  
 }  
  
 public ResponseEntity<AutthenticationResponse> authenticate(AuthenticationRequest authenticationRequest) {  
 authManager.authenticate(  
 new UsernamePasswordAuthenticationToken(  
 authenticationRequest.getUsername(),  
 authenticationRequest.getPassword()  
 )  
 );  
 UserDetails userDetails = userRepository.findByUserName(authenticationRequest.getUsername()).orElseThrow(() -> new RuntimeException("Object not found in the database !"));  
 String jwtToken = jwtService.generateTokenOnlyWithUserDetails(userDetails);  
 AutthenticationResponse authenticationResponse = new AutthenticationResponse();  
 authenticationResponse.setToken(jwtToken);  
 return ResponseEntity.*ok*(authenticationResponse);  
 }  
}