# 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 ptovider..for this project we would be using DaoAuthenticationProvider

**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();  
}