Spring Security -using Java Configuration

In Previous exercise, you understood how to configure spring security using xml. In this lab, you will understand how to do the same using java configuration

In this lab, u will be working on 02mvc-security-javaconfig-start project

1) Do you remember that we have configured DelegatingFilterProxy in web.xml in our step 1 while using xml configuration ? Below was the configuration :

```
<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>
```

Now how to configure DelegatingFilterProxy using java Configuration?

You just have to write a subclass of AbstractSecurityWebApplicationInitializer as shown below :

```
public class SecurityInitializer extends AbstractSecurityWebApplicationInitializer {


}
```

2) We will configure all our security beans in a class SecurityConfig.

Open SecurityConfig.java and observe that it is annotated with @Configuration

In Xml, to enable all the default filters, we used `<security:http auto-config="true" >`

In JavaConfiguration, the equivalent is @EnableWebSecurity .

Complete TODO -1 in SecurityConfig.java

If you want to customize defults. you have to extend WebsecurityConfigurerAdapter

Complete TODO-2 in SecurityConfig.java

WAY2LEARN

3) If you remember, we have configured authentication manager in xml as shown below :

```
<security:authentication-manager >
	<security:authentication-provider>
		<security:user-service>
			<security:user name="siva" password="secret" authorities="ROLE_ADMIN"/>
		</security:user-service>
	</security:authentication-provider>
</security:authentication-manager>
```

If you want to do the same in java configuration, write an @Autowired method as shown below in SecurityConfig.java

```
@Autowired
	protected void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
		auth.inMemoryAuthentication()
			//.passwordEncoder(new ShaPasswordEncoder())
			.withUser("siva").password("secret").roles("ADMIN")
			.and()
			.withUser("prasad").password("secret").roles("USER");

	}
```

4) In xml, we have configured intercept urls and form login as shown below :

```
<security:http auto-config="true" use-expressions="true">
		<security:intercept-url pattern="/**" access="hasRole('ROLE_ADMIN')"/>
		<security:form-login/>
	</security:http>
```

If you want to do the same in java configuration, override a method as shown below :

```
	protected void configure(HttpSecurity http) throws Exception {
		http.authorizeRequests()
					.antMatchers("/login**").permitAll()
					.antMatchers("/s/**").hasRole("USER")
					.anyRequest().authenticated()
				.and()
					.formLogin()


	}
```

We dont want security for static resources. So, override as shown below :

```
@Override
	public void configure(WebSecurity web) throws Exception {
		web.ignoring().antMatchers("/resources/**");
	}
```

5) Open WebInitializer.java and observe that we have already SecurityConfig as RootConfigClass

6) Deploy the application and observe the you will be asked to login.
Login with credentials you configured in Step 3.

You will be taken to Home Page. Click on Logout link and observe that you are shown with login page again after logging out

7) In earlier step, we have used a form based authentication.

Can you reconfigure it to use basic authentication?

8) Now we want to use formbased authentication again. So, switch back to form based authentication.

We want to use out customized login page. Do, you remember how we customized in xml ? It looks like below :

```
<security:form-login login-page="/login/form"
	authentication-failure-url="/login/form?error"
	login-processing-url="/login" username-parameter="username"  password-
	parameter="password"  default-target-url="/home.htm" />
```

If you want to do the same in java configuration, it can be done as below

```
protected void configure(HttpSecurity http) throws Exception {
		http.authorizeRequests()
					.antMatchers("/login**").permitAll()
					.antMatchers("/x/**").hasRole("ADMIN")
					.anyRequest().authenticated()
				.and()
					.formLogin()
						.loginPage("/login").usernameParameter("myusername")

	.passwordParameter("mypassword").loginProcessingUrl("/mylogin")
						.failureUrl("/login?error")
				.and()
					.logout().logoutUrl("/logout");
					//.logoutRequestMatcher(new AntPathRequestMatcher("/logout",
"GET"));


	}
```

Observe how we have configured logout url also

9)Open MvcConfig.java and complete TODO-3 by adding a view controller for /login

10) Open /WEB-INF/jsp/login.jsp. Understand the code

11) Open /WEB-INF/jsp/commons/topMenu.jsp. complete TODO -4 by writing code for

login and logout links as below :

```
<security:authorize access="authenticated" var="isAuthenticated" />
<c:if test="${!isAuthenticated}">
	<c:url var="loginurl" value="/login" />
	<a id="signInLink" class="btn btn-green btn-login" href="${loginurl}">Sign In</a>
</c:if>



<c:if test="${isAuthenticated}">
	<c:url var="logouturl" value="/logout" />
	Welcome  <security:authentication property="name"/>
<form action="${logouturl}" method="post">
	<input type="submit" value="Log out"  class="btn btn-green btn-login" />
	<input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}"/>
</form>
</c:if>
```

12) **Deploy the application and observe that login and logout functionality is working as expected**

Congratulations !! U have configured spring security using java configuration

WAY2LEARN