**Spring Security:**

**Spring Security:**

1) Authentication (verifying credentials)

2) Authorization (can this user access specific functionality)

-> Security is very important for every web application

-> To protect our application & application data we need to implement security logic

-> Spring Security concept we can use to secure our web applications / REST APIs

-> To secure our spring boot application we need to add below starter in pom.xml file

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

Note: When we add this dependency in pom.xml file then by default our application will be secured with basic authentication. It will generate random password to access our application.

Note: Generated Random Password will be printed on console.

-> We need to use below credentials to access our application

Username: user

Password : <copy the pwd from console>

-> When we access our application url in browser then it will display "Login Form" to authenticate our request.

-> To access secured REST API from postman, we need to set Auth values in POSTMAN to send the request

|  |
| --- |
| Auth : Basic Auth  Username : user  Password : <copy-from-console> |

**How to override Spring Security Default Credentials:**

-> To override Default credentials we can configure security credentials in application.properties file or application.yml file like below

spring.security.user.name=ashokit

spring.security.user.password=ashokit@123

-> After configuring credentials like above, we need to give above credentials to access our application / api.

**How to secure specific URL Patterns:**

-> When we add 'security-starter' in pom.xml then it will apply security filter for all the HTTP methods of our application.

-> But in reality we need to secure only few methods not all methods in our application.

##For Example##

/ login-page --> security not required

/ transfer ---> security required

/ balance ---> security required

/ about-us ---> security not required

/ contact-us ---> security not required

-> In order to achieve above requirement we need to Customize Security Configuration in our project like below

|  |
| --- |
| @Configuration  @EnableWebSecurity  public class SecurityConfigurer {  @Bean  public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {  http.authorizeHttpRequests((authorize) -> authorize  .requestMatchers("/contact", "/swagger-ui.html").permitAll()  .anyRequest().authenticated()  )  .httpBasic(withDefaults())  .formLogin(withDefaults());  return http.build();  }  } |

**Spring Security In-Memory Authentication:**

-> In Memory Authentication means storing user credentials in the program for Authentication Purpose.

-> This is not recommended for production.

|  |
| --- |
| @Bean  public InMemoryUserDetailsManager inMemoryUsers() {    UserDetails ashokUser = User.withDefaultPasswordEncoder()  .username("ashok")  .password("ashok")  .authorities("ADMIN")  .build();      UserDetails johnUser = User.withDefaultPasswordEncoder()  .username("john")  .password("john")  .authorities("USER")  .build();    return new InMemoryUserDetailsManager(ashokUser, johnUser);    } |

**Spring Boot Security with JDBC Authentication:**

=> JDBC Authentication is used to fetch Db table data for User authentication purpose.

**Spring Security Work with JDBC Internal flow:**

1. It take login form from data (username & Password)
2. Encrypt user given pwd using pwd-encoder (bcrypt)
3. Retrieve user record from DB based on given username
4. If record is available then it compare encrypted pwd with user pwd
5. Retrieve user Authorities from authorities from DB table based on given username
6. Verify user having permission to access url or not

|  |
| --- |
| ## Step-1 ) Setup Database tables with required data  -- users table structure  CREATE TABLE `users` (  `username` VARCHAR(50) NOT NULL,  `password` VARCHAR(120) NOT NULL,  `enabled` TINYINT(1) NOT NULL,  PRIMARY KEY (`username`)  );  -- authorities table structure  CREATE TABLE `authorities` (  `username` VARCHAR(50) NOT NULL,  `authority` VARCHAR(50) NOT NULL,  KEY `username` (`username`),  CONSTRAINT `authorities\_ibfk\_1` FOREIGN KEY (`username`)  REFERENCES `users` (`username`)  ); |

|  |
| --- |
| ========= Online Encrypt : https://bcrypt-generator.com/ ======================  -- insert records into table  insert into users values ('john', '$2a$12$D8x8tLL4Q4Q/7aLI5vBt8.QS6FKn8tN7h3hzgc8TTimxAKvAnMUFu', 1);  insert into users values ('smith', '$2a$12$hhdXrq63gHFVkL2G1jwDuOBcrNEjX7mwZHUXEgQGwL18v6CD1zkra', 1);  insert into authorities values ('john', 'ROLE\_ADMIN');  insert into authorities values ('john', 'ROLE\_USER');  insert into authorities values ('smith', 'ROLE\_USER');  ## Step-2) Create Boot application with below dependencies  a) web-starter  b) security-starter  c) data-jdbc  d) mysql-connector  e) lombok  f) devtools  ## Step-3 ) Configure Data source properties in application.yml file  spring:  datasource:  driver-class-name: com.mysql.cj.jdbc.Driver  password: AshokIT@123  url: jdbc:mysql://localhost:3306/sbms66  username: ashokit  jpa:  show-sql: true  Step-4) Create Rest Controller with Required methods  @RestController  public class UserRestController {  @GetMapping(value = "/admin")  public String admin() {  return "<h3>Welcome Admin :)</h3>";  }  @GetMapping(value = "/user")  public String user() {  return "<h3>Hello User :)</h3>";  }  @GetMapping(value = "/")  public String welcome() {  return "<h3>Welcome :)</h3>";  }  }  Step-5) Create Security Configuration class like below with Jdbc Authentication Manager  package in.ashokit;  import javax.sql.DataSource;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.context.annotation.Bean;  import org.springframework.context.annotation.Configuration;  import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;  import org.springframework.security.config.annotation.web.builders.HttpSecurity;  import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;  import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;  import org.springframework.security.web.SecurityFilterChain;  @Configuration  @EnableWebSecurity  public class SecurityConfiguration {    private static final String ADMIN = "ADMIN";  private static final String USER = "USER";  @Autowired  private DataSource dataSource;    @Autowired  public void authManager(AuthenticationManagerBuilder auth) throws Exception {  auth.jdbcAuthentication()  .dataSource(dataSource)  .passwordEncoder(new BCryptPasswordEncoder())  .usersByUsernameQuery("select username,password,enabled from users where username=?")  .authoritiesByUsernameQuery("select username,authority from authorities where username=?");  }    @Bean  public SecurityFilterChain securityConfig(HttpSecurity http) throws Exception {    http.authorizeHttpRequests( (req) -> req  .antMatchers("/admin").hasRole(ADMIN)  .antMatchers("/user").hasAnyRole(ADMIN,USER)  .antMatchers("/").permitAll()  .anyRequest().authenticated()  ).formLogin();    return http.build();  }  } |

**How to work with UserDetailsService in Spring Security:**

=> UserDetailsService is a predefined interface which contains loadUserByUsername (String name) method.

=> This is used to load User record for Authentication purpose in Spring Security.

=> We can implement UserDetailsService interface and we can write the logic to retrieve User record based on given username for Authentication purpose.

=> If we give UserDetailsService object to AuthenticationManagerBuild then AuthManager will call this method for every login request.

**Login and Registration using Spring Security:**

|  |
| --- |
| ## Git Hub repo : https://github.com/ashokitschool/SpringBoot\_Security\_Register\_Login.git  ## 1) Create Boot app with required dependencies ##  a) web-starter  b) data-jpa-starter  c) mysql  d) security-starter  e) devtools  ## 2) Configure Data Source properties in yml file ##  ## 2) Create Entity class & Repository interface ##  @Repository  public interface CustomerRepo extends CrudRepository<Customer, Integer> {  public Customer findByUname(String cuname);  }  ## 3) Create UserDetailsService class ##  @Service  public class MyUserDetailsService implements UserDetailsService {  @Autowired  private CustomerRepo crepo;  @Override  public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  Customer c = crepo.findByUname(username);  return new User(c.getUname(), c.getPwd(), Collections.emptyList());  }  }  ## 4) Create Security Config Class ##  @Configuration  @EnableWebSecurity  public class AppSecurityConfig {    @Autowired  private MyUserDetailsService userDtlsSvc;    @Bean  public PasswordEncoder pwdEncoder() {  return new BCryptPasswordEncoder();  }    @Bean  public AuthenticationProvider authenticationProvider(){  DaoAuthenticationProvider authenticationProvider=  new DaoAuthenticationProvider();  authenticationProvider.setUserDetailsService(userDtlsSvc);  authenticationProvider.setPasswordEncoder(pwdEncoder());  return authenticationProvider;  }  @Bean  public AuthenticationManager authenticationManager(AuthenticationConfiguration config) throws Exception {  return config.getAuthenticationManager();  }    @Bean  public SecurityFilterChain securityConfig(HttpSecurity http) throws Exception {  return http.csrf().disable()  .authorizeHttpRequests()  .requestMatchers("/register", "/login").permitAll()  .and()  .build();  }  }  ## 5) Create RestController with required methods  @RestController  public class CustomerRestController {  @Autowired  private CustomerRepo crepo;  @Autowired  private PasswordEncoder pwdEncoder;  @Autowired  private AuthenticationManager authManager;  @PostMapping("/login")  public ResponseEntity<String> loginCheck(@RequestBody Customer c) {    UsernamePasswordAuthenticationToken token =  new UsernamePasswordAuthenticationToken(c.getUname(), c.getPwd());  try {  Authentication authenticate = authManager.authenticate(token);  if (authenticate.isAuthenticated()) {  return new ResponseEntity<>("Welcome To Ashok IT", HttpStatus.OK);  }  } catch (Exception e) {  //logger  }  return new ResponseEntity<String>("Invalid Credentials", HttpStatus.BAD\_REQUEST);  }  @PostMapping("/register")  public String registerCustomer(@RequestBody Customer customer) {    // duplicate check  String encodedPwd = pwdEncoder.encode(customer.getPwd());  customer.setPwd(encodedPwd);  crepo.save(customer);  return "User registered";  }  }  ## 6) Run the application and test it  ##############  OAuth 2.0  ##############  ### 1) Create Spring Boot application with below dependencies  a) web-starter  b) security-starter  c) oauth-client  ### 2) Create OAuth app in Github.com  (Login --> Profile -> Settings --> Developer Settings --> OAuth Apps --> Create App --> Copy Client ID & Client Secret)  ### 3) Configure GitHub OAuth App client id & client secret in application.yml file like below  spring:  security:  oauth2:  client:  registration:  github:  clientId:  clientSecret:  ### 4) Create Rest Controller with method  @RestController  public class WelcomeRestController {  @GetMapping("/")  public String welcome() {  return "Welcome to Ashok IT";  }  }  #### 5) Run the application and test it.  Assignment : Spring Boot with oAuth using google account. Get username also from google and display that in response. |