### Spring Security

**spring-boot-starter-security**dependency adds all the security related dependencies.

**spring-boot-starter-jdbc** is the starter which is needed for using**JDBC**

Servlet filter called “DelegatingFilterProxy” with url-pattern **/\*** intercepts incoming http requests and enter Spring Security framework for security checks to authenticate users.

In the web.xml we add the DelegatingFilterProxy which is delegating proxy to automatically intercept a URL with a particular pattern to apply spring security.

#### **web.xml**

<!-- Spring Security -->

    <filter>

        <filter-name>springSecurityFilterChain</filter-name>

        <filter-class>**org.springframework.web.filter.DelegatingFilterProxy**</filter-class>

    </filter>

    <filter-mapping>

        <filter-name>springSecurityFilterChain</filter-name>

        <url-pattern>/\*</url-pattern>

    </filter-mapping>

#### **spring-security.xml**

    <security:http auto-config="true">

        <security:intercept-url pattern="/admin\*" access="ROLE\_ADMIN" />

        <security:form-login login-page="/login"

default-target-url="/admin" authentication-failure-url="/error" />

        <security:logout logout-success-url="/logout" />

    </security:http>

    <security:authentication-manager>

        <security:authentication-provider>

            <security:user-service>

                <security:user name="admin" password="admin123" authorities="ROLE\_ADMIN" />

            </security:user-service>

        </security:authentication-provider>

    </security:authentication-manager>

* **@EnableWebSecurity** annotation enables spring security configuration which is defined in **WebSecurityConfigurerAdapter**
* We have extended**WebSecurityConfigurerAdapter,**which allows us to override spring’s security default feature. In our example we want all the requests to be authenticated using **HTTP Basic authentication.**
* **configure()** method configures the **HttpSecurity** class which authorizes each **HTTP** request which has been made. In our example **‘/user’**should be allowed for the user with**USER**role  and**‘/admin’** should be allowed for the user with**USER (or) ADMIN**role.
  + **.httpBasic()** **–>** Tells spring to use the **HTTP Basic Authentication** method to authenticate the user
  + **authorizeRequests() .antMatchers(“/user”).hasRole(“USER”) .antMatchers(“/admin”).hasRole(“ADMIN”)** **–>** All requests to the endpoint must be authorized or else they should be rejected.
  + **.csrf().disable() –>** Disables CSRF protection
* **configureGlobal()** method configures the **AuthenticationManagerBuilder** class with the valid user credentials and the allowed roles. The **AuthenticationManagerBuilder** class creates the **AuthenticationManger** which is responsible for authenticating the credentials. In our example we have used the **inMemoryAuthentication**, you can choose other authentication types such **JDBC**, **LDAP**.
* In [**Spring Boot 2**](https://spring.io/projects/spring-boot), we need to pass the encoded password to the **password()** method, since we have used **BCryptPasswordEncoder** we are directly encoding the password using **encode()**method.

The overridden method configure(AuthenticationManagerBuilder auth) configure the in memory authentication with user credentials and roles. You can configure the other authentications too such as JDBC, LDAP etc.

The overridden method configure(HttpSecurity http) configure the web based security for all HTTP request. By default it will be applied to all requests, but can be restricted using the requestMatcher() or other similar methods.

form-based authentication

.formLogin() - It shows the username/password as separate screen.

What is the **HTTP basic authentication**? It is a simple challenge and response mechanism used by a server to challenge a client request. In HTTP basic authentication, client’s username and password are concatenated, base64 encoded and passed to server in Authorization HTTP header.

.httpBasic();

It shows the username/password as popup.

<https://www.boraji.com/spring-security-4-custom-login-from-example>

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.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

@Configuration

@EnableWebSecurity

public class SecurityConfig extends WebSecurityConfigurerAdapter {

@Override

public void configure(AuthenticationManagerBuilder auth) throws Exception {

auth.inMemoryAuthentication().withUser("admin")

.password("admin123").roles("ROLE\_ADMIN ");

OR

authenticationMgr.inMemoryAuthentication()

.withUser("employee").password("employee").authorities("ROLE\_USER")

.and().withUser("javainuse").password("javainuse").authorities("ROLE\_USER", "ROLE\_ADMIN");

OR

auth.jdbcAuthentication().dataSource(dataSource)

.authoritiesByUsernameQuery("select USERNAME, ROLE from EMPLOYEE where USERNAME=?")

.usersByUsernameQuery("select USERNAME, PASSWORD, 1 as enabled from EMPLOYEE where USERNAME=?")

.passwordEncoder(new BCryptPasswordEncoder());

}

@Override

public void configure(HttpSecurity http) throws Exception {

http.antMatcher("/\*\*").authorizeRequests().anyRequest().hasRole("ROLE\_ADMIN ")

.and().formLogin().loginPage("/login")

.failureUrl("/error").loginProcessingUrl("/login")

.permitAll().and().logout()

.logoutSuccessUrl("/listEmployees.html");

OR

http.authorizeRequests().antMatchers("/").hasRole("USER") .and() .httpBasic();

OR

http.authorizeRequests().antMatchers("/").permitAll()

.antMatchers("/welcome").hasAnyRole("USER", "ADMIN")

.antMatchers("/getEmployees").hasAnyRole("USER", "ADMIN")

.antMatchers("/addNewEmployee").hasAnyRole("ADMIN")

.anyRequest().authenticated()

.and().formLogin().**loginPage("/login")**.permitAll()

.and().logout().permitAll();

http.csrf().disable();

}

@Bean

public PasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

}

It is important to note few variables to understand the spring security custom login form. Variable such as j\_spring\_security\_check’, j\_username, j\_password in the below login.jsp are predefined in the spring security framework and we shouldn’t modify it, otherwise it won’t work as expected.

<!-- location of the root application context xml file -->

    <context-param>

        <param-name>contextConfigLocation</param-name>

        <param-value>

            /WEB-INF/mvc-dispatcher-servlet.xml,

            /WEB-INF/spring-security.xml

        </param-value>

    </context-param>

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import org.springframework.security.crypto.password.PasswordEncoder;

public class PasswordHashing {

        public static void main(String[] args) {

        String originalPassword = "admin123";

        PasswordEncoder encoder = new BCryptPasswordEncoder();

        String encodedPassword = encoder.encode(originalPassword);

        System.out.println("Hashed Password : " + encodedPassword);

    }

}

<security:authentication-manager>

        <security:authentication-provider>

            <security:password-encoder hash="bcrypt" />

            <security:user-service>

                <security:user name="admin" password="$2a$10$dIJYGvKNvITeljfEgi9HgOyUZFZqWZca/vLAwHzpgTqVSe/EyXZA." authorities="ROLE\_ADMIN" />

            </security:user-service>

        </security:authentication-provider>

    </security:authentication-manager>

In order to access a secured resource the user has to provide the request to our API with the header information containing the **username** and **password** to access the resource.

CSRF stands for Cross-Site Request Forgery. It is an attack that forces an end user to execute unwanted actions on a web application in which they are currently authenticated. CSRF attacks specifically target state-changing requests, not theft of data, since the attacker has no way to see the response to the forged request.

<http://codenuclear.com/category/java/java-interview-programs/>

# **Spring Security 5 - Custom UserDetailsService example**

The UserDetailsService is a core interface in Spring Security framework, which is used to retrieve the user’s authentication and authorization information.

It has a single read-only method named as loadUserByUsername() which locate the user based on the username.

<https://www.boraji.com/spring-security-5-custom-userdetailsservice-example>

import org.springframework.security.core.userdetails.User.UserBuilder;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.security.core.userdetails.UserDetailsService;

import org.springframework.security.core.userdetails.UsernameNotFoundException;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import com.boraji.tutorial.spring.model.User;

public class UserDetailsServiceImp implements UserDetailsService {

@Override

public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

/\*Here we are using dummy data, you need to load user data from

database or other third party application\*/

User user = findUserbyUername(username);

UserBuilder builder = null;

if (user != null) {

builder = org.springframework.security.core.userdetails.User.withUsername(username);

builder.password(new BCryptPasswordEncoder().encode(user.getPassword()));

builder.roles(user.getRoles());

} else {

throw new UsernameNotFoundException("User not found.");

}

return builder.build();

}

private User findUserbyUername(String username) {

if(username.equalsIgnoreCase("admin")) {

return new User(username, "admin123", "ADMIN");

}

return null;

}

}

import org.springframework.context.annotation.Bean;

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.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

import org.springframework.security.core.userdetails.UserDetailsService;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

@EnableWebSecurity

public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

@Bean

public UserDetailsService userDetailsService() {

return new UserDetailsServiceImp();

};

@Bean

public BCryptPasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

};

@Override

protected void configure(AuthenticationManagerBuilder auth) throws Exception {

auth.userDetailsService(userDetailsService()).passwordEncoder(passwordEncoder());

}

}

# **Spring Security 5 - Remember-Me authentication example**

create table users(

username varchar(50) not null primary key,

password varchar(100) not null,

enabled boolean not null

);

create table authorities (

username varchar(50) not null,

authority varchar(50) not null,

constraint fk\_authorities\_users foreign key(username) references users(username)

);

create unique index ix\_auth\_username on authorities (username,authority);

create table persistent\_logins(

username varchar(50) not null,

series varchar(64) primary key,

token varchar(64) not null,

last\_used timestamp not null

);

<https://www.boraji.com/spring-security-5-remember-me-authentication-example>

To enable remember-me authentication, you need to invoke the rememberMe() method of the HttpSecurity class and to store the generated tokens in database table invoke the tokenRepository() method with PersistentTokenRepository argument as shown in the below section.

Now, create a @Configuration class by extending the WebSecurityConfigurerAdapter class and annotate it with @EnableWebSecurity as follows.

@Override

protected void configure(HttpSecurity http) throws Exception {

http.authorizeRequests().anyRequest().hasAnyRole("ADMIN", "USER")

.and()

.authorizeRequests().antMatchers("/login\*\*").permitAll()

.and()

.formLogin().loginPage("/login").loginProcessingUrl("/loginAction").permitAll()

.and()

.logout().logoutSuccessUrl("/login").permitAll()

.and()

**.rememberMe().rememberMeParameter("remember-me").tokenRepository(tokenRepository())**

.and()

.csrf().disable();

}

@Bean

public **PersistentTokenRepository tokenRepository**() {

JdbcTokenRepositoryImpl jdbcTokenRepositoryImpl=new JdbcTokenRepositoryImpl();

jdbcTokenRepositoryImpl.setDataSource(dataSource);

return jdbcTokenRepositoryImpl;

}

<https://www.boraji.com/spring-mvc-5-spring-security-5-hibernate-5-example>