



INDEX

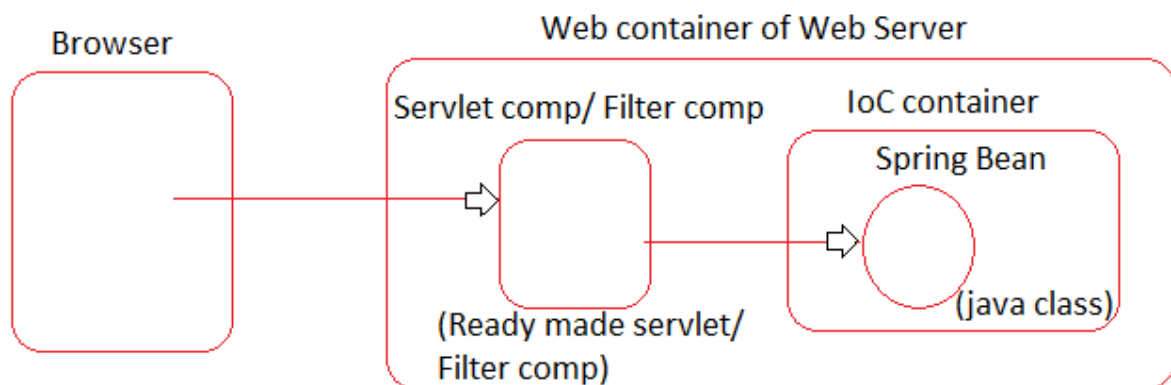
Spring Security -----

1. Introduction [04](#)
2. Security in Web Application [05](#)
 - a. Spring security Web application using Annotation configuration [08](#)
 - b. Spring security Web application using Spring Boot [21](#)
3. LDAP Server [24](#)

Spring Security

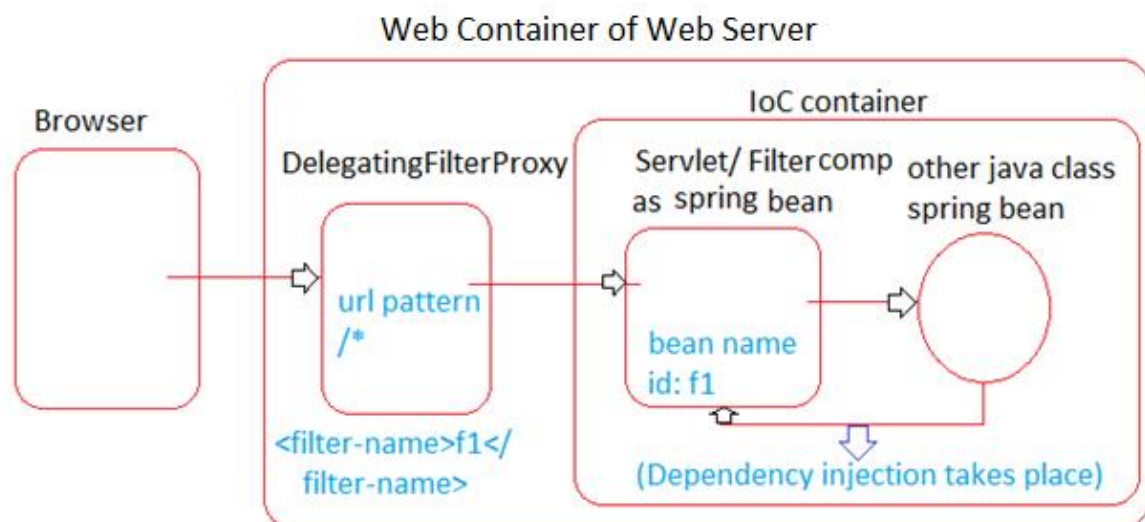
Introduction

Problem:



- Here Spring Bean cannot be injected to Servlet/ Filter comp because the servlet component is not taken as Spring Bean in IoC container.
- If move Servlet/ filter component as Spring bean to IoC container then the other Spring bean can be injected to Servlet/ Filter component but the Servlet/ Filter component cannot take request from browser.

Solution: Use DelegatingFilterProxy as mediator comp



Note: Here DelegatingFilterProxy (Spring MVC supplied pre-defined filter component) takes the request from browser on behalf of Servlet component/ Filter component and delegates the request to that Servlet component/ Filter component that is there in IoC container having "DelegatingFilterProxy" logical name as the bean id.

Security in Web Application

- ✚ It is all about performing authentication and authorization.
- ✚ Authentication is all about checking the identity of a user.
- ✚ Authorization is all about checking the access permissions of use on various services/ resources of the web application.

Note: Always provide access permissions to users based on roles of the users, not based on usernames itself roles are like designation.

<u>username</u>	<u>password</u>	<u>role</u>
raja	rani	ROLE_CLERK
ramesh	hyd	ROLE_MANAGER
suresh	hyd	ROLE_CLERK, ROLE_MANAGER

```
if (username.equals("raja jani")) {  
    allow to loan module  
} (Providing access permissions based on the usernames is bad  
practice)
```

```
if (role.equals("ROLE_CLERK")) {  
    allow to loan module  
} (Good practice)
```

To provide Security for every Web Application we need two major components:

1. Authentication provider/ Authentication Info Provider

- It is the repository/ memory where username, password and roles will be stored and managed,
 - InMemory DB (RAM) (not recommended)
 - XML file
 - Properties file DB s/w
 - LDAP server (Lightweight Directory Access Protocol) (best)

Note: LDAP Server, there is no possibility to recollect password when we forgot the password. The Only possibility is resetting the password.

2. Authentication Manager

- It is the component where given usernames, passwords will be verified with already registered usernames, passwords of Authentication info Provider. This component also takes authorization (checking the access permissions).

Different approaches of implementation Security in Web Applications:

a. Programmatic Approach

- It makes to develop Authentication Manager component manually either as servlet comp/filter component.
- This increase burden on the programmer (not recommended).

b. Declarative Approach

- Here underlying Server/ Framework/ container provides readymade Authentication Manager and we need activate through xml or annotation configurations.

Note:

- ✓ Java Web servers/ Application servers given built-in Authentication Manager as middleware service that can be activated/ configured by writing entries in web.xml file [This security is called Web Server/ Web Container managed Declarative Security].
- ✓ Spring Security module also provides built-in Authentication manager having no link with underlying Web server or Application server security [This is called Spring Security declarative management and this can be activated by importing "security namespace" in Spring Bean configuration file also by adding Spring Security libraries.

Limitations with Server Managed Declarative Security:

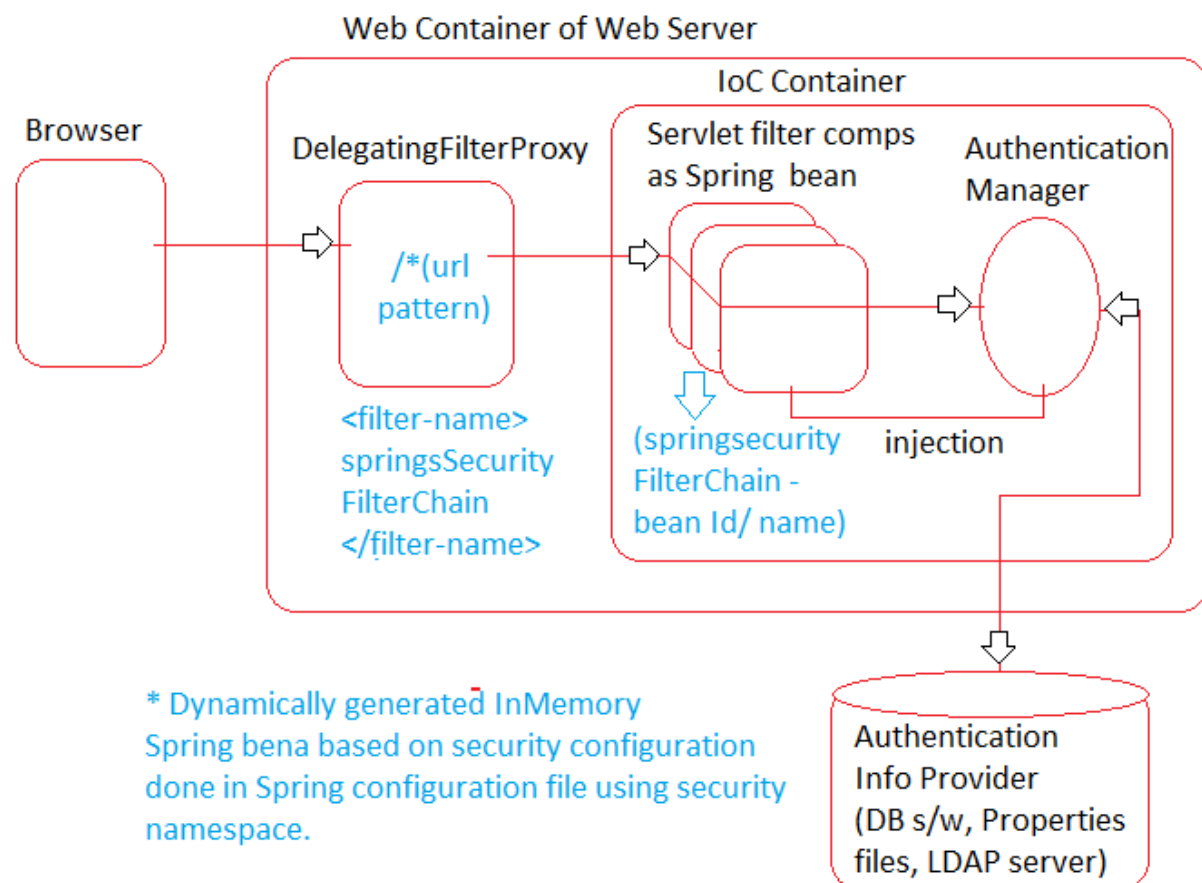
- a. Writing entries in web.xml to enable security is very complex.
- b. Some servers do not support this Declarative Security. Sometimes we need to purchase costly Application servers for only security feature.
- c. Most of the java Web servers/ Application servers do not allow us to take LDAP server as the Authentication provider.
- d. The entries we write in web.xml for security management are not portable across the multiple servers.

To overcome these problems, use Spring Security. The advantages are,

- a. We can use Spring Security in any Web server/ Application server irrespective of whether it supports its own declarative security or not.
- b. This can be applied on spring/spring boot MVC applications and also on non-spring web applications (like JSF web applications, Servlet/ JSP web applications and etc.).
- c. The security entries we write in Spring Bean configuration file are very much portable across the multiple servers/ java frameworks/ technologies.

- d. Supports different Authentication providers including LDAP server.
- e. Gives multiple advanced configurations like "remember me", "max concurrent user" and etc.

Note: When security name space to Spring Bean configuration file the IoC container creates so many pre-defined filters as Spring beans injected Authentication Manager bean. So, to take requests to delegate requests to this filter acting as Spring beans we need to use "DelegatingFilterProxy".



Authentication modes:

- BASIC
- DIGEST
- FORM
- CLIENT-CERT

We can develop spring Security pass in 4 ways:

- a. Using XML driven configuration
- b. Using Annotation driven configuration
- c. Using 100% code driven configuration
- d. Using Spring Boot configuration

Spring security Web application using Annotation configuration

MVCAnnoProj12-LocaleApp is having two pages

|--> Page 1: home page (Permit All)

|--> Page 2: show languages page (Allow only ADMIN,
MANAGER role users)

(Indirectly enable authentication on this page)

- a. Add the spring security dependencies (spring-security-web, spring-security-config on the top of regular jars/ dependencies) to build.gradle.
- b. Develop Spring bean configuration file having name security-beans.xml importing security namespace and link that file to applicationContext.xml.

Note:

- ✓ Spring Boot, Spring Security and etc. are extension modules spring, they are not part of basic spring framework. So, they have their own versions.
 - Spring Boot latest version: 2.3.4
 - Spring Security latest version: 5.4.x and etc.
- ✓ We must give encrypted password by using b-crypt algorithm. Since not giving so we should place {noop} before the password.
- c. Add "DelegatingFilterProxy" as filter component configuration in web.xml.
(make sure that it having fixed filter-name: springSecurityFilterChain).

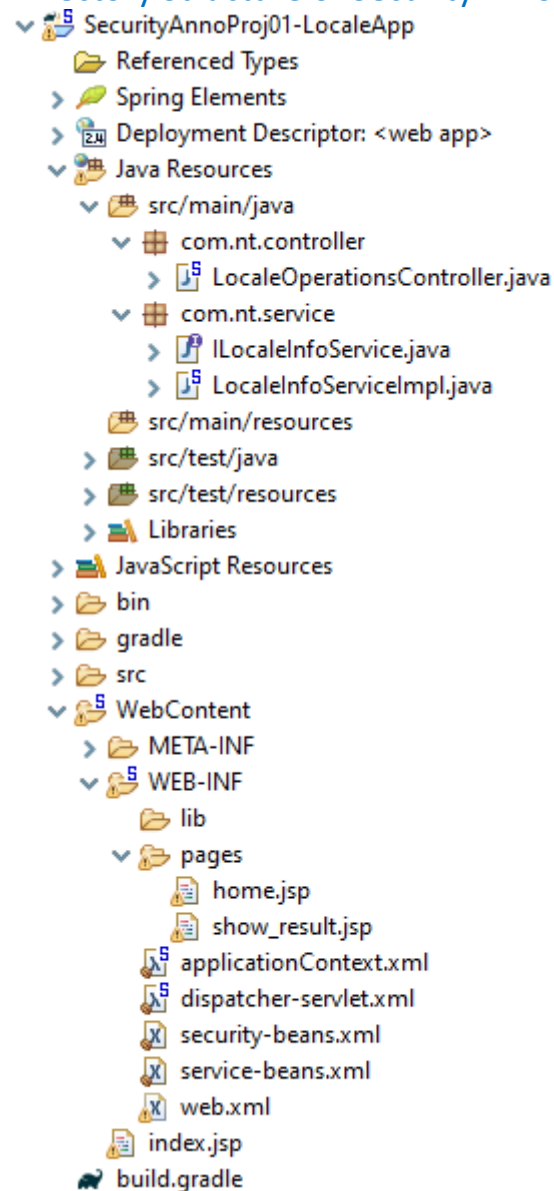
Note: we need to place this configuration, along with DispatcherServlet configuration.

- d. Run the application.

In security environment:

- 401: Authentication failed
- 403: Authorization failed

Directory Structure of SecurityAnnoProj01-LocalApp:



- Develop the below directory structure and folder, package, class, XML, JSP, file after convert to web application and add the jar dependencies in build.gradle file then use the following code with in their respective file.

ILocaleInfoService.java

```
package com.nt.service;

import java.util.Set;

public interface ILocaleInfoService {

    public Set<String> getAllCountries();

}
```

LocaleInfoServiceImpl.java

```
package com.nt.service;

import java.util.Locale;
import java.util.Set;
import java.util.TreeSet;

import org.springframework.stereotype.Service;

@Service("localeService")
public class LocaleInfoServiceImpl implements ILocaleInfoService {

    @Override
    public Set<String> getAllCountries() {
        Locale locales[] = null;
        Set<String> countriesSet = null;
        //get all locales
        locales = Locale.getAvailableLocales();
        countriesSet = new TreeSet<>();
        //copy all contries to List collection
        for (Locale l : locales) {
            if (!l.getDisplayCountry().equals("")) {
                countriesSet.add(l.getDisplayCountry());
            }
        }
        return countriesSet;
    }
}
```

LocaleOperationsController.java

```
package com.nt.controller;

import java.util.Map;
import java.util.Set;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

import com.nt.service.ILocaleInfoService;

@Controller
public class LocaleOperationsController {
```

```

@Autowired
private ILocaleInfoService service;

@RequestMapping("/welcome")
public String showHomePage() {
    return "home";
}

@RequestMapping("/countries")
public String fetchCountries(Map<String, Object> map) {
    Set<String> countriesSet = null;
    //use Service
    countriesSet = service.getAllCountries();
    //add object to mav object
    map.put("listInfo", countriesSet);
    map.put("operation", "countries");
    //return MAV
    return "show_result";
}
}

```

home.jsp

```

<h1 style="color: red; text-align: center">Welcome to Locale App</h1>
<h1 style="color: red; text-align: center">Home</h1>
<h2 style="color: cyan; text-align: left">
    <a href="countries">All Countries</a>
</h2>

```

service-beans.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"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-4.3.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-4.3.xsd">
    <context:component-scan base-package="com.nt.service"/>
</beans>

```

security-beans.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:security="http://www.springframework.org/schema/security"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/context
    http://www.springframework.org/schema/context/spring-context-4.3.xsd"
    http://www.springframework.org/schema/security
    http://www.springframework.org/schema/security/spring-security-5.4.xsd">

  <security:http use-expressions="true">
    <!-- Specify Access restrictions on the URLs -->
    <security:intercept-url pattern="/welcome"
access="permitAll"/>
    <security:intercept-url pattern="/countries"
access="hasAnyRole('ROLE_ADMIN', 'ROLE_MANAGER')"/>
    <!-- Enable Basic logic -->
    <security:http-basic/>
  </security:http>

  <security:authentication-manager>
    <!-- Taking current Spring bean configuration
file itself as authentication info provider -->
    <security:authentication-provider>
      <security:user-service>
        <security:user name="raja"
password="{noop}rani" authorities="ROLE_MANAGER"/>
        <security:user name="rajesh"
password="{noop}hyd" authorities="ROLE_ADMIN"/>
        <security:user name="karan"
password="{noop}kiran" authorities="ROLE_CUSTOMER"/>
        <security:user name="lokesh"
```

```

password="{noop}delhi" authorities="ROLE_VISITOR"/>
    </security:user-service>
</security:authentication-provider>
</security:authentication-manager>

</beans>

```

dispatcher-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"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-4.3.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-
        4.3.xsd">

    <!-- Configure Handler mapping -->
    <bean
class="org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping"/>

    <!-- Configure View Resolver -->
    <bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver"
">
        <property name="prefix" value="/WEB-INF/pages/">
        <property name="suffix" value=".jsp"/>
    </bean>

    <context:component-scan base-package="com.nt.controller"/>

</beans>

```

applicationContext.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"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-4.3.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-4.3.xsd">

```

```
https://www.springframework.org/schema/beans/spring-beans-4.3.xsd
http://www.springframework.org/schema/context
https://www.springframework.org/schema/context/spring-context-
4.3.xsd">
```

```
<import resource="security-beans.xml"/>
<import resource="service-beans.xml"/>
```

```
</beans>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app>
  <!-- Bootstraps the root web application context before servlet
initialization -->
  <listener>
    <listener-
class>org.springframework.web.context.ContextLoaderListener</listener-
class>
  </listener>

  <!-- The front controller of this Spring Web application, responsible
for handling all application requests -->
  <servlet>
    <servlet-name>dispatcher</servlet-name>
    <servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <load-on-startup>2</load-on-startup>
  </servlet>

  <!-- Map all requests to the DispatcherServlet for handling -->
  <servlet-mapping>
    <servlet-name>dispatcher</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>

  <!-- Filter 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>

  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
</web-app>

```

show_result.xml

```

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1" isELIgnored="false"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn"%>
<h1 style="color: red; text-align: center">Welcome to Locale App</h1>
<h1 style="color: red; text-align: center">Result <c:out
value="\${operation}"/></h1>
<c:choose>
  <c:when test="\${!empty listInfo && listInfo ne null}">
    <c:forEach var="country" items="\${listInfo}">
      <table bgcolor="pink" align="center">
        <tr>
          <td style="color: blue;">\${country}</td>
        </tr>
      </table>
    </c:forEach>
  </c:when>
  <c:otherwise>
    <h1 style="color:red; text-align: left;">No countries found</h1>
  </c:otherwise>
</c:choose>
<h3>Countries count : <c:out value="\${fn:length(listInfo)}"/></c:out> </h3>
<a href="welcome">Home</a>

```

index.jsp

```

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<jsp:forward page="welcome"/>

```

build.gradle

```
plugins {  
    // Apply the java-library plugin to add support for Java Library  
    id 'war'  
    id 'eclipse-wtp'  
}  
  
webAppDirName='WebContent'  
  
repositories {  
    jcenter()  
}  
  
dependencies {  
    // https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api  
    implementation group: 'javax.servlet', name: 'javax.servlet-api',  
version: '4.0.1'  
    // https://mvnrepository.com/artifact/org.springframework/spring-  
webmvc  
    implementation group: 'org.springframework', name: 'spring-  
webmvc', version: '5.2.8.RELEASE'  
    // https://mvnrepository.com/artifact/javax.servlet/jstl  
    implementation group: 'javax.servlet', name: 'jstl', version: '1.2'  
    //  
https://mvnrepository.com/artifact/org.springframework.security/spring-  
security-web  
    implementation group: 'org.springframework.security', name: 'spring-  
security-web', version: '5.4.1'  
    //  
https://mvnrepository.com/artifact/org.springframework.security/spring-  
security-config  
    implementation group: 'org.springframework.security', name: 'spring-  
security-config', version: '5.4.1'  
}
```

To disable or lock the user:

```
<security:user name="raja" password="{noop}rani"  
authorities="ROLE_MANAGER" disabled="true"/>  
    <security:user name="rajesh" password="{noop}hyd"  
authorities="ROLE_ADMIN" locked="true"/>
```


To use form login instead basic login:

`<security:form-login/>`:

- Makes InMemory Filter to generate form page for asking username, password from end-user.

`<security:basic-login/>`:

- Makes the InMemory filter to given instruction to browser for generating dialog box for collecting username, password from end-users.

To specify error page for Authorization failure:

- Write the follow in code inside `<security-http>` tag
`<!-- To specify custom error page for 403 error (authorization failure) -->`
`<security:access-denied-handler error-page="/access_denied.jsp"/>`

- Create the `access_denied.jsp` page in WebContent folder

`access_denied.jsp`

```
<h1 style="color: red; text-align: center;">Authorization failed</h1>
<a href="/welcome">Home</a>
```

To enable logout filter:

- Write the follow in code inside `<security-http>` tag
`<!-- For Logout -->`
`<security:logout logout-success-url="/logout.jsp"/>`

- Create the `logout.jsp` page in WebContent folder


`logout.jsp`

```
<h1 style="color: green; text-align: center;">Logged out
Successfully</h1>
<a href="/welcome">Home</a>
```

- Provide logout hyperlink in `show_result.jsp` file

```
<a href="/logout">Logout</a>
```

`<security:remember-me>`:


-  Adds remember me filter to existing filters to remember the authenticated users across the multiple sessions by internally taking the support of persistent cookies. i.e. after authentication, if you close the browser without sign-out. and browser is closed. And if try to login using same browser, it will not ask for authentication.

- You have to write this tag inside <security-http> tag.

To specify maximum sessions will login at a time:

- Write the follow in code inside <security-http> tag

```
<!-- Maximum login -->
<security:session-management>
    <security:concurrency-control error-if-maximum-exceeded="true"
max-sessions="2"/>
</security:session-management>
```

 **max-sessions="2"**: Specifies how many logins/ sessions can be there per user/ principal maximum at a time.

Taking properties file as Authentication Info Provider:

Step 1: Create properties file by having entries in the following format
key (username) = value (password , role, enabled/ disabled)

info.properties (com/nt/commons package)

```
#username=password, role, enable/ disable
raja={noop}rani, ROLE_MANAGER, disabled
rajesh={noop}hyd, ROLE_ADMIN, enabled
karan={noop}kiran, ROLE_CUSTOMER, enabled
lokesh={noop}delhi, ROLE_VISITOR, enabled
```

Step 2: Configure the properties file in service-beans.xml Spring bean configuration file

service-beans.xml

```
<security:authentication-manager>
    <!-- Taking current Spring bean configuration
file itself as authentication info provider -->

    <security:authentication-provider>
        <security:user-service
properties="classpath:com/nt/commons/info.properties"/>
    </security:authentication-provider>

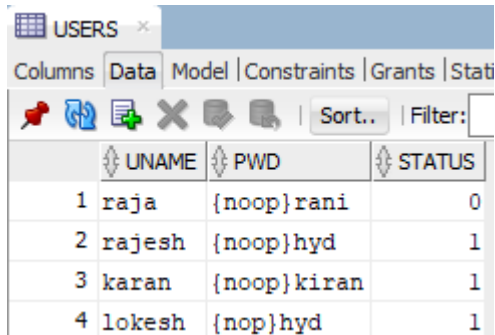
</security:authentication-manager>
```

Configuring DB s/w and its DB tables as Authentication Info provider:

Step 1: Create two DB tables

1. Having user details
2. Having roles details with one to many relationships.

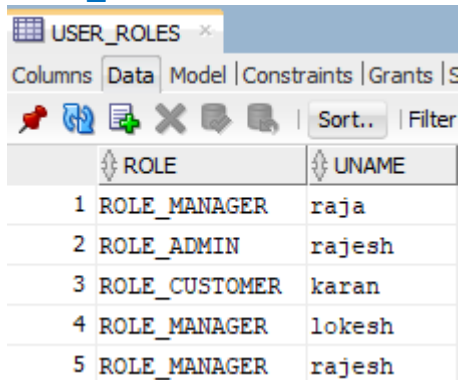
USERS



	UNAME	PWD	STATUS
1	raja	{noop}rani	0
2	rajesh	{noop}hyd	1
3	karan	{noop}kiran	1
4	lokesh	{nop}hyd	1

```
CREATE TABLE "SYSTEM". "USERS"  
("UNAME" VARCHAR2(20 BYTE) NOT NULL ENABLE,  
"PWD" VARCHAR2(20 BYTE),  
"STATUS" NUMBER,  
CONSTRAINT "USERS PK" PRIMARY KEY ("UNAME"));
```

USER_ROLES



	ROLE	UNAME
1	ROLE_MANAGER	raja
2	ROLE_ADMIN	rajesh
3	ROLE_CUSTOMER	karan
4	ROLE_MANAGER	lokesh
5	ROLE_MANAGER	rajesh

```
CREATE TABLE ROLES"  
("ROLE" VARCHAR2(20 BYTE),  
"UNAME" VARCHAR2(20 BYTE),  
CONSTRAINT "FK" FOREIGN KEY ("UNAME")  
REFERENCES "SYSTEM"."USERS" ("UNAME") ENABLE);
```

- After creating the both tables enable the foreign key constraint on USER_ROLES table by right click on the table -> Constraint -> Add Foreign key -> then fill the following details, as like below

Add Foreign Key

Prompts | SQL

Owner: SYSTEM

Name: USER_ROLES

Constraint Name: FK

Column Name: UNAME

References Table Name: USERS

Referencing Column: UNAME

Buttons: Help, Apply, Cancel

Note: Here DB table names, column names are programmer choice.

Step 2: In persistence-beans.xml configure data Source (either standalone DS like HikariCP or Server Managed)

[persistence-beans.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:jee="http://www.springframework.org/schema/jee"
    xsi:schemaLocation="http://www.springframework.org/schema/jee
http://www.springframework.org/schema/jee/spring-jee-4.3.xsd
http://www.springframework.org/schema/beans
https://www.springframework.org/schema/beans/spring-beans-4.3.xsd
http://www.springframework.org/schema/context
https://www.springframework.org/schema/context/spring-context-4.3.xsd">

    <jee:jndi-lookup id="ds" jndi-name="java:/comp/env/DsJndi"/>

</beans>
```

Note: Import this persistence-beans.xml file to applicationContext.xml file

Step 3: Configure DB s/w as persistence provider in security-beans.xml file

security-beans.xml

```
<security:authentication-manager>
  <!-- Taking current Spring bean configuration
  file itself as authentication info provider -->
  <security:authentication-provider>
    <!-- DB configure for authentication provider -->
    <security:jdbc-user-service data-source-ref="ds"
                                users-by-username-
query="SELECT UNAME, PWD, STATUS FROM USERS WHERE UNAME=?"
                                authorities-by-username-
query="SELECT UNAME, ROLE FROM USER_ROLES WHERE UNAME=?" />

  </security:authentication-provider>
</security:authentication-manager>
```

Step 4: Add spring-jdbc-<version>.jar, ojdbc8.jar dependency to pom.xml/
build.gradle

build.gradle

```
//
https://mvnrepository.com/artifact/org.springframework/spring-jdbc
implementation group: 'org.springframework', name: 'spring-
jdbc', version: '5.2.8.RELEASE'
//
https://mvnrepository.com/artifact/com.oracle.database.jdbc/ojdbc6
implementation group: 'com.oracle.database.jdbc', name:
'ojdbc6', version: '11.2.0.4'
```

Spring security Web application using Spring Boot

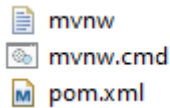
- ✚ When we add Spring web, Spring security starters we will get the following initializer automatically,
 - ServletInitializer -> takes care of Dispatcher Servlet component configuration.
 - SpringsecurityInitializer -> takes care of DelegatingFilterProxy configuration
 - This class internally extends from AbstractSecurityWebApplicationInitializer
- ✚ @Configuration classes are alternate for Spring bean files but we

should develop @Configuration class for security configurations as alternate to security-beans.xml by extending from “org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter”.

- ✚ This Adapter class provides a convenient base class for creating a WebSecurityConfigurer instance. The implementation allows customization by overriding methods.

Directory Structure of SecurityBootProj01-LocalAppBoot:

- ✚ SecurityBootProj01-LocalAppBoot [boot] [devtools]
 - Referenced Types
 - > Deployment Descriptor: SecurityBootProj01-LocalAppBoot
 - ✚ Spring Elements
 - > Beans
 - > JAX-WS Web Services
 - ✚ Java Resources
 - ✚ src/main/java
 - ✚ com.nt
 - > SecurityBootProj01LocalAppBootApplication.java
 - > ServletInitializer.java
 - ✚ com.nt.config
 - > SecurityConfig.java
 - ✚ com.nt.controller
 - > LocaleOperationsController.java
 - ✚ com.nt.service
 - > ILocaleInfoService.java
 - > LocaleInfoServiceImpl.java
 - ✚ src/main/resources
 - static
 - templates
 - application.properties
 - > src/test/java
 - > Libraries
 - > JavaScript Resources
 - ✚ Deployed Resources
 - ✚ webapp
 - ✚ WEB-INF
 - lib
 - ✚ pages
 - home.jsp
 - show_result.jsp
 - access_denied.jsp
 - index.jsp
 - logout.jsp
 - > web-resources
 - > src
 - ✚ target
 - > m2e-wtp
 - HELP.md



- Develop the above directory structure using Spring Starter Project option.
- We have to choose the following Spring Starter Project Dependencies
 - Spring Boot Dev Tools
 - Lombok
 - JDBC API
 - Oracle Driver
 - Spring Security
 - Spring Web
- Add the following code in their respective file, rest of code copy from previous project.

application.properties

```
#View Resolver configuration
spring.mvc.view.prefix=/WEB-INF/pages/
spring.mvc.view.suffix=.jsp

#To work with server managed connection pool
spring.datasource.jndi-name=java:/comp/env/DsJndi
```

SecurityConfig.java

```
package com.nt.config;

import javax.sql.DataSource;

import org.springframework.beans.factory.annotation.Autowired;
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.WebSecurityConfigurerAdapter;

@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter {
```

```

    @Autowired
    private DataSource ds;

    @Override
    public void configure(AuthenticationManagerBuilder auth) throws
Exception {
        auth.jdbcAuthentication().
            dataSource(ds).
            usersByUsernameQuery("SELECT UNAME, PWD,
STATUS FROM USERS WHERE UNAME=?").
            authoritiesByUsernameQuery("SELECT
UNAME, ROLE FROM USER_ROLES WHERE UNAME=?");
    }

    @Override
    public void configure(HttpSecurity http) throws Exception {

        http.authorizeRequests().antMatchers("/welcome").access("permitAll
");

        antMatchers("/countries").access("hasAnyRol('ROLE_ADMIN','ROLE_
MANAGER')");

        and().formLogin().
        and().logout().logoutSuccessUrl("/logout.jsp").

        and().exceptionHandling().accessDeniedPage("/access_denied.jsp").
        and().rememberMe().

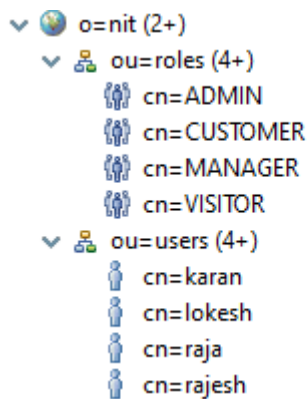
        and().sessionManagement().maximumSessions(2).maxSessionsPreven
tsLogin(true);
    }
}

```

LDAP Server

- ✚ Apache LDAP Server is best Authentication info provider because it does not allow to recover the password, it allows only to reset password.
- ✚ To add users and roles and etc. to Apache LDAP server refer PDF document that is supplied [\[Spring Security LDAP Converted\]](#).
- ✚ To download Apache directory studio: [\[download\]](#)

- ✚ The following tree structure should appear after creating/ adding all the entries of user's details and roles having users, for that follow the PDF given above.



Procedure to configure LDAP server as Authentication provider for spring security Application:

Step 1: Keep any spring security app ready.

Step 2: Add spring-security-ldap dependency to the project.

Step 3: Collect LDAP Server host name, port number, server root username, password details

Host Name: localhost

Port Number: 10389

Root Username: admin

Root Password: secret

Root organization unit name: ou=system

Step 4: Write following entries in security-beans.xml enable LDAP server as Authentication info provider.

```
<security:authentication-manager>
  <!-- For authentication and authorization -->
  <security:ldap-authentication-provider
    user-search-filter="(uid={0})" user-search-base="ou=users"
    group-search-filter="(uniqueMember={0})" group-search-
base="ou=roles" group-role-attribute="cn" role-prefix="ROLE_">
  </security:ldap-authentication-provider>
</security:authentication-manager>

<!-- For Connecting to LDAP server -->
<security:ldap-server id="ldapServer" url="ldap://localhost:10389/o=nit"
manager-dn="uid=admin ou=system" manager-password="secret"/>
<security:ldap-user-service server-ref="ldapServer" user-search-
filter="(uid={0})"/>
```

Step 5: Run the application and make sure while running the application your LDAP server must be in running mode.

Directory Structure of SecurityAnnoProj02-LocaleApp-LDAPServer:

- + Copy paste the SecurityAnnoProj01-LocaleApp application and change rootProject.name to SecurityAnnoProj02-LocaleApp-LDAPServer in settings.gradle file
- + Change the Web Project Settings to SecurityAnnoProj02-LocaleApp-LDAPServer.
- + Add the following code in their respective files following the above steps.

security-beans.xml

```
<security:authentication-manager>
    <!-- For authentication and authorization -->
    <security:ldap-authentication-provider
        user-search-filter="(uid={0})" user-search-
base="ou=users"
        group-search-filter="(uniqueMember={0})" group-search-
base="ou=roles"
        group-role-attribute="cn" role-prefix="ROLE_">
    </security:ldap-authentication-provider>
</security:authentication-manager>

<!-- For Connecting to LDAP server -->
<security:ldap-server id="ldapServer"
url="ldap://localhost:10389/o=nit" manager-dn="uid=admin, ou=system"
manager-password="secret"/>

<security:ldap-user-service server-ref="ldapServer" user-search-
filter="(uid={0})"/>
```

----- The END -----