# Spring Security 4 Hello World Annotation+XML Example

We will demonstrate Spring Security 4 usage to secure a Spring MVC web application, securing URL access with authentication. We will use classic Hello World example to learn Spring Security 4 basics. This post uses Spring Annotation based configuration for Servlet 3.0 containers [thus no web.xml] and also shows corresponding XML based Security configuration for side-by-side comparison where applicable.

**Following technologies being used:**

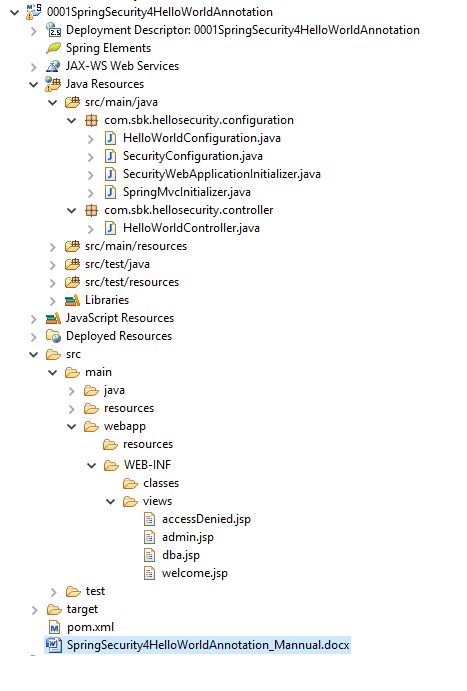
* Spring 4.1.6.RELEASE
* Spring Security 4.0.1.RELEASE
* Maven 3
* JDK 1.7
* Tomcat 7
* STS

Let’s begin.

1. **Create a Spring MVC project using STS**
2. **Delete the web.xml as we will be doing java Configuration**
3. **Delete the spring folder from WEB-INF**
4. **Update pom.xml** ****

* *First thing to notice here is the maven-war-plugin declaration. As we are using full annotation configuration,****we don’t even use web.xml****, so we will need to configure this plugin in order to avoid maven failure to build war package. We are using latest versions(at time of writing) of Spring and Spring Security.*
* *Along with that, we have also included JSP/Servlet/Jstl dependencies which we will be needing as we are going to use servlet api’s and jstl view in our code. In general, containers might already contains these libraries, so we can set the scope as ‘provided’ for them in pom.xml.*

1. **Following is the project structure**



1. **Let’s now add the content mentioned in above structure explaining each in detail**.
2. **Create com.sbk.hellosecurity.configuration class**

The first and foremost step to add spring security in our application is to create **Spring Security Java Configuration**. This configuration creates a Servlet Filter known as the springSecurityFilterChain which is responsible for all the security (protecting the application URLs, validating submitted username and passwords, redirecting to the log in form, etc) within our application.



Method configureGlobalSecurity in above class configures AuthenticationManagerBuilder with user credentials and allowed roles. This AuthenticationManagerBuilder creates AuthenticationManager which is responsible for processing any authentication request. Notice that in above example, we have used in-memory authentication while you are free to choose from JDBC, LDAP and other authentications.

The overridden Method Configure configures HttpSecurity which allows configuring web based security for specific http requests. By default it will be applied to all requests, but can be restricted using requestMatcher(RequestMatcher)/antMathchers or other similar methods.

In above configuration, we say that URL’s ‘/’ & ‘/home’ are not secured, anyone can access them. URL ‘/admin/\*\*’ can only be accessed by someone who have ADMIN role. URL ‘/db/\*\*’ can only be accessed by someone who have both ADMIN and DBA roles.

Method formLogin provides support for form based authentication and will generate a default form asking for user credentials. You are allowed to configure your own login form.We will see examples for the same in subsequent posts.

We have also used exceptionHandling().accessDeniedPage() which in this case will catch all 403 [http access denied] exceptions and display our user defined page instead of showing default HTTP 403 page [ which is not so helpful anyway].

**Above security configuration in XML configuration format would be:**

|  |
| --- |
| <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-4.1.xsd>  <http://www.springframework.org/schema/security> <http://www.springframework.org/schema/security/spring-security-4.0.xsd>">        <http auto-config="true" >          <intercept-url pattern="/" access="permitAll" />          <intercept-url pattern="/home" access="permitAll" />          <intercept-url pattern="/admin\*\*" access="hasRole('ADMIN')" />          <intercept-url pattern="/dba\*\*" access="hasRole('ADMIN') and hasRole('DBA')" />          <form-login  authentication-failure-url="/Access\_Denied" />      </http>        <authentication-manager >          <authentication-provider>              <user-service>                  <user name="bill"  password="abc123"  authorities="ROLE\_USER" />                  <user name="admin" password="root123" authorities="ROLE\_ADMIN" />                  <user name="dba"   password="root123" authorities="ROLE\_ADMIN,ROLE\_DBA" />              </user-service>          </authentication-provider>      </authentication-manager>    </beans:beans> |

#### Register the springSecurityFilter with war

Below specified initializer class registers the springSecurityFilter [created in previous step] with application war. **com.sbk.hellosecurity.configuration**



**Above setup in XML configuration format would be(In web.xml):**

|  |
| --- |
| <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> |

#### Add Controller com.sbk.hellosecurity.controller

#### 

Methods in controller class are trivial. Method getPrincipal is a generic function which returns the logged in user name from Spring SecurityContext. Method logoutPage handles the logging out with a simple call to**SecurityContextLogoutHandler().logout(request, response, auth);**. It’s handy and saves you from putting cryptic logout logic in your JSP’s which is not really manageable. You might have noticed that ‘/login’ is missing, it is because it will be generated and handled by default by Spring Security.

#### Add SpringMVC Configuration Class

**com.sbk.hellosecurity.configuration**



#### Add Initializer class

**com.sbk.hellosecurity.configuration**



Notice that above initializer class extends AbstractAnnotationConfigDispatcherServletInitializerwhich is the base class for all WebApplicationInitializer implementations. Implementations of WebApplicationInitializer configures ServletContext programatically, for Servlet 3.0 environments. It means we won’t be using web.xml and we will deploy the app on Servlet 3.0 container.

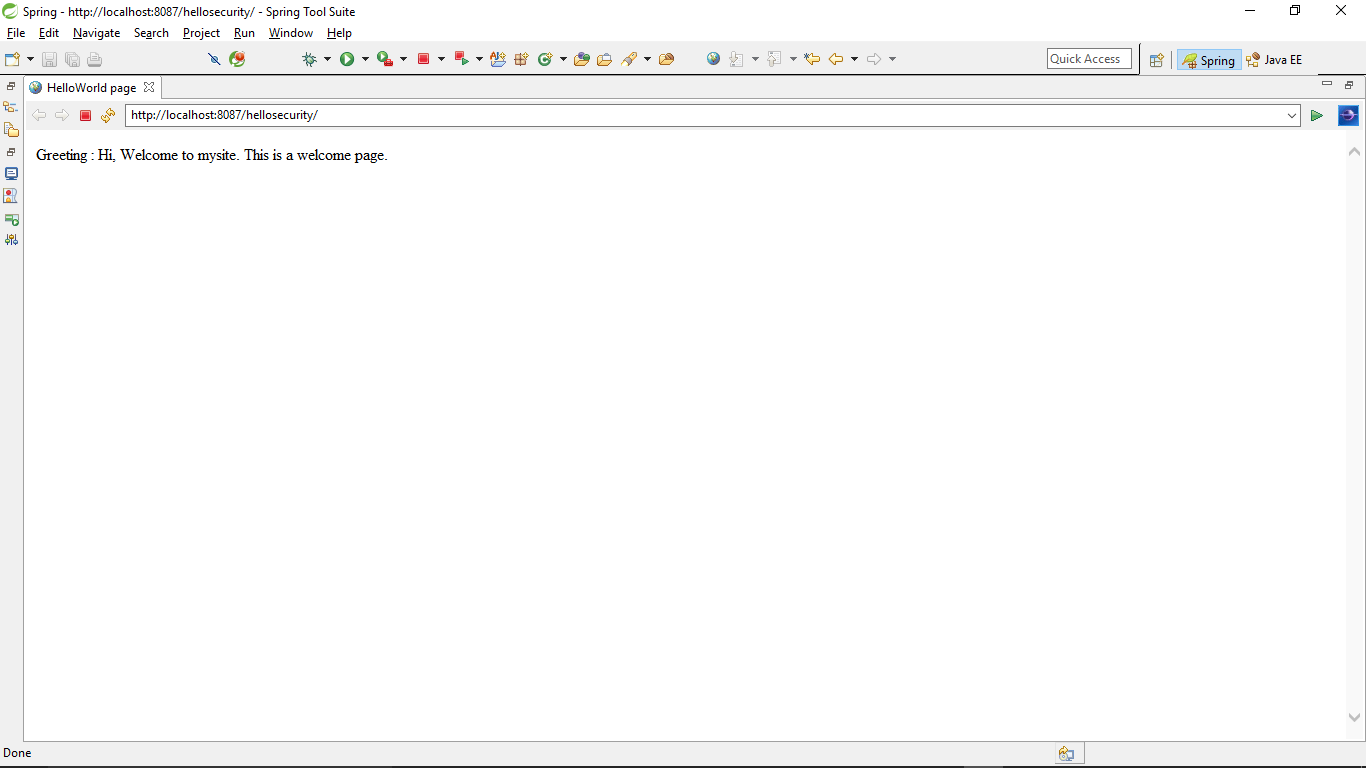
#### Add Views



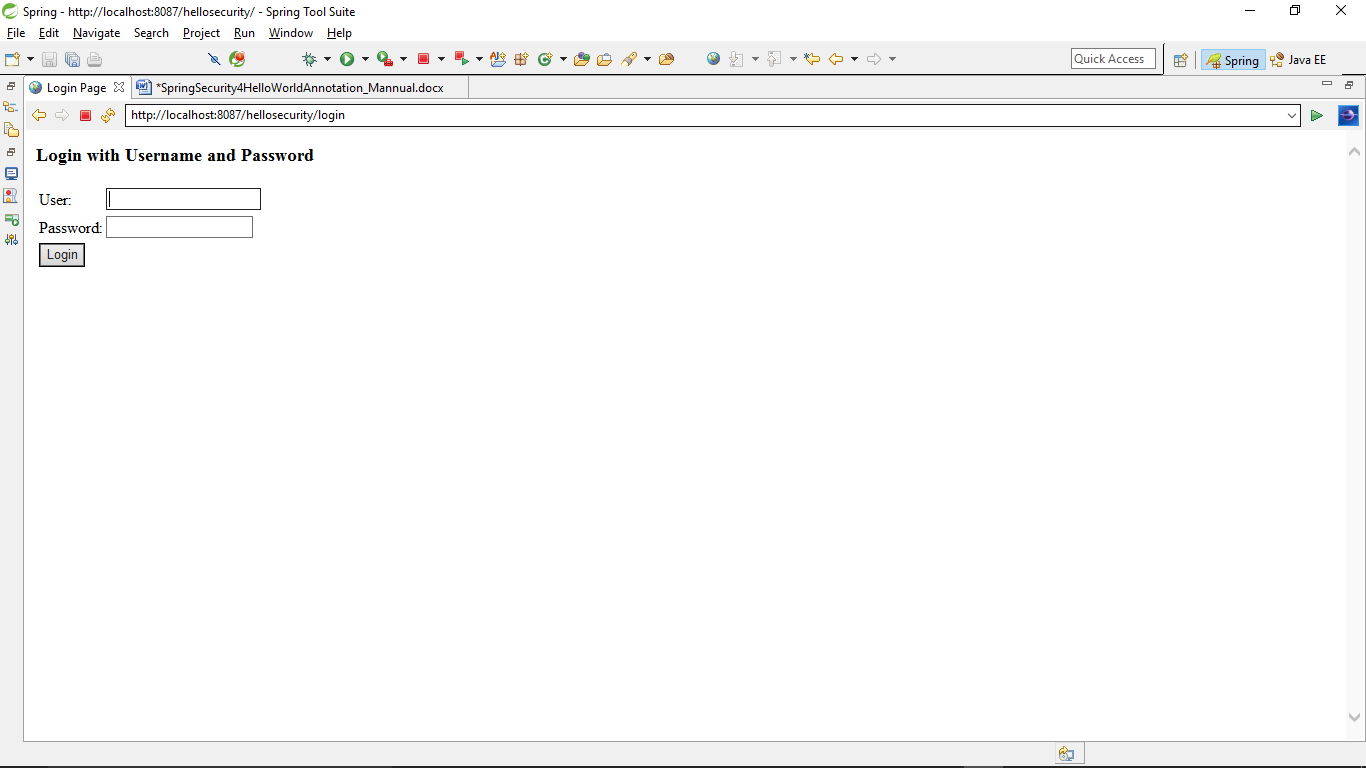
#### Build and Deploy the application

***NOTE : Before deploying, check the java build path(1.7), Check java compiler(1.7), java facets(jdk1.7,web module version 3.0), targeted runtime(tom cat7) and Add Maven dependency to build path***

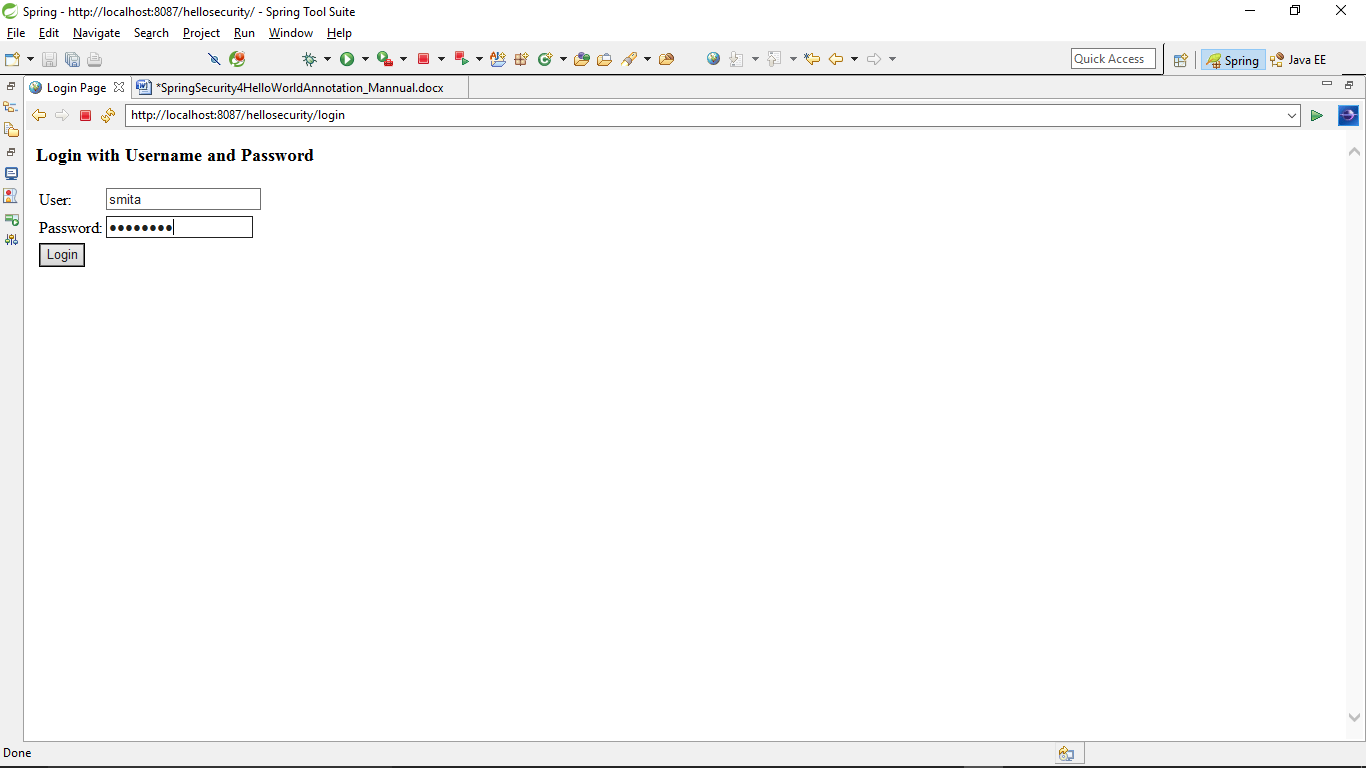
#### Run the application



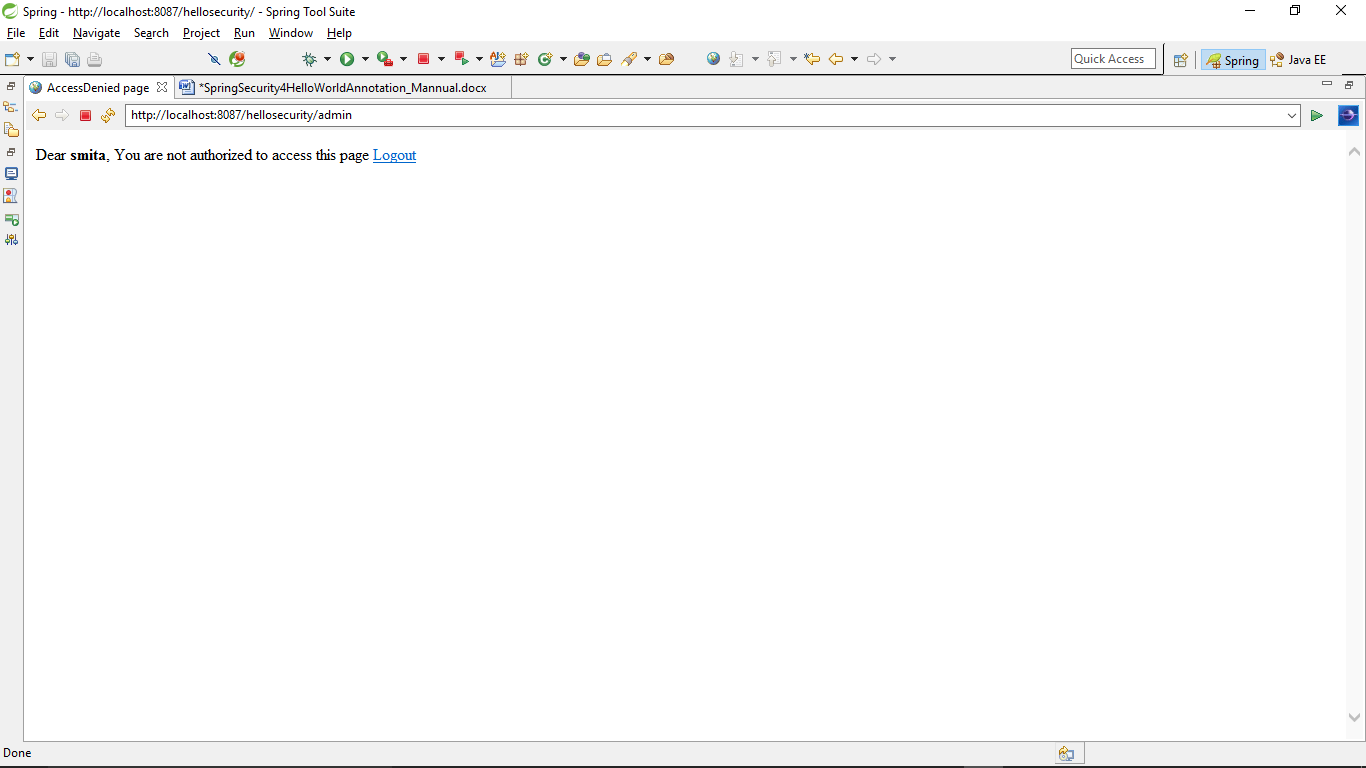
Now try to access admin page on http://localhost:8087/hellosecurity/login, you will be prompted for login.

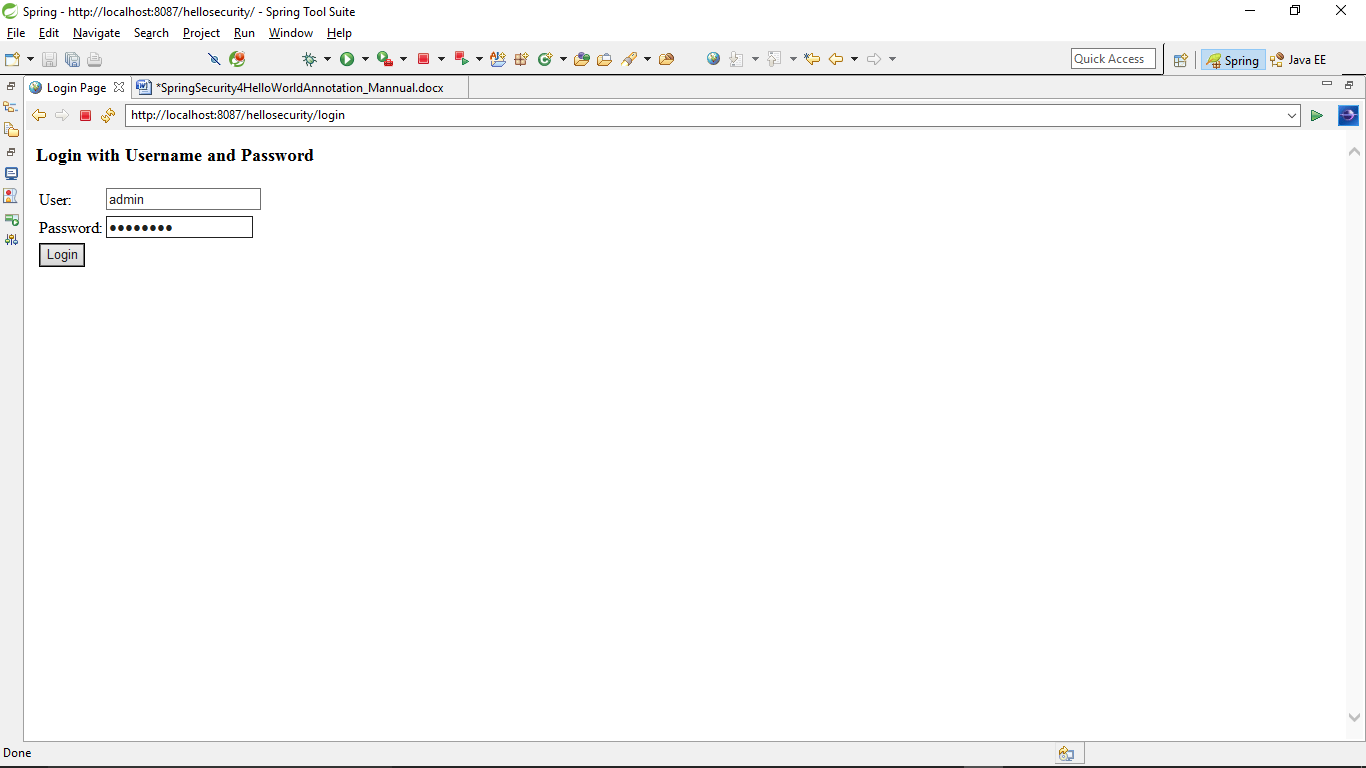


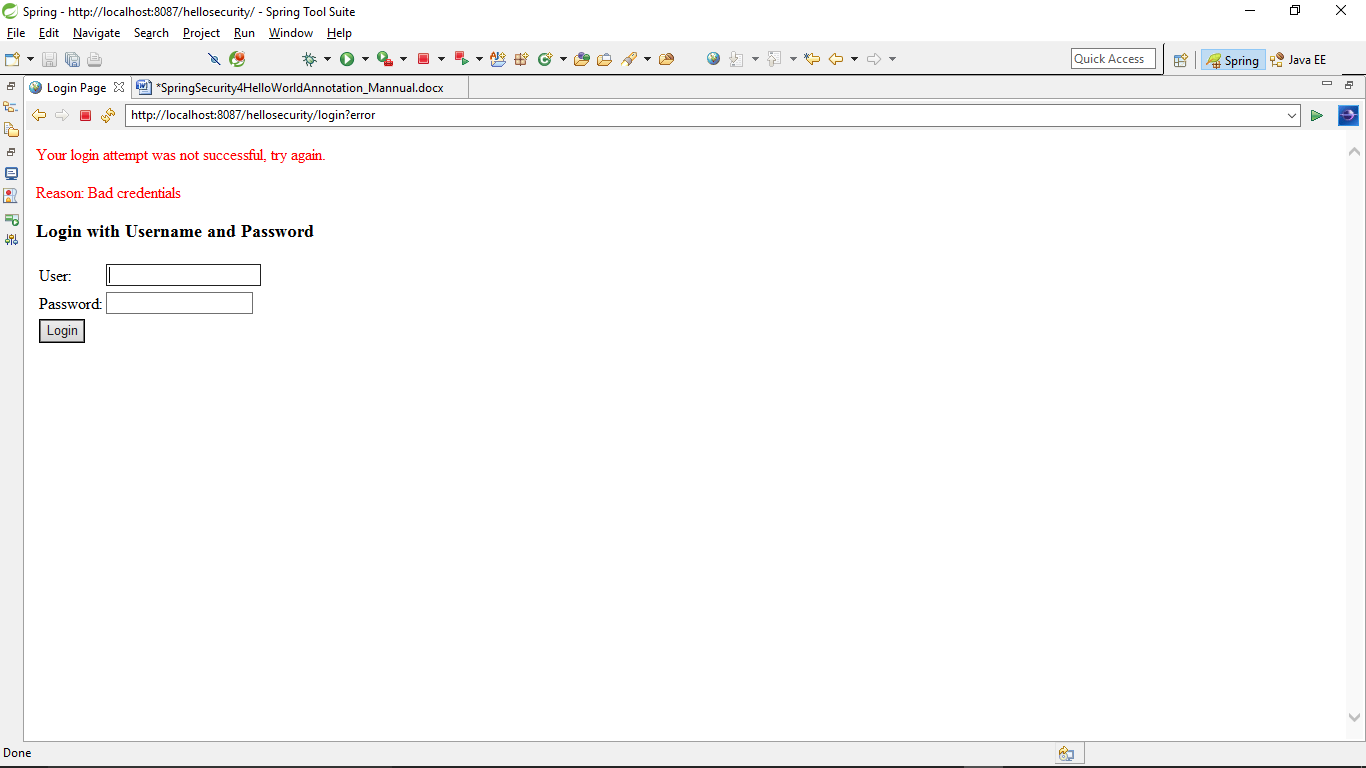
Provide credentials of a ‘USER’ role.*enter [smita,password]*



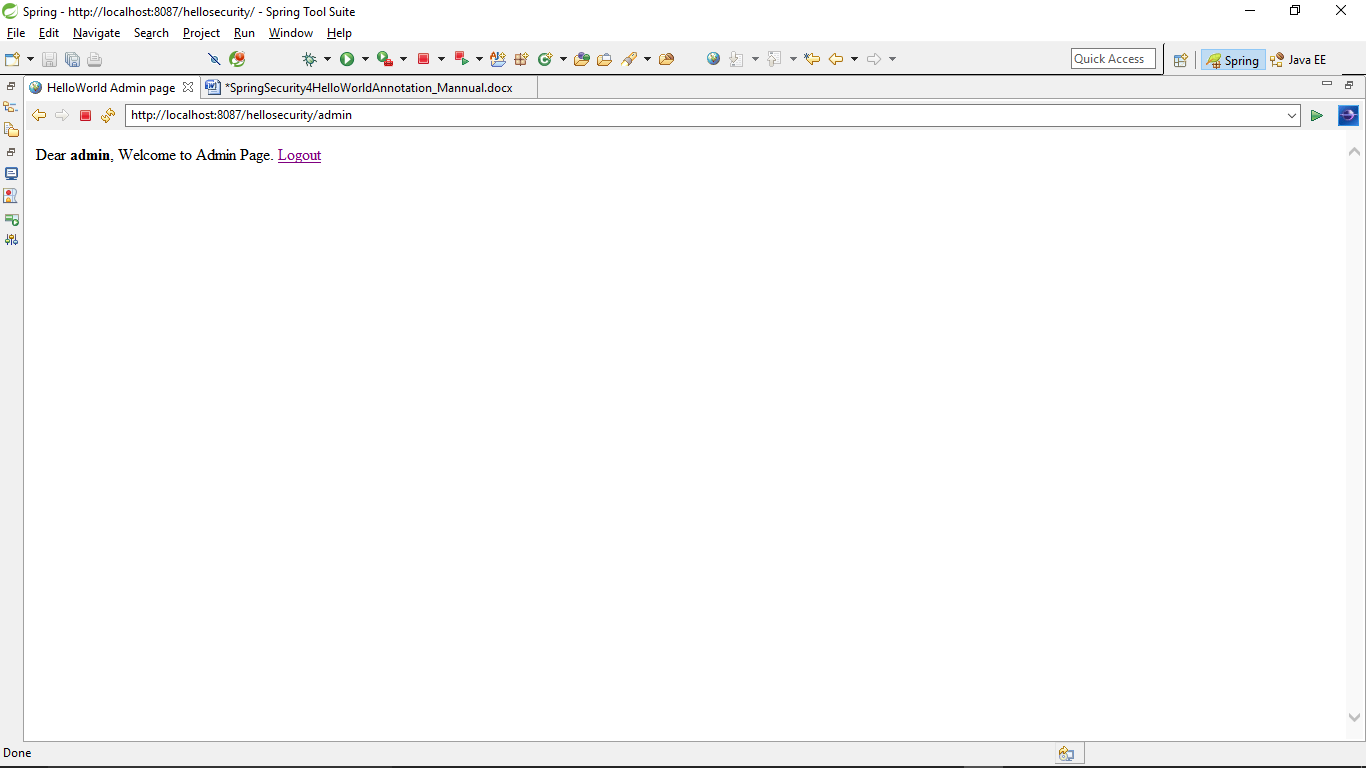
Submit, you will see AccessDenied Page



1. Logout. http://localhost:8087/hellosecurity/login Login with ADMIN role credentials.[admin,wrongpassword]
2. 
3. Provide wrong password



Provide proper admin role credentials and login [admin,password]



Now try to access db page on **http://localhost:8087/hellosecurity/db**, you will get AccessDenied page.

