Spring Security

Spring Security MySQL Part4 Telusko

Security – Introduction

- bcrypt – password hashing function
  - incorporates salt – random data used as an additional input to a one way function
  - iteration can help to slow down brute force attacks
- provides both authentication and authorization (including roles)
-

Spring Security

- tomcat-jasper – Convert JSP into a Servlet
- Use spring-boot-starter-security
  - Once add it will go to a login page instead of going to the home page
  - By default the username is user and you get a generated password
  - Provide UserName and password
- What if you want to have your own username and password
  - Example
    - @Configuration
    - @EnableWebSecurity
    - @Bean
    - public class AppSecurityConfig extends WebSecurityConfigurerAdapter {
      - @Override
      - public UserDetailsService userDetailsService() {
        - List<UserDetails> users = new ArrayList<>(); // UsersDetails is a Spring Class
        - users.add(User.withDefaultPasswordEncoder()).username(“navin”).password(“1234”.roles(“USER”).build());
        -
        - return new InMemoryUserdetailsManager(users);

      }}

Spring Security where username/password will be verified form the database

- Need to add spring-boot-starter-data-jpa and mysql-connector-java
- application.properties spring.datasource[.url, username, password, driver-class-name]
- @Configuration
- @EnableWebSecurity
- public class AppSecurityConfig extends WebSecurityConfigurerAdapter {
- ~ @autowrired
  - private UserDetailsService userDetailsService; // contains loadUserByUsername(String userName)
  - @Bean
  - public AuthentcationProvider authProvider() {
    - DaoAuthenticationProvider provider = new DaoAuthenticationProvier();
    - provider.setUserDetailsService(userDetailsService);
    - provider.setPasswordEncoder(NoOpPasswordEncoder.getInstance());
    -
    - return provider
  - } }

Spring Security where username/password will be verified form the database

- Create a class for the mySQL user table which is an @Entity and have an id, username, password along with setters and getters
- public interface UserRepository exends JpaRepository<User, Long> {}
-
- @Service
- public class MyUserDetailsService implements UserDetailsService {
  - @Autowired
  - private UserRepository repo;
  -
  - @Override
  - public UserDetails loadUserByUserName(String username) throws NameNotFoundException {
    - User user = repo.findByUsername(username);
    - if (user== null) throw new UsernameNotFoundExcepton(“User 404”);
    - return new userPrincipal(user);
  - }
- }

Spring Security where username/password will be verified form the database

- // Principle means current User
- class UserPrinciple implements UserDetails
  - // which has setters/getters for getPassword, getUserName, isAccountExpired, isAccountNonLocked, isEnable
  - private User user
  - public UserPrincipal(User user) {
    - supper();
    - this.user = user }
  - function like password can then use user.getPassword()
  - @Override
  - public Collection<? extends GrantedAuthority> getAuthorities() { return Collections.singleton(new SimpleGrantedAuthority("USER")); }

Bcrypt Password Encoder Spring Security from Login Parts 6

- provider.setPasswordEncoder(new BCryptPasswordEncoder);
- Create
- <Example Page>
  - ${SPRING_SECURITY_LAST_EXCEPTION.message}
  - <form action=”login” method=”post”>
    - <tr>
      - <td>User:</td>
      - <td><input type=”text” name=”username” value=””></td>
    - </tr>
    - <tr>
      - <td>Password:</td>
      - <td><input name=”submit” type=”submit” value=”submit” /></td>
    - </tr>
  - </form>
- How do we tell Spring to use our own login page
  -

Spring Security from Login Parts 6

- How do we tell Spring to use our own login page
  - From WebSecurityConfigurationAdapter overide config(HttpSecurity http)
  - @Override
  - protected void configure(HttpSecurity http) throws Exception {
    - http.
      - csrf().disasble()   // Cross Site request forgery
      - .authorizeRequests().antMatcher(“/login”).permitAll()
      - .anyRequest().authenticated().
      - .formLogin()
      - .loginPage(:/login”)// Calls a Controller to handle the page
      - .logout().invalidateHttpSecssion(true);
      - .clearAuthentication();
      - .logoutRequestMatcher(new AntPathRequestMatcher(“/logout”)
      - ..logoutSuccessURL(“/logout-success”).permitAll();
      -
    - @Controller
    - public class HomeController {
      - @RequestMapping(“/”) public String home() { return “home.jsp”; }
      - @RequestMapping(“/login”) public String loginPage() { return “login.jsp” }
      - have one for logout

Spring Boot Security OAuth2

- oauth2 – authorization framework that enables applications to obtain limited access to user accounts such as Facebook, Github and Digital Ocean
- need to add spring-security-oauth2-autoconfigure
- To application properties
- Most Important
  - Name – The name of the token in the application
  - Scope – Features example access to profile and email id
  - clientId, secretKKey – get google account
- To the class that extends WebSecurityConfigurerAdapter add the annotation @EnableOAuth2Sso
  - don’t need the data base authentication pt userDetailService
    - security.oauth2.client.client, clientSecret, accesssTokenURI, userAuthroizationUri, tokenName, authentationScheme, clientAuthenticationScheme, scope
    -
  - protected void configure(HttpSecurity http) throws Exception {
    - http.
      - csrf().disasble()   // Cross Site request forgery
      - .authorizeRequests().antMatcher(“/login”).permitAll()
      - ..anyRequest()authenticated();
  - }
- For the

Spring Boot Security OAuth2

- In the @Controller add
  - @ResponseBody
  - public Principal user(Principal principal) {
    - return principal;
    - }
- To Login
  - Login again the google login appears
  -