

Spring Security Application

Spring Security is a spring framework for security. It is implemented using servlet filters in the background.

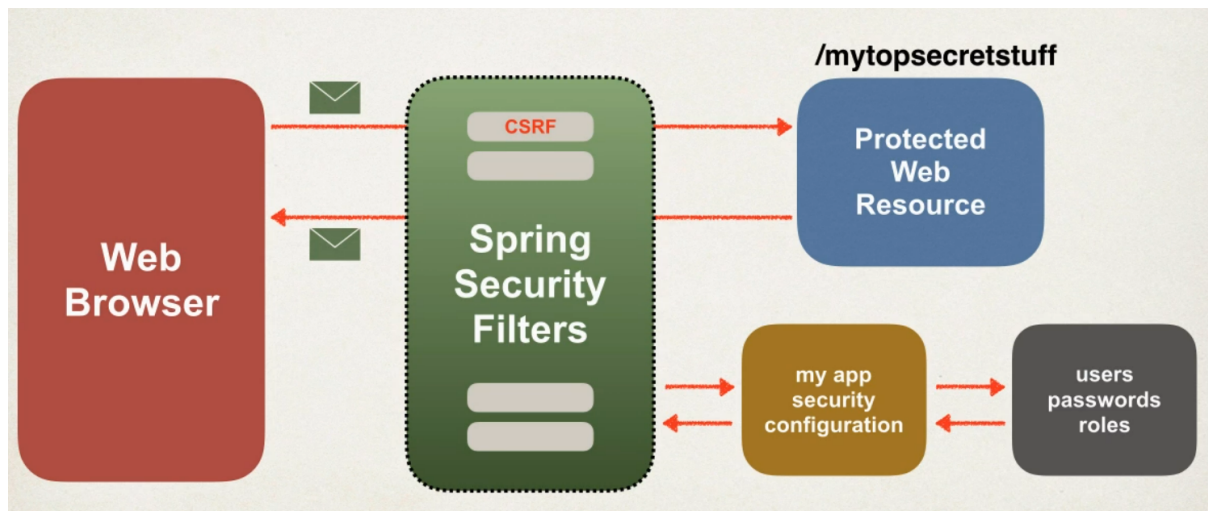
There are two ways to secure spring webapps-

- i. Declarative Security
- ii. Programmatic Security

In Declarative Security we define application security constraints in configuration I.e., all java config (@Configuration, no XML) or spring XML config. It provides separation of concerns between application code and security.

In programmatic Security spring security provides an api for custom application coding and it provides greater customization for specific application requirements.

CSRF (Cross-site-request-forgery) protection



Spring security protects against CSRF(Cross-site-request-forgery). It is a security attack where maybe an evil website tricks you into executing an action on a web application while you are logged in.

For instance, you are logged in your bank portal so you can be tricked into transferring money to others.

To protect against CSRF attacks there are security features you can add in your application, such as:

-

- Embed additional authentication data/token into all HTML forms.
- On subsequent requests, the web app will verify the token before processing.

CSRF protection is enabled by default in spring security as it uses synchronization token pattern where each session includes a session cookie and randomly generated token. For request processing, spring security verifies the token before processing. All of this is handled by spring security filters.

Just include CSRF token in your .jsp file form submission section: -

`<form: form>` automatically adds CSRF token

Or if you are using normal `<form>` you must manually add CSRF token

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

Spring Security Application

Let's create a basic project where we can use spring security features in our spring web app.

Required Dependencies

- spring-webmvc
- javax.servlet-api
- javax.servlet.jsp-api
- jstl
- spring-security-web
- spring-security-config
- add maven war plugin
- spring-security-taglibs

Note: -

We will not use web.xml and servlet.xml files for configuration rather we will use `@Configuration` annotation, Spring Dispatcher Servlet Initializer and to provide mvc support to our application we will use `@EnableWebMvc` annotation.

Basic development process (no xml--web.xml and servlet.xml)

- Add maven dependencies for maven Spring MVC app
- Create Spring app configuration (`@Configuration`)
- Create Spring Dispatcher Servlet Initializer
- Develop Spring Controller
- Develop JSP page for view

Context root is the root of your web application. It will be the name which you want to be displayed in your localhost address. To change the name: -

go to project>properties>web project settings>context root--change the name

1. Import a demo project from git.
2. Open pom.xml file and add all the above dependencies.
3. Create a package for config and class AppConfig with `@Configuration`, `@EnableWebMvc` and `ComponentScan(basePackages="package name")`. In this class, we will create a method for viewResolver with `@Bean` and use `InternalResourceViewResolver` and its objects to create prefixes and suffixes.
4. In the config package, create a class `AppDispatcherServletInitializer` and make it a super class of `AbstractAnnotationConfigDispatcherServletInitializer`.
Add below codes in mentioned methods: -

```
return new Class [] {configurationclass_name.class} in getServletConfigClasses() method  
return new String [] {""} in getServletMappings() method
```

5. For security, create a class name - `SecurityWebApplicationInitializer` extending superclass `AbstractSecurityWebApplicationInitializer`.

Spring Security Application

6. Create class AppSecurityConfig extending superclass WebSecurityConfigurerAdapter. Annotate this class with @Configuration and @EnableWebSecurity. Go to source>override implement methods>select configure AuthenticationManagerBuilder) Remove all default code and replace it with the below: -

```
//add users for in memory authentication
UserBuilder users = User.withDefaultPasswordEncoder();
auth.inMemoryAuthentication()
.withUser(users.username("Kavya").password("root").roles("Founder"))
.withUser(users.username("Devendra Rao").password("root").roles("Co-Founder"))
.withUser(users.username("Ata B").password("root").roles("Co-Founder"));
```

7. Create a controller class and define methods with mapping in it.
8. Create a .jsp file with the name returned in your controller method.

Following the above, we have our basic spring web app project. Let's beautify and secure this.

Adding Custom Login Form: -

1. Update AppSecurityConfig class
go to the class go to override/implement methods and select configure(HttpSecurity)
Remove the default code and replace it with the below-

```
http.authorizeRequests()
.anyRequest().authenticated().and().formLogin()
.loginPage("/showLoginPage")
.loginProcessingUrl("/authenticateTheUser")
.permitAll();
```

2. Create a new controller class name LoginController

```
@GetMapping("/showLoginPage") //same as name given in AppSecurityConfig class method
return the .jsp file name in the method body
```

3. Create the .jsp file mentioned in the controller in the view folder and create a form there and add css and bootstrap to beautify your form.
4. Now to include logout button-

```
add .logout().permitAll() in AppSecurityConfig class method [configure(HttpSecurity)]
```

User roles-

Dependency required is spring-security-taglibs for spring security taglibs support

In home.jsp: -

```
add <%@ taglib prefix="security" uri="http://www.springframework.org/security/tags" %>
add <c:if> for logout code in login.jsp
<p>
User: <security:authentication property="principal.username" />
<br><br>
Role(s): <security:authentication property="principal.authorities" />
</p>
```

Spring Security Application

Restrict access-

If you want to show specific information to specific roles, then using this feature of spring security you can restrict access.

Development process

- create supporting controller code with view pages
- update user roles
- restrict access based on roles

Create method for in controller class with mapping and return with .jsp file name. Create that jsp file in the view folder and write the required info which only specific people will see.

go to the AppSecurityConfig file and in the remove method. request (). authentication () and replace them with .antMatchers("/").hasRole("ROLE") where you map your HTTP request name in antMatchers() and write the role in hasRole().

Access Denied-

go to AppSecurityConfig file and in the method add

.exceptionHandling().accessDeniedPage("/access-denied"). Map a controller method with .jsp for access denied. Create a .jsp for access denied and write some code.

Display content to specific people-

Add <security:authorize access="hasRole("")"></security:authorize> where you have hidden data for specific people and who you want can access that update their role in hasRole().

Adding JDBC connection-

Till now we are hardcoding values like people and roles in our project. So, let's connect our application with a database to ease up our work.

Development Process-

- Develop SQL script to setup database tables
 - Add database support to maven POM file
 - Create JDBC properties
 - Define DataSource in Spring Configuration
 - Update Spring security configuration to use JDBC
1. Import sql scripts from Git.
 2. Add jdbc properties in application.properties
 3. Update App.config class, see [What to change for JDBC connection in Spring Security](#) for detailed reference.
 4. While using jdbc authentication no need to hardcode user roles.
instead replace user roles code and use
auth.jdbcAuthentication().dataSource(DataSource
object).

Password encryption

Run sql schema which have encrypted password, modify DDL for password field, length should be 68
modify database properties file to point to new database schema


Encrypt passwords from: <https://www.bcryptcalculator.com/encode> and update the encrypted password in db.


Spring Security Application

Refer Github for code:- <https://github.com/Leaf-Co-Kb/User-Management-SpringSecurity>

Below are Screenshots of output-

Sign In


 Kavya




Login

Sign In

Invalid username and password.


 username


 password

Login

Sign In

You have been logged out.

 username

 password

Login