## **BASIC:**

## In order to add security to our Spring Boot application, we need to add the *security starter dependency*:

|  |  |
| --- | --- |
|  | <dependency>      <groupId>org.springframework.boot</groupId>      <artifactId>spring-boot-starter-security</artifactId>  </dependency> |

1. application will prompt for user name and password; user name is user and password is printed on console.
2. This will include the *SecurityAutoConfiguration*class – containing the initial/default security configuration.
3. Notice how we didn’t specify the version here, with the assumption that the project is already using Boot as the parent.
4. Simply put, **by default, the Authentication gets enabled for the Application. Also, content negotiation is used to determine if basic or formLogin should be used.**
5. There are some predefined properties, such as:

|  |  |
| --- | --- |
| 1  2 | spring.security.user.name  spring.security.user.password |

1. If we don’t configure the password using the predefined property *spring.security.user.password*and start the application, we’ll notice that a default password is randomly generated and printed in the console log:

## CUSTOMIZE SECURITY- custom user name and password – in memory or from database:

1. For customization – WebSecurityConfigurerAdapter class is used
2. Use @EnableWebSecurity annotation on this class --- WebSecurityConfigurerAdapter
3. Override 2 Configure methods – one takes httpSecurity and authenticationManagerBuilder.

OR

1. Override UserDetailsService
2. @Override
3. @Bean
4. **protected** UserDetailsService userDetailsService() {
5. List<UserDetails> user = **new** ArrayList();
6. user.add(User.~~withDefaultPasswordEncoder~~().username("HK").password("HK").roles("USER").build());
8. **return** **new** InMemoryUserDetailsManager(user);
9. }
10. @Configuration
11. @EnableWebSecurity
12. public class BasicConfiguration extends WebSecurityConfigurerAdapter {
14. @Override
15. protected void configure(AuthenticationManagerBuilder auth)
16. throws Exception {
17. auth
18. .inMemoryAuthentication()
19. .withUser("user")
20. .password("password")
21. .roles("USER")
22. .and()
23. .withUser("admin")
24. .password("admin")
25. .roles("USER", "ADMIN");
26. }
28. @Override
29. protected void configure(HttpSecurity http) throws Exception {
30. http
31. .authorizeRequests()
32. .anyRequest()
33. .authenticated()
34. .and()
35. .httpBasic();
36. }
37. }

## Oauth 2.0

1. use-security-oauth2 in pom.xml for oAuth2.0
2. OAuth components:
   1. Authorization server
   2. resource owner,
   3. resource server,
   4. client

an OAuth 2 authorization server is responsible for managing client details, verifying a resource owner’s authorization, and generating tokens such as authorization code, access, and refresh tokens.

**GRANT TYPE:**

**Authorization Code:** used with server-side Applications

**Implicit:** used with Mobile Apps or Web Applications (applications that run on the user's device)

**Resource Owner Password Credentials:** used with trusted Applications, such as those owned by the service itself

**Client Credentials:** used with Applications API access