# Spring Security with JWT Authentication

1. Initialize the project, setup your database and the project structure (Entity,Service and Repository).
2. Import the spring security dependency.
3. Implementing the csrf token
4. Implement custom username and password.
5. Adding a custom Security configuration.
6. Adding User and Product (Entity,Service and Repository) GET and POST call
7. Implement a login api to check the credentials (locally)
8. WE had to have the userDetails…Thereby implementing custom CustomUserDetailsService to fetch users from our database.
9. Now comes the authentication provider..for this project we would be using DaoAuthenticationProvider
10. Encode the password before saving them to DB.
11. VOLLAH!! Authentication is Done  
      
    11. Now comes JWT token retrieval after successful login.
12. Implement the Authentication Manager Method.
13. Load the three JJWT Dependency--------- JJWT-API, JJWT-IMPL, JJWT- JACKSON
14. Import a base64 jwt secret key in application.properies
15. For jwt token create jwtService and a method to retreive JWT Token
16. Now with the help of generated JWT TOKEN….bypass the UserNameAndPasswordAuthentication in the filter chain.
17. For the above we would be needing a JWTAuthenticationFilter.. Obviouslusly!!!

It will extend OncePerRequestFilter which have one method doFilterInternal(request,resposnse,filterChain)

**Documentation**

3.1) Now when GET request is made with authentication it dosen’t need csrf token..It can be overlooked but when changing the information(POST,PUT,DELETE,UPDATE) csrf token is mandatory!!!

@GetMapping("/csrf") //this method will get me the csrf token for the already authenticated user  
 //This api will only hity when the user is authenticated.  
public CsrfToken getCSRFToken(HttpServletRequest request)  
{  
 return (CsrfToken) request.getAttribute("\_csrf");  
}

This will return the below thing

{

    "parameterName": "\_csrf",

    "token": "\_YTV9n\_BvMda1oqdkoLcVFxztpNx5GLOQJY9X9JVRp7\_RCcGm7zmzk\_2j\_F347n\_pq\_obD9Lm6pI1QbjcaAEbuJtcfzMdURl",

    "headerName": "X-CSRF-TOKEN"

}

Now put the token and headername in header section of the post call

4.1)   
spring.security.user.name=admin  
spring.security.user.password: 123

5.1) Now till now we were using deafult configuratuion but now we will be defineing our custom file

@Configuration  
@EnableWebSecurity  
public class SecurityConfig  
{  
 @Bean  
 public SecurityFilterChain securityFilterChain(HttpSecurity httpSecurity) throws Exception {  
 return httpSecurity.build();  
 }  
}

Like right now the filterchain is empty so it dosent have any authentication right now.  
5.2) Adding further configurations…..  
@Bean  
public SecurityFilterChain securityFilterChain(HttpSecurity httpSecurity) throws Exception {  
 httpSecurity  
 .csrf(csrf -> csrf.disable()) // disableing the csrf token  
 .authorizeHttpRequests( // this will tell it to authenticate all the requests before passing  
 request-> request.anyRequest().authenticated()  
 )  
 .formLogin(Customizer.*withDefaults*()) // will give me a basic form  
 .httpBasic(Customizer.*withDefaults*()); // adding basic authentication with a popup form if form login is not mentioned  
 return httpSecurity.build();  
}

7.1) @PostMapping("/login")  
public String loginFailOrPass(@RequestBody User user)  
{  
 String res="Login failed";  
 if (userRepo.existsByUsername(user.getUsername()))  
 {  
 User dbUser= userRepo.findByUsername(user.getUsername());  
 if(dbUser.getPassword().equals(user.getPassword()))  
 res="Login Success and password matches as well";  
 else  
 res="Login Success with username only";  
 }  
  
 return res;  
}

8.3) Ha  
8.1) CustomUserDetailService implement UserDetailsService   
8.2) Had to override loadUserByUsername method  
public class CustomUserDetailsService implements UserDetailsService {

private final UserRepo userRepo;

public CustomUserDetailsService(UserRepo userRepo)

{

this.userRepo = userRepo;

}

@Override

public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException

{

User user=userRepo.findByUsername(username);

if(Objects.isNull(user)) // similar to if(user==null)

{

System.out.println("User not available");

throw new UsernameNotFoundException("User not found");

}

return new CustomUserDetails(user);

} // now this method returns UserDetails therby we implement CutomUserDetails seperataly

}

8.3) CutomUserDetails

public class CustomUserDetails implements UserDetails {  
 private final User user;  
 public CustomUserDetails(User user) {  
 this.user = user;  
 }  
 @Override  
 public Collection<? extends GrantedAuthority> getAuthorities() {  
 return Collections.*singleton*(new SimpleGrantedAuthority("USER"));  
 }  
  
 @Override  
 public String getPassword() {  
 return user.getPassword();  
 }  
 @Override  
 public String getUsername() {  
 return user.getUsername();  
 }  
 @Override  
 public boolean isAccountNonExpired() {  
 return true;  
 }  
 @Override  
 public boolean isAccountNonLocked() {  
 return true;  
 }  
 @Override  
 public boolean isCredentialsNonExpired() {  
 return true;  
 }  
 @Override  
 public boolean isEnabled() {  
 return true;  
 }  
}

9.1) @Bean  
public AuthenticationProvider authenticationProvider()  
{  
 DaoAuthenticationProvider daoAuthenticationProvider=new DaoAuthenticationProvider();  
 daoAuthenticationProvider.setUserDetailsService(userDetailsService);  
 daoAuthenticationProvider.setPasswordEncoder(passwordEncoder());  
 return daoAuthenticationProvider;  
  
}

9.2) had to have a Password encoder  
@Bean  
public PasswordEncoder passwordEncoder()  
{  
 return new BCryptPasswordEncoder();  
}

12.1)

@Bean  
public AuthenticationManager authenticationManager(AuthenticationConfiguration configuration) throws Exception {  
 return configuration.getAuthenticationManager();  
}

12.2)Now we can use this authenticationManger to authenticate our user…. we don’t have to do it the hard coded way as below

@PostMapping("/login") //Without Authentication manager  
 public String loginFailOrPass(@RequestBody User user)  
 {  
 String res="";  
 if (userRepo.existsByUsername(user.getUsername()))  
 {  
 User dbUser= userRepo.findByUsername(user.getUsername());  
 if(passwordEncoder.matches(user.getPassword(),dbUser.getPassword()))  
 res="Login Success and password matches as well";  
 else  
 res="Login Failed... Enter the correct password!";  
 }  
 else {  
 res="No such user with username = "+ user.getUsername()+" is present";  
 }  
 return res;  
 }

12.3) Login Authentication with Authentication Manger:

@PostMapping("/login")  
public String loginFailOrPass(@RequestBody User user)  
{  
 Authentication authenticate = authenticationManager  
 .authenticate(new UsernamePasswordAuthenticationToken(user.getUsername(), user.getPassword()));  
 if(authenticate.isAuthenticated())  
 {  
 //*TODO: Return JWT Token* }  
}

13.1)

<!-- https://mvnrepository.com/artifact/io.jsonwebtoken/jjwt-jackson -->  
<dependency>  
 <groupId>io.jsonwebtoken</groupId>  
 <artifactId>jjwt-jackson</artifactId>  
 <version>0.12.6</version>  
 <scope>runtime</scope>  
</dependency>  
<!-- https://mvnrepository.com/artifact/io.jsonwebtoken/jjwt-impl -->  
<dependency>  
 <groupId>io.jsonwebtoken</groupId>  
 <artifactId>jjwt-impl</artifactId>  
 <version>0.12.6</version>  
 <scope>runtime</scope>  
</dependency>  
<!-- https://mvnrepository.com/artifact/io.jsonwebtoken/jjwt-api -->  
<dependency>  
 <groupId>io.jsonwebtoken</groupId>  
 <artifactId>jjwt-api</artifactId>  
 <version>0.12.6</version>  
</dependency>

14.1) For a Jwt Token….It has three sections Header,Payload and verifying Signature.  
  
14.2) We need Jwts to build the jwt token.

@Override  
public String getJwtToken(User user) //generates a JWT (JSON Web Token) for a given User  
{  
 Map<String,Object> claims=new HashMap<>();  
 return Jwts.*builder*() // This creates a new JWT builder instance using jjwt (JSON Web Token library), initializes a JWT object, allowing us to set various fields  
 .claims() // initializes the claims section of the JWT.  
 .add(claims)  
 .subject(user.getUsername())  
 .issuer("HANEEF") //Helps in tracking **which service generated** the JWT.  
 .issuedAt(new Date(System.*currentTimeMillis*()))  
 .expiration(new Date(System.*currentTimeMillis*()+60\*10\*1000))  
 .and()  
 .signWith(generateKey()) //Signing ensures the token cannot be tampered with,Any modification to the JWT will make it invalid during verification

.compact(); //Generates the final compact JWT string.  
}

14.3) We define claims for the payload section…….**Why use claims?**

* Claims store key-value pairs inside the JWT.
* They can include **metadata** like role, permissions, or any other user-specific info.

Example:

java

CopyEdit

claims.put("role", user.getRole());

claims.put("email", user.getEmail());

14.4)  
private SecretKey generateKey()  
{  
 byte[] decode= Decoders.*BASE64*.decode(secretKey);  
 return Keys.*hmacShaKeyFor*(decode);  
}

**Why use** HMAC-SHA **instead of other algorithms?**

* **HMAC (Hash-based Message Authentication Code)** is widely used for JWT signing.
* It's **fast** and does not require a public/private key pair like RSA.
* If **stronger security** is needed, **RSA or EC** could be used.

| **Code** | **Purpose** |
| --- | --- |

|  |  |
| --- | --- |
| Map<String, Object> claims = new HashMap<>(); | Stores additional metadata (optional) |

|  |  |
| --- | --- |
| .claims().add(claims) | Adds claims to the token |

|  |  |
| --- | --- |
| .subject(user.getUsername()) | Identifies the user |

|  |  |
| --- | --- |
| .issuer("HANEEF") | Specifies who issued the token |

|  |  |
| --- | --- |
| .issuedAt(new Date(System.currentTimeMillis())) | Sets token creation time |

|  |  |
| --- | --- |
| .expiration(new Date(System.currentTimeMillis() + 60 \* 10 \* 1000)) | Sets expiration (10 min) |

|  |  |
| --- | --- |
| .signWith(generateKey()) | Signs the token securely |

|  |  |
| --- | --- |
| .compact() | Generates the final JWT string |

|  |  |
| --- | --- |
| generateKey() | Decodes the Base64 secret key for signing |

15.1)   
@Value("${jwt.secret}") //Application properties me jwt.secret define karna padega  
private String secretKey;

15.2) @Value("${jwt.secret}") //Application properties me jwt.secret define karna padega  
private String secretKey;  
/\*  
Note: For a Safer Method.....USE ENVIRONMENT VARIABLE:  
1. Set an environment variable (JWT\_SECRET) in your system:  
set JWT\_SECRET=YourSuperSecretKeyHere in terminalL  
2. Access it in application.properties  
jwt.secret=${JWT\_SECRET} in application.properties  
\*/

17.1)

@Override  
protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain) throws ServletException, IOException {  
 final String authHeader=request.getHeader("Authorization");  
 if(authHeader== null || !authHeader.startsWith("Bearer")) //Agar phle se autheticated nahi hoga iska matlab abhi tak jwt token nahi generate hua hoga  
 {  
 filterChain.doFilter(request,response);  
 return;  
 }  
 final String jwt=authHeader.substring(7);  
 final String username= jwtService.ectractUserName(jwt);  
 Authentication authentication= SecurityContextHolder.*getContext*().getAuthentication();  
 if(username != null && authentication== null)  
 {  
 //Authenticate karo  
 UserDetails userDetails=userDetailsService.loadUserByUsername(username);  
 if (jwtService.isTokenValid(jwt,userDetails))  
 {  
 UsernamePasswordAuthenticationToken authenticationToken=new UsernamePasswordAuthenticationToken(  
 userDetails,  
 null,  
 userDetails.getAuthorities()  
 );  
 authenticationToken.setDetails(  
 new WebAuthenticationDetailsSource().buildDetails(request)  
 );  
 SecurityContextHolder.*getContext*().setAuthentication(authenticationToken);  
 }  
 }  
 filterChain.doFilter(request,response);  
}