* Khi ta gửi 1 request lên server (có spring security) nó sẽ đi qua fillter
* @Override  
  protected void configure(HttpSecurity http) throws Exception {  
   http  
   .cors()  
   .and()  
   .sessionManagement()  
   .sessionCreationPolicy(SessionCreationPolicy.*STATELESS*)  
   .and()  
   .csrf()  
   .disable()  
   .authorizeRequests()  
   .antMatchers("/api/user/login").permitAll()// Cho phép tất cả mọi người truy cập vào địa chỉ này  
   .anyRequest().authenticated();  
    
   // Thêm một lớp Filter kiểm tra jwt  
   http.addFilterBefore(jwtAuthenticationFilter(), UsernamePasswordAuthenticationFilter.class);  
    
  }

ở đây ta tạo 1 tự custom 1class fillter và cho nó thứ tự ưu tiên là đầu tiên

public class JwtAuthenticationFilter extends OncePerRequestFilter {  
 @Autowired  
 private JwtTokenProvider tokenProvider;  
  
 @Autowired  
 private CustomUserDetailService customUserDetailsService;  
 @Override  
 protected void doFilterInternal(HttpServletRequest request,  
 HttpServletResponse response, FilterChain filterChain)  
 throws ServletException, IOException {  
 try {  
 String jwt = getJwtFromRequest(request);  
  
 if (StringUtils.*hasText*(jwt) && tokenProvider.validateToken(jwt)) {  
 String userId = tokenProvider.getUserIdFromJWT(jwt);  
  
 UserDetails userDetails = customUserDetailsService.loadUserById(userId);  
 if(userDetails != null) {  
 UsernamePasswordAuthenticationToken  
 authentication = new UsernamePasswordAuthenticationToken(userDetails, null,  
 userDetails  
 .getAuthorities());  
 authentication.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));  
  
 SecurityContextHolder.*getContext*().setAuthentication(authentication);  
 }  
 }  
 } catch (Exception ex) {  
 *log*.error("failed on set user authentication", ex);  
 }  
  
 filterChain.doFilter(request, response);  
 }  
  
 private String getJwtFromRequest(HttpServletRequest request) {  
 String bearerToken = request.getHeader("Authorization");  
 // Kiểm tra xem header Authorization có chứa thông tin jwt không  
 if (StringUtils.*hasText*(bearerToken) && bearerToken.startsWith("Bearer ")) {  
 return bearerToken.substring(7);  
 }  
 return null;  
 }  
}

Hàm doFilterInternal có nhiệm vụ kiểm tra header của request gửi lên có JWT hay không, JWT có hợp lệ không. Nếu không hợp lệ nó sẽ đi vào những fillter tiếp theo. Nếu JWT hợp lệ nó sẽ parse token và lấy ra userID từ token và gọi hàm customUserDetailsService.loadUserById(userId) để trả về 1 CustomUserDetail

Ta sẽ tạo Đối tượng Athencation từ CustomUserDetail này(mà không cần phải authen) và lưu vào trong SecurityContextHolder để Spring security quản lí -> lúc này spring đã biết ai là người gửi request và có những quyền gì.

* Trường hợp mà request gửi lên không có JWT or JWT không hợp lệ nó sẽ đi qua lần lượt các filter. Nếu request mà đến 1 URL yêu cầu xác thực hoặc cả xác thực lẫn phân quyền nó sẽ trả về 1 lỗi
* ở đây ta cho phép .antMatchers("/api/user/login").permitAll()
* @PostMapping("/login")  
  public LoginResponse authenticateUser(@RequestBody LoginRequest loginRequest) {  
    
   // Xác thực thông tin người dùng Request lên  
   Authentication authentication = authenticationManager.authenticate(  
   new UsernamePasswordAuthenticationToken(  
   loginRequest.getUsername(),  
   loginRequest.getPassword()  
   )  
   );  
   // Nếu không xảy ra exception tức là thông tin hợp lệ  
   // Set thông tin authentication vào Security Context  
   SecurityContextHolder.*getContext*().setAuthentication(authentication);  
   // trả về jwt cho người dùng  
   String jwt = tokenProvider.generateToken((CustomUserDetails) authentication.getPrincipal());  
   return new LoginResponse(jwt);  
    
  }

ở đây ta gọi hàm Authenticate của interface AuthenticationManager(mặc định impliment sẽ là DaoAuthenticationProvider ) để authen username password, hàm này nhận vào Authentication và trả về 1 Authentication

* Phương thức authenticate sẽ sử dụng đối tượng UserDetailService gọi hàm
* UserDetails loadUserByUsername(String userName)

Với userName của request gửi lên để trả về 1 UserDetail.

Ở đây ta đã tạo class CustomUserDetailService implement interface UserDetailService và override lại hàm loadUserByUsername() theo ý của mình (giả sử muốn login theo userName, theo email) và trả về 1 CustomUserDetail

@Override  
public UserDetails loadUserByUsername(String userName) throws UsernameNotFoundException {  
 Optional<UserEntity> optionalUserEntity = userRepository.findByUserName(userName);  
 if (!optionalUserEntity.isPresent()) {  
 throw new UsernameNotFoundException("Login fail! User name is not exist");  
 } else {  
 UserEntity userEntity = optionalUserEntity.get();  
 CustomUserDetails customUserDetails = new CustomUserDetails(userEntity);  
 return customUserDetails;  
 }  
}

Spring sẽ sử dụng đối UserDetail trả về để so sánh với mật khẩu người dùng gửi lên

Nếu sai hàm authenticate ném ra 1 exception, nếu đúng sẽ trả về 1 Authentication chứa thông tin UserDetail ( ở đây ta đã tự custom interface UserDetail)

Tiếp theo ta sẽ ném cái authentication vừa trả ra từ authenticate cho thằng security quản lí

SecurityContextHolder.*getContext*().setAuthentication(authentication);

Tiếp theo ta sẽ tạo Jwt từ đối tượng authentication trả về từ hàm authenticate và trả về cho client

String jwt = tokenProvider.generateToken((CustomUserDetails) authentication.getPrincipal());  
return new LoginResponse(jwt);

Mỗi lần client gửi request sẽ gửi kèm cái JWT, Server sẽ giải mã token và biết người gửi request là ai và có quyền gì.