# Spring MVC Form Validator

When we accept user inputs in any web application, it become necessary to validate them. We can validate the user input at client side using JavaScript but it’s also necessary to validate them at server side to make sure we are processing valid data incase user has JavaScript disabled.

[Spring MVC Framework](https://www.journaldev.com/2433/spring-mvc-tutorial) supports JSR (**Java Specification Requests**)-303 specs by default and all we need is to add JSR-303 and its implementation dependencies in Spring MVC application. Spring also provides **@Validator** annotation and **BindingResult** class through which we can get the errors raised by Validator implementation in the controller request handler method.

We can create our custom validator implementations by two ways – first one is to create an annotation that confirms to the JSR-303 specs and implement its Validator class. Second approach is to implement the **org.springframework.validation.Validator** interface and add set it as validator in the Controller class using **@InitBinder** annotation.

Let’s create a simple Spring MVC project in Spring Tool Suite where we will use JSR-303 specs with its implementation artifact **hibernate-validator**. We will use annotation-based form validation and create our own custom validator based on JSR-303 specs standards. We will also create our own custom validator class by implementing **Validator** interface and use it in one of the controller handler methods

***Example***

POM.xml

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.1.1.RELEASE</version>

</dependency>

<dependency>

<groupId>org.hibernate.validator</groupId>

<artifactId>hibernate-validator</artifactId>

<version>6.0.13.Final</version>

</dependency>

Student.java

**public** **class** Student {

@NotNull(message = "Please enter some sid")

Integer sid;

@Size(min=3,message="required")

String sName;

@NotNull(message = "Roll Cannot be Empty")

@Min(2)

@Max(4)

Integer roll;

@NotEmpty(message = "Please Enter valid Email")

@Email

String email;

@DateTimeFormat(pattern = "MM/dd/yyyy")

@NotNull(message = "BirthDate Cannot be Empty")

@Past

Date birthDay; ***Getter and Setter***

}

Index.jsp

<html>

<body>

<h2>Hello World!</h2>

<a href=*"showStudent"*>Show Student Form</a>

</body>

</html>

StudentController.java

@Controller

**public** **class** StudentController {

@RequestMapping(value = "/showStudent")

**public** String showStudent(Model m) {

System.***out***.println("First");

m.addAttribute("stu", **new** Student());

**return** "../Student";

}

@RequestMapping(value = "student", method = RequestMethod.***POST***)

**public** String getStudent(@Valid @ModelAttribute("stu") Student stu,BindingResult br) {

System.***out***.println("Method");

**if**(br.hasErrors()) {

System.***out***.println("In Error");

**return** "../Student";

}**else** {

System.***out***.println("Else Error");

**return** "Success";

Student.jsp

<%@ page language=*"java"* contentType=*"text/html; charset=ISO-8859-1"*

pageEncoding=*"ISO-8859-1"*%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<%@ taglib prefix=*"form"* uri=*"http://www.springframework.org/tags/form"* %>

<html>

<style>

*.error*{color:*red*}

</style>

<head>

<meta http-equiv=*"Content-Type"* content=*"text/html; charset=ISO-8859-1"*>

<title>Insert title here</title>

</head>

<body>

<form:form action=*"student"* modelAttribute=*"stu"* method=*"post"*>

Sid: <form:input path=*"sid"*/>

<form:errors path=*"sid"* cssClass=*"error"*/><br><br>

S-Name: <form:input path=*"sName"*/>

<form:errors path=*"sName"* cssClass=*"error"*/><br><br>

S-Roll: <form:input path=*"roll"*/>

<form:errors path=*"roll"* cssClass=*"error"*/><br><br>

S-Email: <form:input path=*"email"*/>

<form:errors path=*"email"* cssClass=*"error"*/><br><br>

S-B-Date: <form:input path=*"birthDay"*/>

<form:errors path=*"birthDay"* cssClass=*"error"*/><br><br>

<input type=*"submit"* value=*"submit"*>

</form:form>

</body>

</html>

One-servlet.xml

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xmlns:context=*"http://www.springframework.org/schema/context"*

xmlns:mvc=*"http://www.springframework.org/schema/mvc"*

xsi:schemaLocation=*"*

*http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans.xsd*

*http://www.springframework.org/schema/context*

*http://www.springframework.org/schema/context/spring-context.xsd*

*http://www.springframework.org/schema/mvc*

*http://www.springframework.org/schema/mvc/spring-mvc.xsd"*>

<context:component-scan base-package=*"p1"*></context:component-scan>

<mvc:annotation-driven></mvc:annotation-driven>

<bean id=*"viewResolver"*

class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>

<property name=*"prefix"* value=*"/OutPut/"*></property>

<property name=*"suffix"* value=*".jsp"*></property>

</bean>

</beans>

Web.xml

<servlet>

<servlet-name>One</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>One</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

# Spring MVC Form Custom Validator

# To create our own valiator annotation we need to use one interface called [javax](eclipse-javadoc:%E2%98%82=SpringMavenPro/C:%5C/Users%5C/shahn%5C/.m2%5C/repository%5C/javax%5C/validation%5C/validation-api%5C/2.0.1.Final%5C/validation-api-2.0.1.Final.jar%3Cjavax).[validation](eclipse-javadoc:%E2%98%82=SpringMavenPro/C:%5C/Users%5C/shahn%5C/.m2%5C/repository%5C/javax%5C/validation%5C/validation-api%5C/2.0.1.Final%5C/validation-api-2.0.1.Final.jar%3Cjavax.validation).ConstraintValidator

***With Appending above Example***

Student.java

**public** **class** Student {

@Phone

String phone; ***Add on above Example class Getter and Setter***

Phone.java Create own Annotation

@Documented

@Constraint(validatedBy = PhoneValidator.**class**)

@Target({ElementType.***FIELD***,ElementType.***METHOD***})

@Retention(RetentionPolicy.***RUNTIME***)

**public** **@interface** Phone {

String message() **default** "Enter Phone";

Class<?>[] groups() **default** {};

Class<? **extends** Payload>[] payload() **default** {};

}

PhoneValidator.java

**public** **class** PhoneValidator **implements** ConstraintValidator<Phone, String>{

**public** **boolean** isValid(String phoneNo, ConstraintValidatorContext ctx) {

**if**(phoneNo == **null**){

**return** **false**;

}**else** **if**(phoneNo.length() < 10){

**return** **false**;

}**else** {

**return** **true**;

Student.jsp Append in Student.jsp

S-Phone: <form:input path=*"phone"*/>

<form:errors path=*"phone"* cssClass=*"error"*/><br><br>

# -----------: *Spring Security Start Here*: -----------

# Spring Security provide comprehensive security services for J2EE based application. So, it’s not a Firewall, proxy servers, OS security, JVM security etc.

# *Major Operation provides*

# Authentication: (who you are) like taking username and password where username is Identification and password is Verification

# Means it say I am SAM in username so Spring will tell enter your pass and prove you are a SAM, So Authentication is a combination of Identification + Verification

# Authorization: means Access Control like Admin can access more pages as compare to normal user

# *Possession Based authentication?*

# Phone / Text Message: Like Receiving OTP

# Key Cards and Badge: Like Id in Office

# *Principal in Spring Security?* It’s a currently Logged In user

# *Spring Security Default Behavior?*

# Add Mandate auth for all URLs

# Adds login form

# Handles login error

# *Default Credential?*

# If we don’t provide any credential to S-S it will create Username = user and password it will print in console in Encrypted form

# To change go to application.properties

# spring.security.user.name=foo

# spring.security.user.password=foo

# *How S-S internally Work?*

# S-S Internally used Filter Mechanism of Servlet, so before calling our controller It will internally call Filter and it will check the Security status valid or not with the help of Spring-security.xml file.

# *S-S Use Cases*

# First User reached Homepage like Amazon.com (without Login)

# When it clicks to some secure thing like Click on Buy some product then S-S will come into picture and tell user to Login first

# Now Login page will do credential check like using DB etc.... and based on that it will provide Fail or Success

# *Important Jars*

# spring-security-config-5.3.8.RELEASE.jar

# spring-security-core-5.3.8.RELEASE.jar

# spring-security-web-5.3.8.RELEASE.jar

# and Some DTD part also

# *3 types of Configurations mostly use*

# Http Basic

# Form Login (Default InBuild)

# Custom Form Login

# *Http Basic Example...*

**Pom.xml** <dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-web</artifactId>

<version>5.3.8.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-core</artifactId>

<version>5.3.8.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-config</artifactId>

<version>5.3.8.RELEASE</version>

</dependency>

**web.xml** *(Along with DispatcherServlet)*

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<!DOCTYPE xml>

<web-app xmlns=*"http://xmlns.jcp.org/xml/ns/javaee"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://xmlns.jcp.org/xml/ns/javaee*

*http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"*

version=*"3.1"*>

**<!-- change webApp tag DTD also -->**

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

<filter>

<filter-name>springSecurityFilterChain**(FIXED NAME)**</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>

<context-param>

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

<param-value>

/WEB-INF/One-servlet.xml

/WEB-INF/spring-security.xml

</param-value>

</context-param>

One-servlet.xml

**<mvc:annotation-driven />**

WEB-INF/spring-security.xml

**<beans:beans**

xmlns=*"http://www.springframework.org/schema/security"*

xmlns:beans=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans.xsd*

*http://www.springframework.org/schema/security*

*http://www.springframework.org/schema/security/spring-security.xsd"*>

<http auto-config=*"true"* use-expressions=*"true"*>

<intercept-url pattern=*"/welcome"* access=*"hasRole('ROLE\_ADMIN')"* />

<http-basic/>

</http>

<authentication-manager>

<authentication-provider>

<user-service>

<user name=*"Sam"* password=*"{noop}123"* authorities=*"ROLE\_ADMIN"* />

</user-service>

</authentication-provider>

</authentication-manager>

</beans:beans>

**auto-config=*"true"*** it will show default login Spring embedded form if we don’t define any like **http-basic** etc.

**use-expressions=*"true"***use to enable EL in-xml file like we are using noop in password field.

***{noop}*** Spring by default encode password before sending in request to disable encoder.

**pattern=*"/\*\*"*** means it will work for All URLs

**How its work:** It will check **Username = Sam**, **Password = 123** and **Role = ROLE\_ADMIN**

**EmployeeController**

**@Controller**

**public** **class** EmployeeController {

@RequestMapping(value = "welcome")

**public** String getData(ModelMap map, Principal principal) {

String name = principal.getName();

map.addAttribute("key", name); **return** "Hello";

webapp/Home.jsp

<a href=*"welcome"*>Home</a>

Output/Hello.jsp

**UserName : ${key}**

# *Add LogOut Functionality*

Output/Hello.jsp

<form action="j\_spring\_security\_logout" id="logout" method="post">

<input type="hidden" name="${\_csrf.parameterName}" value="${\_csrf.token}" />

</form>

<a href="#" onclick="document.getElementById('logout').submit();">Logout</a>

WEB-INF/spring-security.xml

**Below <form-login/> tag**

**<**logout logout-success-url="/Home" logout-url="/j\_spring\_security\_logout"

invalidate-session="true" />

EmployeeController

@RequestMapping(value = "Home")

**public** String home() {

**return** "../Home";

Now Suppose we have links in Hello.jsp

<li><a href=*"java"*>Java</a></li> <li><a href=*"php"*>PHP</a></li> <li><a href=*"mysql"*>MySQL</a></li> and Respective methods in Controller

spring-security.xml

<http use-expressions=*"true"* auto-config=*"true"*>

<intercept-url pattern=*"/hello"* access=*"hasRole('ROLE\_ADMIN')"* />

<intercept-url pattern=*"/java"* access=*"hasRole('ROLE\_FOO')"*/>

<form-login />

</http>

<authentication-manager>

<authentication-provider>

<user-service>

<user name=*"Sam"* password=*"{noop}123"* authorities=*"ROLE\_ADMIN"* />

<user name=*"Pam"* password=*"{noop}456"* authorities=*"ROLE\_FOO"*/>

</user-service>

</authentication-provider>

</authentication-manager>

So, if we login with **SAM** we will see 3 links we can see details of **PHP, MySQL** but we should get **error for java** because Java has separate username and Pass with **separate Role**

***ROLE\_ANONYMOUS***: we can take any Role like **ROLE\_ADMIN, ROLE\_USER** or **ROLE\_FOO** these roles are not define anywhere in Spring but if I want to view some Page like **Home Page** spring should not ask for **Authentication** to users so we can use this role and it’s **predefined in Spring**

<intercept-url pattern=*"/\*\*"* access=*"hasRole('ROLE\_ANONYMOUS')"*/>

We must put this Intercept URL at Last from All otherwise it will consider all **request as Anonymous because of \*\***

Multiple Authority

<user name=*"Pam"* password=*"{noop}456"* authorities=*"ROLE\_FOO, ROLE\_ADMIN"*/>

***Custom Login Form Example***

webapp/Login.jsp

**<h3>Custom Login Page</h3>**

<%

String error = (String) request.getAttribute("error");

**if** (error != **null** && error.equals("true")){

out.println("<h4 style=\"color:red\">Invalid login credentials. Please try again!!</h4>");

}%>

<form action=*"login"* method=*'POST'*>

<table>

<tr><td>User:</td><td><input type=*'text'* name=*'username'*></td></tr>

<tr><td>Password:</td><td><input type=*'password'* name=*'password'* /></td></tr>

<tr>

<td><input name=*"submit"* type=*"submit"* value=*"submit"* /></td>

<td><input name=*"reset"* type=*"reset"* />

<input type=*"hidden"* name="${\_csrf.parameterName}" value="${\_csrf.token}" /></td>

</tr>

</table>

**</form>**

webapp/Home.jsp

<a href=*"login"*>Home</a>

EmployeeController

@RequestMapping(value = "Hello")

**public** String hello(ModelMap modelMap,Principal principal) {

String name = principal.getName();

modelMap.addAttribute("msg", "Welcome Custom Login");

modelMap.addAttribute("name", name);

**return** "Hello";

}

@RequestMapping("/error")

**public** String error(ModelMap model){

model.addAttribute("error", "true");

**return** "../Login";

}

@RequestMapping(value = "/login")

**public** String login(){

System.***out***.println("Inside login");

**return** "../Login";

}

@RequestMapping("/logout")

**public** String logout(){

**return** "logout";

}

webapp/Hello.jsp

Messagge : ${msg}<br>

UserName : ${name}

<form action=*"j\_spring\_security\_logout"* id=*"logout"* method=*"post"*>

<input type=*"hidden"* name=*"*${\_csrf.parameterName}*"* value=*"*${\_csrf.token}*"* />

</form>

<a href=*"#"* onclick="document.getElementById('logout').submit();">Logout</a>

WEB-INF/spring-security.xml

*<http auto-config="true" use-expressions="true">*

<intercept-url pattern=*"/welcome"* access=*"hasRole('ROLE\_ADMIN')"* />

<form-login login-page=*"/login"* authentication-failure-url=*"/error"* username-parameter=*"username"* password-parameter=*"password"* default-target-url=*"/Hello"*/>

<logout logout-success-url=*"/login"* logout-url=*"/j\_spring\_security\_logout"* invalidate-session=*"true"* />

</http>

**username-and-password-parameter** will take username and password attribute if we not define

Alternative of CSRF\_TOKEN **::** we need import a Jar file **spring-security-taglib then**

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-taglibs</artifactId>

<version>5.3.8.RELEASE</version>

</dependency>

Login.jsp

<%@ taglib uri=*"http://www.springframework.org/security/tags"* prefix=*"sec"* %> and add inside form tag <sec:csrfInput/>

**default-target-url=*"/Hello"*** if we want to send user to particular page whenever user Logged In

**authentication-failure-forward-url=*"/login?error=true"*** In some case**default-target-url** will not work so we can use this.

Login.jsp

<% String error = (String) request.getParameter("error");

**if** (error != **null** && error.equals("true")){

out.println("<h4 style=\"color:red\">Invalid login credentials. Please try again!!</h4>"); }%>

Authority Work:

<intercept-url pattern=*"/hello"* access=*"hasRole('ROLE\_ADMIN')"* />

<intercept-url pattern=*"/java"* access=*"hasRole('ROLE\_FOO')"* />

<intercept-url pattern=*"/php"* access=*"hasRole('ROLE\_BAR')"* />

<user name=*"Sam"* password=*"{noop}123"* authorities=*"ROLE\_ADMIN"* />

<user name=*"Pam"* password=*"{noop}456"* authorities=*"ROLE\_FOO,ROLE\_ADMIN"* />

<user name=*"Dam"* password=*"{noop}789"* authorities=*"ROLE\_ADMIN,ROLE\_FOO,ROLE\_BAR"* />

**Sam** can access only */hello* links

**Pam** can access */hello, /java* links

**Dam** can access */hello, /java, /php* links

Show Links on Conditions:If user is logged in show logout and Username

Hello.jsp

<%@ taglib uri=*"http://www.springframework.org/security/tags"* prefix=*"sec"*%>

<%@ taglib uri=*"http://java.sun.com/jsp/jstl/core"* prefix=*"c"*%>

<ul>

<li><a href=*"java"*>Java</a></li>

<li><a href=*"php"*>PHP</a></li>

<li><a href=*"mysql"*>MySQL</a></li>

**<sec:authorize access=*"authenticated"* var=*"authenticated"*></sec:authorize>**

<c:choose>

**<c:when test=*"*${authenticated}*"*>**

<li><sec:authentication property=*"name"* /></li>

<form action=*"j\_spring\_security\_logout"* id=*"logout"* method=*"post"*>

<sec:csrfInput/>

</form>

<li><a href=*"#"*

onclick="document.getElementById('logout').submit();">Logout</a></li>

</c:when>

<c:otherwise>

<li><a href=*"login"*>Login</a></li>

</c:otherwise>

</c:choose>

</ul>

***Custom Login Form with SQL Example***

First create table

create table user(uid int, uname varchar(120),password varchar(120), pass varchar(120),enabled boolean);

**alter table user add primary key(uid);**

create table userRole(id int, uname varchar(120), auth varchar(45));

**then....**

**In password field enter encrypted pass**

insert into user values(1,'Sam','$2a$10$sCrpbptlhsO9GLmLF4','.c2Ont0vhl89uwXUtBIIhU.A9c4o1wUqcFK',true);

insert into user values (2,'Pam', '$2a$10$zjCOFL0PNYecziJk24R3cOv','KFrQx5hAZqp9TlnPJgzVP3h1JkXHtO',true);

insert into user values(3,'Dam','$2a$10$Zkruufv5q7aeWWVk9BfI3OT','V19GGyiO7aTUS/weALrPHoJlPZHAse',true);

**then....**

insert into userRole values(1,'Sam','ROLE\_ADMIN');

insert into userRole values(2,'Pam','ROLE\_USER');

insert into userRole values(3,'Dam','ROLE\_USER');

Client.java **first Encrypt Password**

**public** **static** **void** main(String[] args) {

**int** i = 0;

**while** (i < 1) {

String password = "789";

BCryptPasswordEncoder passwordEncoder = **new** BCryptPasswordEncoder();

String hashedPassword = passwordEncoder.encode(password);

System.***out***.println(hashedPassword);

i++;

Pom.xml

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-jdbc</artifactId>

<version>5.3.8 </version>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>8.0.18</version>

</dependency>

WEB-INF/spring-security.xml

<beans:bean id=*"encoder"* class=*"org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder"*></beans:bean>

<beans:bean id=*"dataSource"*

class=*"org.springframework.jdbc.datasource.DriverManagerDataSource"*>

<beans:property name=*"driverClassName"* value=*"com.mysql.jdbc.Driver"* />

<beans:property name=*"url"* value=*"jdbc:mysql://localhost:3306/school"* />

<beans:property name=*"username"* value=*"root"* />

<beans:property name=*"password"* value=*""* />

</beans:bean>

<authentication-manager>

<authentication-provider>

<password-encoder ref=*"encoder"* />

<jdbc-user-service data-source-ref=*"dataSource"*

users-by-username-query=*"select uname,password,enabled from user where uname=?"*

authorities-by-username-query=*"select ur.uname,ur.auth from user as u INNER JOIN userrole as ur ON u.uname = ?"* />

</authentication-provider>

</authentication-manager>

**Might be we cannot store long text Encrypt password in password column So divide that pass into 2 parts and save in Column password and pass in DB**

users-by-username-query=*"select uname,CONCAT(password,pass) as password,enabled from user where uname=?"*

**Remember Me in Login**

Login.jsp

<td>Remember Me: <input type=*"checkbox"* name=*"remember-me"* /></td>

WEB-INF/spring-security.xml

<remember-me remember-me-parameter=*"remember-me"* remember-me-cookie=*"remember-me"*

token-validity-seconds=*"24"* data-source-ref=*"dataSource"* />

***Inside <http> tag***

MySQL

CREATE TABLE **persistent\_logins**(username **varchar(64)**,series **varchar(64)**,token **varchar(64)**,

last\_used **timestamp**, **PRIMARY KEY** (series));

# -----------: *Spring Security Java Configuration START HERE*: -----------

Considering above example but remove **Spring-security.xml** Config from **web.xml**

SecurityConfiguration.java

**import** org.springframework.context.annotation.Bean;

**import** org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;

**import** org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

**import** org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

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

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

@EnableWebSecurity

**public** **class** SecurityConfiguration **extends** **WebSecurityConfigurerAdapter**{

@Override

**protected** **void** configure(AuthenticationManagerBuilder auth) **throws** Exception {

auth.inMemoryAuthentication()

.withUser("Sam")

.password("$2a$10$sCrpbptlhsO9GLmLF4.c2Ont0vhl89uwXUtBIIhU.A9c4o1wUqcFK")

.roles("USER")

.and()

.withUser("Pam")

.password("$2a$10$zjCOFL0PNYecziJk24R3cOvKFrQx5hAZqp9TlnPJgzVP3h1JkXHtO")

.roles("ADMIN");

}

@Bean

**public** PasswordEncoder getPasswordEncoder() {

**return** **new** BCryptPasswordEncoder();

***Role Based Authorization***

EmployeeController.java

@RestController

**public** **class** EmployeeController {

@RequestMapping(value = "/")

**public** String home() {

**return** "<h1>Welcome</h1>";

}

@RequestMapping(value = "/user")

**public** String user() {

**return** "<h1>Welcome User</h1>";

}

@RequestMapping(value = "/admin")

**public** String admin() {

**return** "<h1>Welcome Admin</h1>";

SecurityConfiguration.java

Append this method and remember we have configured **2 users with different role**

@Override

**protected** **void** configure(HttpSecurity http) **throws** Exception {

http.authorizeRequests()

.antMatchers("/\*\*").hasRole("ADMIN")

.and().formLogin();

}

**/\*\*** for all URLs

**hasRole** for Single Role

**hasAnyRole** for Multiple Role and its take String[]

**Default logout page** just append logout at end of URL

**NOW** we can have **Home URLs** or **static/css** or **static/js** which we have included in our JSPs so that should be **accessible to all**

.antMatchers("/","static/css","static/js").permitAll()

***Assign particular Role***

SecurityConfiguration.java

@Override

**protected** **void** configure(HttpSecurity http) **throws** Exception {

http.authorizeRequests()

.antMatchers("/admin").hasRole("ADMIN")

.antMatchers("/user").hasRole("USER")

.antMatchers("/").permitAll()

.and().formLogin();

**Recommended** to put Higher role on TOP like above

Access URLs accordingly so we can see Admin cannot access USER and Vice-Versa

**hasAnyRole** .antMatchers("/user").hasAnyRole("USER","ADMIN")

***AuthenticationProvider***: has a method called authenticate(Authentication auth) as a input and return Authentication as a output also we can get principal object also using **getPrincipal()**

**A single app can have multiple AuthenticationProvider for different type Auth mechanism like Password auth, Oauth, or LDAP auth**

**SO** we have **multiple AuthenticationProvider** but we need one which can check which we need **like right now we need Password Auth** So we will tell this to **AuthenticationManager** and it ask to **AuthenticationProvider** do you support Password auth by calling support() method of each which is inside **AuthenticationProvider** and it will return **true** or **false**

***Authentication FLOW***

1. we will **Input credential** it gets bind into **Authentication Object**
2. **Authentication object** get transfer to **AuthenticationManager authenticate()** and it will check provider it can use we saw above
3. **AuthenticationProvider** use **UserDetailsService** and return DB stored credential and forward it to **AuthenticationManager** and it compare **Input and DB details** and if its success it will return **Authentication Object as output**
4. Now if we hit another URL, it will **not compare credential again** it will stored previously get **Authentication Object** **into session** and we have **separate Filter** for that to compare that Credential for **Every request**

***Java Based configuration using UserDetailsService interface (HARD-CODE)***

SecurityConfiguration.java

@EnableWebSecurity

**public** **class** SecurityConfiguration **extends** WebSecurityConfigurerAdapter{

@Autowired

MyUserDetailsService myUserDetailsService;

@Override

**protected** **void** configure(AuthenticationManagerBuilder auth) **throws** Exception {

auth.userDetailsService(**this**.myUserDetailsService);

}

@Bean

**public** PasswordEncoder getPasswordEncoder() {

**return** **new** BCryptPasswordEncoder();

MyUserDetailsService.java

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

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

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

**import** org.springframework.stereotype.Service;

@Service

**public** **class** MyUserDetailsService **implements** UserDetailsService{

**public** UserDetails loadUserByUsername(String username) **throws** UsernameNotFoundException {

**return** **new** MyUserDetails(username);

}

MyUserDetails.java

**Its like we are passing hardCode details of user not using DB YET**

**import** org.springframework.security.core.GrantedAuthority;

**import** org.springframework.security.core.authority.SimpleGrantedAuthority;

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

**public** **class** MyUserDetails **implements** UserDetails{

String userName;

**public** MyUserDetails(String userName) {

**this**.userName = userName;

}

**public** MyUserDetails() {

}

@Override

**public** Collection<? **extends** GrantedAuthority> getAuthorities() {

**return** Arrays.*asList*(**new** SimpleGrantedAuthority("ROLE\_USER"));

}

@Override

**public** String getPassword() {

**return** "$2a$10$sCrpbptlhsO9GLmLF4.c2Ont0vhl89uwXUtBIIhU.A9c4o1wUqcFK";

}

@Override

**public** String getUsername() {

**return** **this**.userName;

***Custom Login Form DAO with SQL Example JPA***

Pom.xml

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-entitymanager</artifactId>

<version>5.4.4.Final</version>

</dependency>

<dependency>

<groupId>org.springframework.data</groupId>

<artifactId>spring-data-jpa</artifactId>

<version>2.2.2.RELEASE</version>

</dependency>

AppConfig.java Contains ***JPA*** configuration file

LoginUser.java

@Entity

@Table(name = "user")

**public** **class** LoginUser{

@Id

@GeneratedValue(strategy = GenerationType.***IDENTITY***)

Integer uid;

String uname;

String password;

String pass;

**boolean** enabled;

p2.LoginRepository.java

LoginUser findByuname(String uname);

MyUser.java

**public** **class** MyUser **implements** UserDetails{

LoginUser loginUser;

**public** MyUser(LoginUser loginUser) {

**this**.loginUser = loginUser;

}

**public** Collection<? **extends** GrantedAuthority> getAuthorities() {

**return** Arrays.*asList*(**new** SimpleGrantedAuthority("ROLE\_USER"));

}

**public** String getPassword() {

**return** **this**.loginUser.getPassword() + loginUser.getPass();

}

**public** String getUsername() {

**return** **this**.loginUser.getUname();

}

MyUserDetailService.java

@Service

**public** **class** MyUserDetailService **implements** UserDetailsService{

@Autowired

LoginRepository loginRepository;

**public** UserDetails loadUserByUsername(String username) **throws** UsernameNotFoundException {

LoginUser loginUser = **this**.loginRepository.findByuname(username);

**return** **new** MyUser(loginUser);

SecurityConfiguration.java will be same as above

***UserDetailsManager Interface***

While we implement**UserDetailsService** we can **loadUserByUsername** but if we want to perform **CRUD** operation on user using spring-security we can use this interface.

Create Table first and its structure must like this only

create table **users** (uid int,username varchar(45),password varchar(90),enabled boolean);

create table **authorities** (aid int,username varchar(45),authority varchar(45));

Insert

insert into **users** values (1,'Sam',' $2a$10$sCrpbptlhsO9GLmLF4.c2Ont0vhl89uwXUtBIIhU.A9c4o1wUqcFK',1);

insert into **authorities** values(1,'Sam','USER');

LoginUser.java

**public** **class** LoginUser **implements** UserDetails{

String username;

String password;

**public** String getUsername() {

**return** username;

}

**public** **void** setUsername(String username) {

**this**.username = username;

}

**public** String getPassword() {

**return** password;

}

**public** **void** setPassword(String password) {

**this**.password = password;

}

**public** Collection<? **extends** GrantedAuthority> getAuthorities() {

**return** Arrays.*asList*(**new** SimpleGrantedAuthority("USER"));

}

SecurityConfiguration.java

@Autowired

DataSource dataSource;

@Bean

**public** UserDetailsManager userDetailsService() {

**return** **new** JdbcUserDetailsManager(dataSource);

}

@Override

**protected** **void** configure(HttpSecurity http) **throws** Exception {

http.authorizeRequests()

.antMatchers("/user").hasAuthority("USER")

.antMatchers("/admin").hasAnyAuthority("ADMIN","USER")

.antMatchers("/hello").permitAll()

.and().formLogin();

}

Till now we have fetch user using UserDetailsManager now we will create new user

CreateUser.jsp

<form action=*"createUser"* method=*"POST"*>

UserName <input type=*"text"* name=*"username"*> <br>

Password <input type=*"text"* name=*"password"*> <br>

<sec:csrfInput/>

<input type=*"submit"*>

</form>

EmployeeController.java

@Autowired

UserDetailsManager userDetailsManager;

@Autowired

PasswordEncoder passwordEncoder;

@RequestMapping(value = "/createUser",method = RequestMethod.***POST***)

**public** **void** createUser(@ModelAttribute LoginUser loginUser) {

loginUser.setPassword(**this**.passwordEncoder.encode(loginUser.getPassword()));

**this**.userDetailsManager.createUser(loginUser);

}

***AuthenticationProvider Interface***

As we know it check our login credential behind the scene and return Authentication object, we can customize it’s also, Below Example will show how its work internally.

CustomAuthenticationProvider.java

@Component

**public** **class** CustomAuthenticationProvider **implements** AuthenticationProvider{

@Autowired

UserDetailsService userDetailsService;

@Autowired

PasswordEncoder passwordEncoder;

**public** Authentication authenticate(Authentication authentication)

**throws** AuthenticationException {

//authentication we will get details user input in HTML form

String rawPassword = authentication.getCredentials().toString();

UserDetails userDetails = **this**.userDetailsService.loadUserByUsername(authentication.getName());

**if**(userDetails != **null** && **this**.passwordEncoder.matches(rawPassword, userDetails.getPassword())) {

**return** **new** UsernamePasswordAuthenticationToken(userDetails.getUsername(), userDetails.getPassword());

}

**return** **null**;

}

**public** **boolean** supports(Class<?> authentication) {

**return** UsernamePasswordAuthenticationToken.**class**.equals(authentication);

}

}

***Custom Login Form DAO with SQL Example plain XML***

Pom.xml

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>5.3.10.Final</version>

</dependency>

P1.model.UserRole

**int** urid;

String role; Getters and Setters

P1.model.User

**int** uid;

String uName;

String password;

**private** **boolean** enabled;

Set<UserRole> role; Getters and Setters

Employee.hbm.xml

Declare One to Many with user and userRole and for Boolean datatype in like enabled

<property name=*"enabled"* type = *"java.lang.Boolean"*>

<column name=*"enabled"* sql-type=*"tinyint"*/>

Hibernate.cfg.xml **will be same as usual**

Client.javaadd some value in both table byConfiguring in mainClass and **PASSWORD MUST BE EnCrypted**

P1.model.UserDAO **(Interface)**

User findByUserName(String uName);

P1.model.UserDAOImpl **implements UserDao**

**public** User findByUserName(String uName) {

Session s = sf.openSession();

Query<User> q = s.createQuery("from User where uName = :n",User.**class**);

q.setParameter("n", uName);

User user = q.uniqueResult();

System.***out***.println(">>> " + user);

**return** user;

P1.model. MyUserDetailsService

**public** **class** MyUserDetailsService **implements** UserDetailsService{

UserDao userDao;

**public** UserDao getUserDao() {

**return** userDao;

}

**public** **void** setUserDao(UserDao userDao) {

**this**.userDao = userDao;

}

**public** UserDetails **loadUserByUsername**(String username) **throws** UsernameNotFoundException {

User user = **this**.getUserDao().findByUserName(username);

List<GrantedAuthority> authorities = buildUserAuthority(user.getRole());

**return** buildUserForAuthentication(user, authorities);

}

**private** List<GrantedAuthority> **buildUserAuthority**(Set<UserRole> userRoles) {

Set<GrantedAuthority> setAuths = **new** HashSet<GrantedAuthority>();

**for** (UserRole userRole : userRoles) {

setAuths.add(**new** SimpleGrantedAuthority(userRole.getRole()));

}

List<GrantedAuthority> Result = **new** ArrayList<GrantedAuthority>(setAuths);

**return** Result;

}

@SuppressWarnings("unused")

**private** ***org.springframework.security.core.userdetails.User*** **buildUserForAuthentication**(User user,List<GrantedAuthority> authorities) {

org.springframework.security.core.userdetails.User **sUser** = **new**

***org.springframework.security.core.userdetails.User*(user.getuName(), user.getPassword(), user.isEnabled(),true,true,true,authorities);**

**// Single Object**

**return** **sUser**;

WEB-INF/spring-security.xml

<authentication-manager>

<authentication-provider user-service-ref=*"myUserDetailsService"*>

<password-encoder ref=*"passwordEncoder"*></password-encoder>

</authentication-provider>

</authentication-manager>

<beans:bean id=*"UserDaoImpl"* class=*"p1.model.UserDaoImpl"*></beans:bean>

<beans:bean id=*"myUserDetailsService"* class=*"p1.model.MyUserDetailsService"*>

<beans:property name = *"userDao"* ref=*"UserDaoImpl"*></beans:property>

</beans:bean>

**Exceptions can come** **ContextLoaderListener ClassNotFoundException**

Right Click on Project ->Deployment Assembly -> Add -> Add Maven Dependencies