* Basic Security
  1. Create a Spring MVC Project using sts

**Update project dependencies in pom.xml**

* Lets start with very first step i.e. update the project dependencies. It will add following four sub-modules in demo for following reasons:
* **spring-security-core**  
  It contains core authentication and access-contol classes and interfaces.
* **spring-security-web**  
  It contains filters and related web-security infrastructure code. It also enable URL based security which we are going to use in this demo.
* **spring-security-config**  
  It contains the security namespace parsing code. You need it if you are using the Spring Security XML file for configuration.
* **spring-security-taglibs**  
  It provides basic support for accessing security information and applying security constraints in JSPs.
* So the pom.xml file will be updated with: 

**Configure DelegatingFilterProxy in web.xml**

Spring Security’s web infrastructure is based entirely on standard servlet filters. These filters are defined in web.xml file or they will be ignored by the servlet container. In Spring Security, the filter classes are also Spring beans defined in the application context and thus able to take advantage of Spring’s rich dependency-injection facilities and lifecycle interfaces. Spring’s [**DelegatingFilterProxy**](http://static.springsource.org/spring/docs/2.0.8/api/org/springframework/web/filter/DelegatingFilterProxy.html) provides the link between web.xml and the application context.

<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>

|  |  |
| --- | --- |
|  |  |

If you are not using any explicit filter definitions and wants spring to configure basic infrastructure for you, then use filter name as ‘**springSecurityFilterChain**‘ as in above example. Note that you should not use this bean name yourself. Once you’ve added this to your web.xml, you’re ready to start editing your spring security configuration file. Web security services are configured using the element.

Also do not forget to put security configuration file in context config location setting.

|  |  |
| --- | --- |
|  |  |

<context-param>

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

<param-value>/WEB-INF/spring/root-context.xml, /WEB-INF/spring/appServlet/spring-security.xml</param-value>

</context-param>A complete web.xml file will look like this:



**Add security configuration in spring-security.xml**

As we learned in last section that using filter name as “*springSecurityFilterChain*” can help you configure the basic infrastructure using element. Lets see how it is configured first? I have written a basic configuration for this demo:

<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-3.0.xsd*

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

*http://www.springframework.org/schema/security/spring-security-3.2.xsd"*>

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

<intercept-url pattern=*"/admin\*\*"* access=*"ROLE\_USER"* />

<http-basic /> <!-- This tag is essential to trigger Windows dialog box for accepting credentials.

If not given, Spring uses its own login page. -->

</http>

<authentication-manager>

<authentication-provider>

<user-service>

<user name=*"smita"* password=*"password"* authorities=*"ROLE\_USER"* />

</user-service>

</authentication-provider>

</authentication-manager>

</beans:beans>

http: Include configuration related url level security. This element is the parent for all web-related namespace functionality.

**intercept-url**: This will match the requested url pattern from request and will decide what action to take based on access value.

**Update the controller**

I will reuse the previous controller and will add additional mappings and handler methods in controller. These additional URLs are /admin and /innerpage. The updated controller having all method handlers looks like this:



**Add related views**

