Spring Security Documentation :

Step 1 : Add the Following Dependency :

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

<!-- https://mvnrepository.com/artifact/io.jsonwebtoken/jjwt -->

<dependency>

<groupId>io.jsonwebtoken</groupId>

<artifactId>jjwt</artifactId>

<version>0.9.1</version>

</dependency>

<!-- https://mvnrepository.com/artifact/javax.xml.bind/jaxb-api -->

<dependency>

<groupId>javax.xml.bind</groupId>

<artifactId>jaxb-api</artifactId>

<version>2.3.1</version>

</dependency>

Step 2 : class Implements UserDetails

Step 3 : Creating the Authority Class :

**package** com.example.demo.entity;

**import** org.springframework.security.core.GrantedAuthority;

**public** **class** Authority **implements** GrantedAuthority {

/\*\*

\*

\*/

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**private** String authority;

**public** Authority(String authority) {

**super**();

**this**.authority = authority;

}

**public** Authority() {

**super**();

// **TODO** Auto-generated constructor stub

}

@Override

**public** String getAuthority() {

// **TODO** Auto-generated method stub

**return** **this**.authority;

}

**public** **void** setAuthority(String authority) {

**this**.authority = authority;

}

}

Step 4 : in the grant Authority function we will assign authority to the user with the help of the authority class which we have created

@Override

**public** Collection<? **extends** GrantedAuthority> getAuthorities() {

Set<Authority> st = **new** HashSet<>();

// adding authority for the user using the userRole

**this**.userRoles.forEach(ur -> {

st.add( **new** Authority(ur.getRole().getRoleName()));

});

**return** **null**;

}

Step 5 : Now we will Create Two More Classes JwtReq , JwtRes

jwtRequest

**package** com.example.demo.entity;

**public** **class** JwtRequest

{

String username;

String password;

**public** String getUsername() {

**return** username;

}

**public** **void** setUsername(String username) {

**this**.username = username;

}

**public** String getPassword() {

**return** password;

}

**public** **void** setPassword(String password) {

**this**.password = password;

}

**public** JwtRequest() {

**super**();

// **TODO** Auto-generated constructor stub

}

**public** JwtRequest(String username, String password) {

**super**();

**this**.username = username;

**this**.password = password;

}

}

jwtResponce

**package** com.example.demo.entity;

**public** **class** JwtResponce {

String token;

**public** JwtResponce() {

**super**();

// **TODO** Auto-generated constructor stub

}

**public** JwtResponce(String token) {

**super**();

**this**.token = token;

}

**public** String getToken() {

**return** token;

}

**public** **void** setToken(String token) {

**this**.token = token;

}

}

Step 6 : we will create a serviceImpl that implements UserDetailsService which will return us the user

**package** com.example.demo.services.impl;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.security.core.userdetails.UserDetails;

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

**import** org.springframework.security.core.userdetails.UsernameNotFoundException;

**import** com.example.demo.entity.User;

**import** com.example.demo.repository.UserRepository;

@Service

**public** **class** UserDetailsServiceImpl **implements** UserDetailsService {

@Autowired

**private** UserRepository userRepository;

@Override

**public** UserDetails loadUserByUsername(String username) **throws** UsernameNotFoundException {

User user = **this**.userRepository.findByUsername(username);

**if**(user == **null**) {

System.***out***.println("User not found!");

**throw** **new** UsernameNotFoundException("No User Found!!");

}

**return** user;

}

}

Step 7 : Now we will make the configuration Class :

This class will have

@Configuration

@EnableWebSecurity

@EnableGlobalMethodSecurity

And it will extend the WebSecurityConfigureAdapter class

We will override two methods

Configure(AuthentictionManager)

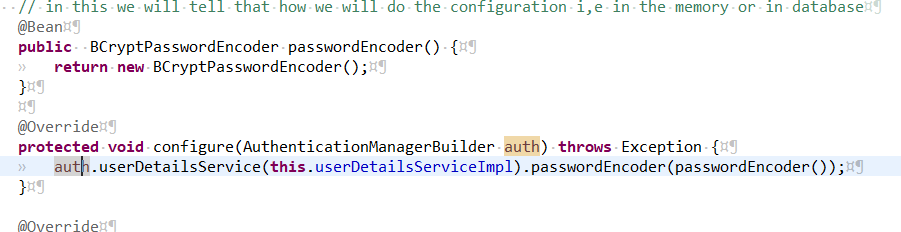
And Configure(http)

We will override two methods

Configure(AuthentictionManager)

And Configure(http)

After this we will inject the UserDetailsServiceImpl class in the auth.userDetailsService and also will use the password encoder

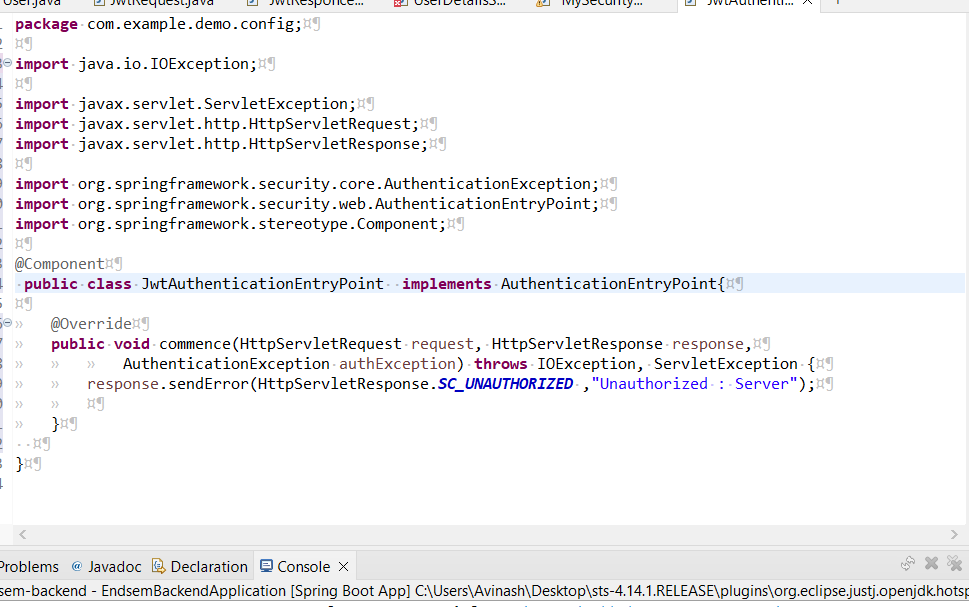


Now we will configure the next method :



Step 8: Now we will Create the classes for exception Handling and for the filtering

Jwt Authentication



JwtFIter : this will extend the OncePerUserFilter

We will also create the jwtUtil class and inject it for the operations on the token

**package** com.example.demo.config;

**import** java.io.IOException;

**import** javax.servlet.FilterChain;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**import** org.springframework.beans.factory.annotation.Autowired;

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

**import** org.springframework.security.core.context.SecurityContextHolder;

**import** org.springframework.security.core.userdetails.UserDetails;

**import** org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

**import** org.springframework.security.web.authentication.WebAuthenticationDetailsSource;

**import** org.springframework.stereotype.Component;

**import** org.springframework.web.filter.OncePerRequestFilter;

**import** com.example.demo.services.impl.UserDetailsServiceImpl;

**import** io.jsonwebtoken.ExpiredJwtException;

@Component

**public** **class** JwtAuthenticationFilter **extends** OncePerRequestFilter {

@Autowired

**private** UserDetailsServiceImpl userDetalsService;

@Autowired

**private** JwtUtil jwtUtil;

@Override

**protected** **void** doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)

**throws** ServletException, IOException

{

**final** String reqTokenHeader = request.getHeader("Authorization");

String username=**null**;

String jwtToken = **null**;

// first we will get the token

**if**(reqTokenHeader!=**null** && reqTokenHeader.startsWith("Bearer "))

{

jwtToken = reqTokenHeader.substring(7);

**try** {

username = **this**.jwtUtil.extractUsername(jwtToken);

} **catch** (ExpiredJwtException e) {

// **TODO**: handle exception

e.printStackTrace();

System.***out***.println("token has expired");

}

}

**else**

{

System.***out***.println("Invalid Token!");

}

// Token Valdiation :

**if**(username!=**null** && SecurityContextHolder.*getContext*().getAuthentication() == **null**) {

**final** UserDetails ud = **this**.userDetalsService.loadUserByUsername(username);

**if**(**this**.jwtUtil.validateToken(jwtToken, ud)) {

UsernamePasswordAuthenticationToken up= **new** UsernamePasswordAuthenticationToken(ud,**null**,ud.getAuthorities());

up.setDetails(**new** WebAuthenticationDetailsSource().buildDetails(request));

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

}

}

**else**

{

System.***out***.println("Invalid Token");

}

filterChain.doFilter(request, response);

}

}

Step 9 : Now we will create our AuthentcationController and In that we will generate-token