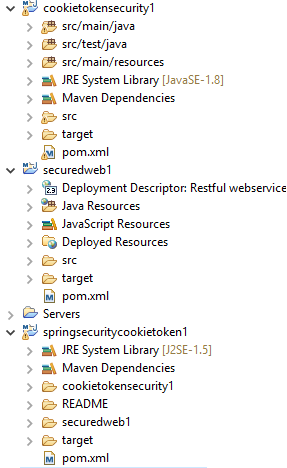
Both Cookie and Token Based Authentication in Spring Security – 2017

# Introduction

Here both cookie and token are used for authentication. The technologies used are both Spring and Jersey. Here security is used as separate service which is abstracted away from core restful web service. For simplicity it is used as maven multi module project.

# Project Structure



# Parent Maven Configuration (pom.xml)

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"* xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<groupId>springsecuritycookietoken1</groupId>

<artifactId>springsecuritycookietoken1</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>pom</packaging>

<name>springsecuritycookietoken1</name>

<url>http://maven.apache.org</url>

<properties>

<org.springframework.version>4.0.7.RELEASE</org.springframework.version>

<log4j.version>1.2.17</log4j.version>

<jdk.version>1.8</jdk.version>

<spring.security.version>3.2.5.RELEASE</spring.security.version>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

<dependencies>

<!-- Jersey related jar files -->

<dependency>

<groupId>javax.ws.rs</groupId>

<artifactId>javax.ws.rs-api</artifactId>

<version>2.0</version>

</dependency>

<dependency>

<groupId>org.glassfish.jersey.containers</groupId>

<artifactId>jersey-container-servlet</artifactId>

<version>2.6</version>

</dependency>

<dependency>

<groupId>org.glassfish.jersey.core</groupId>

<artifactId>jersey-client</artifactId>

<version>2.6</version>

</dependency>

<dependency>

<groupId>org.codehaus.jackson</groupId>

<artifactId>jackson-core-asl</artifactId>

<version>1.9.13</version>

</dependency>

<dependency>

<groupId>org.codehaus.jackson</groupId>

<artifactId>jackson-mapper-asl</artifactId>

<version>1.9.13</version>

</dependency>

<dependency>

<groupId>org.glassfish.jersey.media</groupId>

<artifactId>jersey-media-multipart</artifactId>

<version>2.5.1</version>

</dependency>

<dependency>

<groupId>org.glassfish.jersey.media</groupId>

<artifactId>jersey-media-json-jackson</artifactId>

<version>2.5.1</version>

</dependency>

<!-- Java Servlet -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>3.1.0</version>

</dependency>

<!-- Aspect Oriented Programming (AOP) Framework (depends on spring-core,

spring-beans) Define this if you use Spring AOP APIs (org.springframework.aop.\*) -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>${org.springframework.version}</version>

</dependency>

<dependency>

<groupId>javax.annotation</groupId>

<artifactId>jsr250-api</artifactId>

<version>1.0</version>

</dependency>

<dependency>

<groupId>javax.inject</groupId>

<artifactId>javax.inject</artifactId>

<version>1</version>

</dependency>

<dependency>

<groupId>javax.enterprise</groupId>

<artifactId>cdi-api</artifactId>

<version>1.1-20130403</version>

</dependency>

<!-- Jersey Spring Integration -->

<dependency>

<groupId>org.glassfish.jersey.ext</groupId>

<artifactId>jersey-spring3</artifactId>

<version>2.10.1</version>

</dependency>

<!-- Spring Security -->

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-web</artifactId>

<version>${spring.security.version}</version>

</dependency>

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-config</artifactId>

<version>${spring.security.version}</version>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>3.0.1</version>

<scope>provided</scope>

</dependency>

<!-- https://mvnrepository.com/artifact/commons-codec/commons-codec -->

<dependency>

<groupId>commons-codec</groupId>

<artifactId>commons-codec</artifactId>

<version>1.10</version>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jstl</artifactId>

<version>1.2</version>

</dependency>

<!-- log4j -->

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>${log4j.version}</version>

</dependency>

<!-- Commons-logging -->

<dependency>

<groupId>commons-logging</groupId>

<artifactId>commons-logging</artifactId>

<version>1.2</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.3</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</build>

**<modules>**

**<module>cookietokensecurity1</module>**

**<module>securedweb1</module>**

**</modules>**

</project>

# Security as a service or Module

# Project cookietokensecurity1

## **Maven Configuration (pom.xml)**

<?xml version=*"1.0"*?>

<project xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"* xmlns=*"http://maven.apache.org/POM/4.0.0"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*>

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>springsecuritycookietoken1</groupId>

<artifactId>springsecuritycookietoken1</artifactId>

<version>0.0.1-SNAPSHOT</version>

</parent>

<artifactId>cookietokensecurity1</artifactId>

<name>cookietokensecurity1</name>

<url>http://maven.apache.org</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.8.1</version>

<scope>test</scope>

</dependency>

</dependencies>

</project>

## **Spring Security Configuration(security-Context.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:p=*"http://www.springframework.org/schema/p"*

xmlns:context=*"http://www.springframework.org/schema/context"*

xmlns:jee=*"http://www.springframework.org/schema/jee"* xmlns:util=*"http://www.springframework.org/schema/util"*

xmlns:security=*"http://www.springframework.org/schema/security"*

xsi:schemaLocation=*"*

*http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.2.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.2.xsd*

*http://www.springframework.org/schema/jee http://www.springframework.org/schema/jee/spring-jee-3.2.xsd*

*http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-3.2.xsd*

*http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-3.2.xsd*

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

<context:annotation-config/>

<context:component-scan base-package=*"com.ddlab.rnd.spring.token"* />

**<bean id=*"unauthorizedEntryPoint"***

***class="com.ddlab.rnd.spring.token.auth.UnauthorizedEntryPoint" />***

**<bean id=*"tokenAuthFilter"***

**class=*"com.ddlab.rnd.spring.token.auth.TokenAuthenticationFilter"* />**

**<bean id=*"successHandler"***

**class=*"com.ddlab.rnd.spring.token.auth.MySuccessHandler"* />**

**<bean id=*"failureHandler"***

**class=*"org.springframework.security.web.authentication.SimpleUrlAuthenticationFailureHandler"* />**

**<bean id=*"logoutHandler"***

**class=*"com.ddlab.rnd.spring.token.auth.CustomLogoutSuccessHandler"* />**

<!-- create-session="stateless" -->

<!-- If you mention create-session="stateless", then cookie based auth will not work -->

<!-- invalidate-session="true" , bydefault it is always true, no need to mention in logout-->

**<security:http use-expressions=*"true"* entry-point-ref=*"unauthorizedEntryPoint"***

**authentication-manager-ref=*"authenticationManager"*>**

**<security:intercept-url pattern=*"/api/1/authservices/authenticate"* access=*"permitAll"* />**

**<security:intercept-url pattern=*"/api/1/authservices/login"* access=*"permitAll"* />**

**<security:logout logout-url=*"/api/1/authservices/logout"* success-handler-ref=*"logoutHandler"* />**

**<security:intercept-url pattern=*"/api/\*\*"* access=*"isAuthenticated()"* />**

**<security:custom-filter ref=*"tokenAuthFilter"* after=*"FORM\_LOGIN\_FILTER"* />**

**<security:form-login authentication-success-handler-ref=*"successHandler"***

**login-processing-url=*"/api/1/authservices/login"***

**authentication-failure-handler-ref=*"failureHandler"* />**

**</security:http>**

**<security:authentication-manager id=*"authenticationManager"*>**

**<security:authentication-provider>**

**<security:user-service>**

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

**<security:user name=*"piku"* password=*"piku"* authorities=*"ROLE\_USER"* />**

**<security:user name=*"deba"* password=*"deba"* authorities=*"ROLE\_ADMIN"* />**

**<security:user name=*"d"* password=*"d"* authorities=*"ROLE\_ADMIN"* />**

**<security:user name=*"a"* password=*"a"* authorities=*"ROLE\_USER"* />**

**</security:user-service>**

**</security:authentication-provider>**

**</security:authentication-manager>**

<!-- <security:http realm="Protected API" use-expressions="true" auto-config="false"

create-session="stateless" entry-point-ref="unauthorizedEntryPoint"

authentication-manager-ref="authenticationManager"> -->

</beans>

# Java Files

## **Service Layer**

### **UserServices.java**

**package** com.ddlab.rnd.spring.token.services;

**import** org.springframework.security.core.userdetails.UserDetails;

**public** **interface** UserServices {

**public** UserDetails loadUserByUsername(String userName);

}

### **UserServicesImpl.java**

**package** com.ddlab.rnd.spring.token.services;

**import** java.util.ArrayList;

**import** java.util.List;

**import** org.springframework.security.core.GrantedAuthority;

**import** org.springframework.security.core.authority.SimpleGrantedAuthority;

**import** org.springframework.security.core.userdetails.User;

**import** org.springframework.security.core.userdetails.UserDetails;

**import** org.springframework.stereotype.Component;

@Component(value="userServices")

**public** **class** UserServicesImpl **implements** UserServices {

@Override

**public** UserDetails loadUserByUsername(String userName) {

**List<GrantedAuthority> gList = new ArrayList<GrantedAuthority>();**

**GrantedAuthority ga = new SimpleGrantedAuthority("ROLE\_USER");**

**gList.add(ga);**

**UserDetails userDetails = new User(userName,"user",gList);**

**return userDetails;**

}

}

## **Entity layer**

## **TokenTransfer.java**

**package** com.ddlab.rnd.spring.token.entity;

**public** **class** TokenTransfer {

**private** **final** String token;

**public** TokenTransfer(String token) {

**this**.token = token;

}

**public** String getToken() {

**return** **this**.token;

}

}

## **Authentication Layer**

### **UnAuthorizedEntryPoint.java**

**package** com.ddlab.rnd.spring.token.auth;

**import** java.io.IOException;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**import** org.springframework.security.core.AuthenticationException;

**import** org.springframework.security.web.AuthenticationEntryPoint;

**public** **class** UnauthorizedEntryPoint **implements** AuthenticationEntryPoint {

@Override

**public void commence(HttpServletRequest request,**

**HttpServletResponse response, AuthenticationException authException)**

**throws IOException, ServletException {**

**response.sendError(HttpServletResponse.*SC\_UNAUTHORIZED*,**

**"Unauthorized: Authentication token was either missing or invalid.");**

**}**

}

### **TokenAuthenticationFilter.java**

**package** com.ddlab.rnd.spring.token.auth;

**import** java.io.IOException;

**import** javax.servlet.FilterChain;

**import** javax.servlet.ServletException;

**import** javax.servlet.ServletRequest;

**import** javax.servlet.ServletResponse;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

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

**import** org.springframework.beans.factory.annotation.Qualifier;

**import** org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

**import** org.springframework.security.core.Authentication;

**import** org.springframework.security.core.context.SecurityContextHolder;

**import** org.springframework.security.core.userdetails.UserDetails;

**import** org.springframework.security.web.authentication.WebAuthenticationDetailsSource;

**import** org.springframework.security.web.authentication.logout.SecurityContextLogoutHandler;

**import** org.springframework.web.filter.GenericFilterBean;

**import** com.ddlab.rnd.spring.token.services.UserServicesImpl;

**public** **class** TokenAuthenticationFilter **extends** GenericFilterBean {

@Autowired

@Qualifier("userServices")

**private** UserServicesImpl services;

@Override

**public** **void** doFilter(ServletRequest request, ServletResponse response, FilterChain chain)

**throws** IOException, ServletException {

HttpServletRequest httpRequest = (HttpServletRequest) request;

HttpServletResponse httpResponse = (HttpServletResponse) response;

System.***out***.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*AuthenticationTokenProcessingFilter\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

**String authToken = TokenUtils.*extractAuthTokenFromRequest*(httpRequest);**

**String userName = TokenUtils.*getUserNameFromToken*(authToken);**

System.***out***.println("UserName :::" + userName);

Authentication auth = SecurityContextHolder.*getContext*().getAuthentication();

**if** (userName != **null**) {

**UserDetails userDetails = services.loadUserByUsername(userName);**

**boolean validToken = TokenUtils.*isTokenValid*(authToken, userDetails);**

System.***out***.println("Is Token Valied :::" + validToken);

**if** (validToken) {

**UsernamePasswordAuthenticationToken authentication = new UsernamePasswordAuthenticationToken(**

**userDetails, null, userDetails.getAuthorities());**

**authentication.setDetails(new WebAuthenticationDetailsSource().buildDetails(httpRequest));**

**SecurityContextHolder.*getContext*().setAuthentication(authentication);**

}

} **else** {

**SecurityContextLogoutHandler ctxLogOut = new SecurityContextLogoutHandler();**

**ctxLogOut.logout(httpRequest, httpResponse, auth);**

}

**chain.doFilter(request, response);**

}

}

### **MySuccessHandler.java**

**package** com.ddlab.rnd.spring.token.auth;

**import** java.io.IOException;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.Cookie;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

**import** org.apache.commons.codec.binary.Base64;

**import** org.springframework.security.core.Authentication;

**import** org.springframework.security.core.context.SecurityContextHolder;

**import** org.springframework.security.core.userdetails.User;

**import** org.springframework.security.core.userdetails.UserDetails;

**import** org.springframework.security.web.authentication.SimpleUrlAuthenticationSuccessHandler;

**public** **class** MySuccessHandler **extends** SimpleUrlAuthenticationSuccessHandler {

@Override

**public** **void** onAuthenticationSuccess(HttpServletRequest request, HttpServletResponse response, Authentication authentication)

**throws** IOException, ServletException {

System.***out***.println("---------MySuccessHandler-----------");

**UserDetails userDetails = (UserDetails) authentication.getPrincipal();**

System.***out***.println("UserDetails :::"+userDetails);

**User authUser = (User) SecurityContextHolder.*getContext*().getAuthentication().getPrincipal();**

System.***out***.println("authUser :::"+authUser);

**String cookieToken = TokenUtils.*createToken*(userDetails);**

System.***out***.println("Now cookie Token ::::"+cookieToken);

String ecodedCookieToken = **new** String(

Base64.*encodeBase64*(cookieToken.getBytes()));

System.***out***.println("Now ecodedCookieToken ::::"+ecodedCookieToken);

Cookie cook = **new** Cookie("APPASSERTION", ecodedCookieToken);

cook.setPath("/");

**//1 day = 24 \* 60 \* 60**

**//five minutes (5 \* 60)**

**//cook.setMaxAge(1 \* 2 \* 60);**

cook.setMaxAge(60);// 60 seconds

cook.setVersion(1);

response.addCookie(cook);

getRedirectStrategy().sendRedirect(request, response, "/home.jsp");//working

}

}

### **CustomLogoutSuccessHandler.java**

**package** com.ddlab.rnd.spring.token.auth;

**import** java.io.IOException;

**import** javax.servlet.ServletException;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.servlet.http.HttpServletResponse;

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

**import** org.springframework.beans.factory.annotation.Qualifier;

**import** org.springframework.security.core.Authentication;

**import** org.springframework.security.web.authentication.logout.LogoutSuccessHandler;

**import** com.ddlab.rnd.spring.token.services.UserServicesImpl;

**public** **class** CustomLogoutSuccessHandler **implements** LogoutSuccessHandler {

@Autowired

@Qualifier("userServices")

**private** UserServicesImpl services;

@Override

**public** **void** onLogoutSuccess(HttpServletRequest request, HttpServletResponse response, Authentication authentication)

**throws** IOException, ServletException {

HttpServletRequest httpRequest = (HttpServletRequest) request;

String authToken = TokenUtils.*extractAuthTokenFromRequest*(httpRequest);

String userName = TokenUtils.*getUserNameFromToken*(authToken);

**try** {

TokenUtils.*removeUser*(authToken, services.loadUserByUsername(userName));

response.setStatus(HttpServletResponse.***SC\_OK***);

response.getWriter().write("User logged out successfully ...");

} **catch** (Exception ex) {

ex.printStackTrace();

}

}

}

### **AuthenticationResource.java**

**package** com.ddlab.rnd.spring.token.auth;

**import** javax.servlet.ServletConfig;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.ws.rs.FormParam;

**import** javax.ws.rs.POST;

**import** javax.ws.rs.Path;

**import** javax.ws.rs.Produces;

**import** javax.ws.rs.core.Context;

**import** javax.ws.rs.core.MediaType;

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

**import** org.springframework.beans.factory.annotation.Qualifier;

**import** org.springframework.security.authentication.AuthenticationManager;

**import** org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

**import** org.springframework.security.core.Authentication;

**import** org.springframework.security.core.context.SecurityContextHolder;

**import** com.ddlab.rnd.spring.token.entity.TokenTransfer;

**import** com.ddlab.rnd.spring.token.services.UserServices;

@Path("1/authservices")

**public** **class** AuthenticationResource {

@Context

HttpServletRequest request;

@Context

ServletConfig servletConfig;

@Autowired

@Qualifier(value="userServices")

**private** UserServices services;

@Autowired

@Qualifier("authenticationManager")

**private** AuthenticationManager authManager;

@Path("authenticate")

@POST

@Produces(MediaType.***APPLICATION\_JSON***)

**public** TokenTransfer authenticate(@FormParam("username") String username, @FormParam("password") String password)

{

UsernamePasswordAuthenticationToken authenticationToken =

**new** UsernamePasswordAuthenticationToken(username, password);

Authentication authentication = authManager.authenticate(authenticationToken);

SecurityContextHolder.*getContext*().setAuthentication(authentication);

**return** **new** TokenTransfer(TokenUtils.*createToken*(services.loadUserByUsername(username)));

}

}

### **TokenUtils.java**

**package** com.ddlab.rnd.spring.token.auth;

**import** java.security.MessageDigest;

**import** java.security.NoSuchAlgorithmException;

**import** java.util.HashMap;

**import** java.util.Map;

**import** java.util.concurrent.ConcurrentHashMap;

**import** javax.servlet.http.HttpServletRequest;

**import** org.apache.commons.codec.binary.Base64;

**import** org.springframework.security.core.userdetails.UserDetails;

**import** org.springframework.security.crypto.codec.Hex;

**public** **class** TokenUtils {

**public** **static** **final** String ***MAGIC\_KEY*** = "obfuscate";

**private** **static** Map<UserDetails, String> *tokenMap* = **new** ConcurrentHashMap<UserDetails, String>();

**public** **static** String createToken(UserDetails userDetails) {

/\* Expires in one hour \*/

// long expires = System.currentTimeMillis() + 1000L \* 60 \* 60;

/\* Expires in 2 min \*/

**long** expires = System.*currentTimeMillis*() + 1000L \* 60 \* 2;

StringBuilder tokenBuilder = **new** StringBuilder();

tokenBuilder.append(userDetails.getUsername());

tokenBuilder.append(":");

tokenBuilder.append(expires);

tokenBuilder.append(":");

tokenBuilder.append(TokenUtils.*computeSignature*(userDetails, expires));

*tokenMap*.put(userDetails, tokenBuilder.toString());

**return** tokenBuilder.toString();

}

**public** **static** String getUserNameFromToken(String authToken) {

**if** (**null** == authToken) {

**return** **null**;

}

String[] parts = authToken.split(":");

**return** parts[0];

}

**public** **static** **void** removeUser(String authToken, UserDetails userDetails) {

// Validate and then remove it.

**if** (*isTokenValid*(authToken, userDetails))

*tokenMap*.remove(userDetails);

}

**public** **static** **void** removeUser(UserDetails userDetails) {

// Validate and then remove it.

*tokenMap*.remove(userDetails);

}

**public** **static** String computeSignature(UserDetails userDetails, **long** expires) {

StringBuilder signatureBuilder = **new** StringBuilder();

signatureBuilder.append(userDetails.getUsername());

signatureBuilder.append(":");

signatureBuilder.append(expires);

signatureBuilder.append(":");

signatureBuilder.append(userDetails.getPassword());

signatureBuilder.append(":");

signatureBuilder.append(TokenUtils.***MAGIC\_KEY***);

MessageDigest digest;

**try** {

digest = MessageDigest.*getInstance*("MD5");

} **catch** (NoSuchAlgorithmException e) {

**throw** **new** IllegalStateException("No MD5 algorithm available!");

}

**return** **new** String(Hex.*encode*(digest.digest(signatureBuilder.toString().getBytes())));

}

**public** **static** **boolean** isTokenValid(String authToken, UserDetails userDetails) {

**boolean** flag = **true**;

String[] parts = authToken.split(":");

**long** expires = Long.*parseLong*(parts[1]);

System.***out***.println("Expires token :::" + expires);

String signature = parts[2];

System.***out***.println("Signature-------->" + signature);

**if** (*tokenMap*.containsKey(userDetails) && *tokenMap*.get(userDetails).equals(authToken)) {

**boolean** tempFlag = expires > System.*currentTimeMillis*();

System.***out***.println("Temp Flag --->" + tempFlag);

System.***out***.println("Computed Signature---->" + TokenUtils.*computeSignature*(userDetails, expires));

**if** (expires > System.*currentTimeMillis*()

&& signature.equals(TokenUtils.*computeSignature*(userDetails, expires))) {

flag = **true**;

} **else** {

flag = **false**;

}

} **else** {

flag = **false**;

}

**return** flag;

}

**public** **static** String extractAuthTokenFromRequest(HttpServletRequest httpRequest) {

/\* Get token from header \*/

String authToken = httpRequest.getHeader("X-Auth-Token");

System.***out***.println("Actual authToken ->" + authToken);

/\* If token not found get it from request parameter \*/

**if** (authToken == **null**) {

authToken = httpRequest.getParameter("token");

System.***out***.println("Auth Token from token ->" + authToken);

}

**if** (authToken == **null**) {

// Get it from cookie

authToken = *getCookieValueByName*(httpRequest, "APPASSERTION");

System.***out***.println("Auth Token from Cookie ->" + authToken);

}

**return** authToken;

}

**private** **static** String getCookieValueByName(HttpServletRequest httpRequest, String cookieName) {

String cookieValue = **null**;

String completeCookies = httpRequest.getHeader("Cookie");

**if** (completeCookies != **null**) {

completeCookies = completeCookies.replaceAll("\"", "");

**if** (completeCookies != **null**) {

String[] cooks = completeCookies.split(";");

Map<String, String> cookieMap = **new** HashMap<String, String>();

**for** (String cookie : cooks) {

String[] keyVal = cookie.split("=", 2);

cookieMap.put(keyVal[0].trim(), keyVal[1].trim());

}

cookieValue = cookieMap.get(cookieName);

}

}

// Decode the Cookie Token

**if** (cookieValue != **null**)

cookieValue = **new** String(Base64.*decodeBase64*(cookieValue.getBytes()));

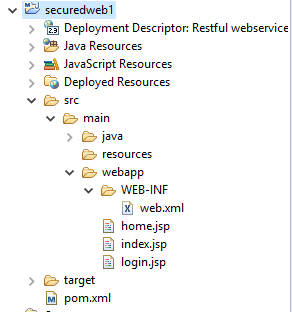
**return** cookieValue;

}

}

# Web Layer

# Project Structure



# Maven Configuration

<?xml version=*"1.0"*?>

<project xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"* xmlns=*"http://maven.apache.org/POM/4.0.0"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*>

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>springsecuritycookietoken1</groupId>

<artifactId>springsecuritycookietoken1</artifactId>

<version>0.0.1-SNAPSHOT</version>

</parent>

<artifactId>securedweb1</artifactId>

<packaging>war</packaging>

<name>securedweb1 Maven Webapp</name>

<url>http://maven.apache.org</url>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.8.1</version>

<scope>test</scope>

</dependency>

**<dependency>**

**<groupId>springsecuritycookietoken1</groupId>**

**<artifactId>cookietokensecurity1</artifactId>**

**<version>0.0.1-SNAPSHOT</version>**

**</dependency>**

</dependencies>

<build>

<finalName>securedweb1</finalName>

</build>

</project>

# Deployment Descriptor (web.xml)

<!DOCTYPE web-app PUBLIC

"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"

"http://java.sun.com/dtd/web-app\_2\_3.dtd" >

<web-app>

<display-name>Restful webservice with Jersey and Spring with OAuth</display-name>

**<context-param>**

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

**<param-value>classpath:security-Context.xml</param-value>**

**</context-param>**

<!-- Spring Security -->

**<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>

<listener>

<listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>

</listener>

<servlet>

<servlet-name>com.ddlab.rnd.spring.token.resources.ApplicationPkgs</servlet-name>

<servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>

<init-param>

<param-name>javax.ws.rs.Application</param-name>

<param-value>com.ddlab.rnd.spring.token.resources.ApplicationPkgs</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>com.ddlab.rnd.spring.token.resources.ApplicationPkgs</servlet-name>

<url-pattern>/api/\*</url-pattern>

</servlet-mapping>

<welcome-file-list>

<welcome-file>index.jsp</welcome-file>

</welcome-file-list>

</web-app>

# All JSP Files

### **Index.jsp**

<html>

<body>

<h2>Hello World!</h2>

</body>

<a href=*"/securedweb1/login.jsp"*>Login</a>

</html>

### **Login.jsp**

<%@ taglib prefix=*"c"* uri=*"http://java.sun.com/jsp/jstl/core"* %>

<html>

<head>

<title>Login</title>

</head>

<body>

<c:if test=*"*${**not empty** param.error}*"*>

<font color=*"red"*>

Login error. <br />

Reason : ${sessionScope["SPRING\_SECURITY\_LAST\_EXCEPTION"].message}

</font>

</c:if>

<form method=*"POST"* action=*"*<c:url value=*"/api/1/authservices/login"* />*"*>

<table>

<tr>

<td align=*"right"*>Username</td>

<td><input type=*"text"* name=*"j\_username"* /></td>

</tr>

<tr>

<td align=*"right"*>Password</td>

<td><input type=*"password"* name=*"j\_password"* /></td>

</tr>

<tr>

<td colspan=*"2"* align=*"right"*>

<input type=*"submit"* value=*"Login"* />

</td>

</tr>

</table>

</form>

</body>

</html>

### **Home.jsp**

<%@ taglib prefix=*"c"* uri=*"http://java.sun.com/jsp/jstl/core"* %>

<html>

<head>

<title>Home</title>

</head>

<body>

<a href=<c:url value=*"/api/1/authservices/logout"*/>>Logout</a><br/>

<!-- <sec:authorize ifAnyGranted="ROLE\_ADMIN">

<h1>Only admin can see this</h1><br/>

<a href="admin"> Admin Home </a>

</sec:authorize> -->

<h1>Welcome, This page is for normal user</h1>

</body>

</html>

# REST layer

## **ApplicationPkgs.java**

**package** com.ddlab.rnd.spring.token.resources;

**import** org.glassfish.jersey.jackson.JacksonFeature;

**import** org.glassfish.jersey.server.ResourceConfig;

**import** com.ddlab.rnd.spring.token.auth.AuthenticationResource;

**public** **class** ApplicationPkgs **extends** ResourceConfig {

**public** ApplicationPkgs() {

**super**(BankResources.**class**,JacksonFeature.**class**,AuthenticationResource.**class**);

}

}

## **BaseResource.java**

**package** com.ddlab.rnd.spring.token.resources;

**import** java.nio.file.AccessDeniedException;

**import** javax.ws.rs.NotFoundException;

**import** javax.ws.rs.WebApplicationException;

**import** javax.ws.rs.core.Response;

**import** javax.ws.rs.core.Response.Status;

**public** **class** BaseResource {

**protected** WebApplicationException createWebappException(Exception incomingException) {

Status status;

String string = **null**;

**if** (incomingException **instanceof** SecurityException || incomingException **instanceof** AccessDeniedException) {

status = Status.***FORBIDDEN***;

} **else** **if** (incomingException **instanceof** IllegalArgumentException) {

status = Status.***BAD\_REQUEST***;

string=incomingException.getMessage();

} **else** **if** (incomingException **instanceof** NotFoundException) {

status = Status.***NOT\_FOUND***;

} **else** **if** (incomingException **instanceof** Exception) {

status = Status.***INTERNAL\_SERVER\_ERROR***;

} **else** {

status = Status.***INTERNAL\_SERVER\_ERROR***;

}

**return** **new** WebApplicationException(Response.*status*(status).entity(string).type("text/plain").build());

}

}

## **BankResources.java**

**package** com.ddlab.rnd.spring.token.resources;

**import** javax.servlet.ServletConfig;

**import** javax.servlet.http.HttpServletRequest;

**import** javax.ws.rs.Consumes;

**import** javax.ws.rs.GET;

**import** javax.ws.rs.Path;

**import** javax.ws.rs.Produces;

**import** javax.ws.rs.QueryParam;

**import** javax.ws.rs.core.Context;

**import** javax.ws.rs.core.MediaType;

**import** javax.ws.rs.core.Response;

@Path("1/bankingservices")

**public** **class** BankResources **extends** BaseResource {

@Context

HttpServletRequest request;

@Context

ServletConfig servletConfig;

@Path("/userid")

@GET

@Consumes({ MediaType.***APPLICATION\_XML***, MediaType.***APPLICATION\_JSON*** })

@Produces({ MediaType.***APPLICATION\_XML***, MediaType.***APPLICATION\_JSON*** })

**public** Response getUser(@QueryParam("id") **int** id) {

**if**(id == 0 )

**throw** createWebappException(**new** IllegalArgumentException("Incorrect ID"));

**return** Response.*ok*().entity("You have successfully configured").build();

}

@Path("/info")

@GET

@Produces({ MediaType.***TEXT\_PLAIN***})

**public** Response getInfo() {

**return** Response.*ok*().entity("Application configured with Spring security ...").build();

}

}

# How to Test

First Authneticate using following rest URL, you will get the token. Use username and password as contentType:x-www-form-urlencoded

**POST http://localhost:8090/securedweb1/api/1/authservices/authenticate**

**username=piku&password=piku**

Invoke the Resource

**GET http://localhost:8090/securedweb1/api/1/bankingservices/userid?id=123**

**X-Auth-Token : piku:1495317439680:3a4d1fe6b1ddf8efa83c0d84d0e97325**

Logout

**POST** <http://localhost:8090/securedweb1/api/1/authservices/logout>

**X-Auth-Token : piku:1495317439680:3a4d1fe6b1ddf8efa83c0d84d0e97325**

Login Directly using Username and Password to store the cookies

**POST**

**http://localhost:8090/securedweb1/api/1/authservices/login?j\_username=d&j\_password=d**

The Cookies will be **APPASSERTION=safasasas**

Invoke the Resource

**GET http://localhost:8090/securedweb1/api/1/bankingservices/userid?id=123**

Logout

**GET http://localhost:8090/securedweb1/api/1/authservices/logout**

For Web Access in the browser, try to login with username as d and password as d, then invoke the resource.

http://localhost:8090/securedweb1

<http://localhost:8090/securedweb1/api/1/bankingservices/info>

Method Level Security using Spring standalone

# Maven Configuration (pom.xml)

<project xmlns=*"http://maven.apache.org/POM/4.0.0"* xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xsi:schemaLocation=*"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"*>

<modelVersion>4.0.0</modelVersion>

<groupId>springsecuritywithcustomroles</groupId>

<artifactId>springsecuritywithcustomroles</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>springsecuritywithcustomroles</name>

<url>http://maven.apache.org</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<spring.version>3.2.8.RELEASE</spring.version>

<spring.security.version>3.2.3.RELEASE</spring.security.version>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.0</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>commons-logging</groupId>

<artifactId>commons-logging</artifactId>

<version>1.2</version>

</dependency>

<!-- Spring Security, it will automatically download all Spring libraries -->

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-web</artifactId>

<version>${spring.security.version}</version>

</dependency>

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-config</artifactId>

<version>${spring.security.version}</version>

</dependency>

<dependency>

<groupId>org.springframework.security</groupId>

<artifactId>spring-security-taglibs</artifactId>

<version>${spring.security.version}</version>

</dependency>

</dependencies>

</project>

# Spring Security Configuration (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:aop=*"http://www.springframework.org/schema/aop"*

xmlns:c=*"http://www.springframework.org/schema/c"* xmlns:context=*"http://www.springframework.org/schema/context"*

xmlns:security=*"http://www.springframework.org/schema/security"*

xmlns:util=*"http://www.springframework.org/schema/util"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-3.1.xsd*

*http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-3.1.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-3.1.xsd*

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

*http://www.springframework.org/schema/util http://www.springframework.org/schema/util/spring-util-3.1.xsd"*>

<security:global-method-security secured-annotations=*"enabled"* pre-post-annotations=*"enabled"*>

<security:expression-handler ref=*"defaultMethodSecurityExpressionHandler"* />

</security:global-method-security>

<bean id=*"roleHierarchy"*

class=*"org.springframework.security.access.hierarchicalroles.RoleHierarchyImpl"*>

<property name=*"hierarchy"*>

<value>

ROLE\_ADMIN > ROLE\_MANAGER

ROLE\_MANAGER > ROLE\_TELLER

ROLE\_TELLER > ROLE\_USER

</value>

</property>

</bean>

<bean id=*"defaultMethodSecurityExpressionHandler"*

class=*"org.springframework.security.access.expression.method.DefaultMethodSecurityExpressionHandler"*>

<property name=*"roleHierarchy"* ref=*"roleHierarchy"* />

</bean>

<!-- <security:global-method-security secured-annotations="enabled" pre-post-annotations="enabled">

<security:expression-handler ref="defaultMethodSecurityExpressionHandler"

/> </security:global-method-security> -->

<bean id=*"bankTxn"* class=*"com.ddlab.rnd.spring.security.BankTransactionImpl"* />

</beans>

# Java Files

## **BankTrasaction.java**

**package** com.ddlab.rnd.spring.security;

**import** org.springframework.security.access.annotation.Secured;

**import** org.springframework.security.access.prepost.PreAuthorize;

**public** **interface** BankTransaction {

/\*\*

\* Can be done by Manager and Admin

\*/

@PreAuthorize ("hasRole('ROLE\_MANAGER')")

**public** **void** approveLoan(String actNo);

/\*\*

\* Can be done only by USER

\*/

@Secured("ROLE\_USER")

**public** **void** openAccount(String customerName);

/\*\*

\* Can be done by Manager and Admin

\*/

@PreAuthorize ("hasRole('ROLE\_MANAGER')")

**public** **void** resetNetBankingPassword(String customerName);

/\*\*

\* Can be done by Teller,Manager and Admin

\*/

@PreAuthorize ("hasRole('ROLE\_TELLER')")

**public** **void** checkAccountStatus(String actNo);

/\*\*

\* Can be done by anybody in bank other than Normal user or customer

\*/

@Secured({"ROLE\_ADMIN", "ROLE\_MANAGER" , "ROLE\_TELLER"})

**public** **void** suggestCreditCard(String customerName);

/\*\*

\* Can be approved only by Manager and not by any other persons

\*/

@Secured({"ROLE\_MANAGER"})

**public** **void** approveOverDraftFacility(String customerName);

}

## **BankTractionImpl.java**

**package** com.ddlab.rnd.spring.security;

**public** **class** BankTransactionImpl **implements** BankTransaction {

**public** **void** approveLoan(String actNo) {

System.***out***.println("The loan has been approved for the account no : " + actNo);

}

**public** **void** openAccount(String customerName) {

System.***out***.println("Hi " + customerName + " , your account will be activated within 24 hours");

}

**public** **void** resetNetBankingPassword(String customerName) {

System.***out***.println("Hi " + customerName + ", Administrator has rest your internet banking password "

+ "and an email has been sent you");

}

**public** **void** checkAccountStatus(String actNo) {

System.***out***.println("You available balance is 2000 INR");

}

**public** **void** suggestCreditCard(String customerName) {

System.***out***.println("Hi " + customerName + " , a life time free Titanium credit card is available for you");

}

**public** **void** approveOverDraftFacility(String customerName) {

System.***out***.println("Hi " + customerName + " , a overdraft facility is approved for you");

}

}

## **ROLETYPE.java**

**package** com.ddlab.rnd.spring.security;

**public** **enum** ROLETYPE {

***USER***("ROLE\_USER") , ***ADMIN***("ROLE\_ADMIN") , ***MANAGER***("ROLE\_MANAGER") , ***TELLER***("ROLE\_TELLER");

// USER("USER") , ADMIN("ADMIN") , MANAGER("MANAGER") , TELLER("TELLER");

**private** String role;

**private** ROLETYPE(String role) {

**this**.role = role;

}

@Override

**public** String toString() {

**return** role;

}

}

## **SecurityUtil.java**

**package** com.ddlab.rnd.spring.security;

**import** java.util.List;

**import** org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

**import** org.springframework.security.core.Authentication;

**import** org.springframework.security.core.authority.SimpleGrantedAuthority;

**import** org.springframework.security.core.context.SecurityContextHolder;

**import** org.springframework.security.core.userdetails.User;

**public** **class** SecurityUtil {

**public** **static** **void** loginAs(String userName, String password, ROLETYPE roleType) {

*setSecurity*(userName, password, roleType.toString());

}

**public** **static** **void** logout() {

// SecurityContextHolder.getContext().setAuthentication(null);

// SecurityContextHolder.getContext().getAuthentication().setAuthenticated(false);

SecurityContextHolder.*clearContext*();

}

**private** **static** **void** setSecurity(String userName, String password,String roleType) {

List<SimpleGrantedAuthority> auths = **new** java.util.ArrayList<SimpleGrantedAuthority>();

auths.add(**new** SimpleGrantedAuthority(roleType));

User user = **new** User(userName, password, auths);

Authentication authToken = **new** UsernamePasswordAuthenticationToken (user.getUsername(), user.getPassword(), user.getAuthorities());

SecurityContextHolder.*getContext*().setAuthentication(authToken);

}

}

## **App.java**

**package** com.ddlab.rnd.spring.security;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**public** **class** App

{

**public** **static** **void** main( String[] args )

{

// ApplicationContext applicationContext = new ClassPathXmlApplicationContext(new String[]{"beans1.xml","application-security.xml"});

ApplicationContext applicationContext = **new** ClassPathXmlApplicationContext("beans.xml");

// SecurityUtil.loginAs("Deb", "pqrs", "ROLE\_USER");

// SecurityUtil.loginAs("Deb", "pqrs", "ROLE\_MANAGER");

// SecurityUtil.loginAs("Deb", "pqrs", ROLETYPE.TELLER);

SecurityUtil.*loginAs*("Deb", "pqrs", ROLETYPE.***MANAGER***);

BankTransaction bankTxn = (BankTransaction) applicationContext.getBean("bankTxn");

bankTxn.approveLoan("12345");

// bankTxn.openAccount("Deb");

// bankTxn.checkAccountStatus("12345");

}

}

## **Test.java**

**package** com.ddlab.rnd.spring.security;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**public** **class** Test {

//Person having designation more than MANAGER

**public** **static** **void** approveLoan(BankTransaction bankTxn) {

// SecurityUtil.loginAs("Deb", "pqrs", ROLETYPE.MANAGER);

SecurityUtil.*loginAs*("Deb", "pqrs", ROLETYPE.***ADMIN***);

bankTxn.approveLoan("12345");

}

**public** **static** **void** openAccount(BankTransaction bankTxn) {

SecurityUtil.*loginAs*("Deb", "pqrs", ROLETYPE.***USER***);

bankTxn.openAccount("Deb");

}

**public** **static** **void** main(String[] args) {

ApplicationContext applicationContext = **new** ClassPathXmlApplicationContext("beans.xml");

BankTransaction bankTxn = (BankTransaction) applicationContext.getBean("bankTxn");

*approveLoan*(bankTxn);

*openAccount*(bankTxn);

}

}

# Unit Test Cases

## **TestBankTransaction.java**

**package** com.ddlab.rnd.spring.security;

**import** org.junit.BeforeClass;

**import** org.junit.Test;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**import** org.springframework.security.access.AccessDeniedException;

**import** org.springframework.security.authentication.AuthenticationCredentialsNotFoundException;

**import** org.springframework.security.core.context.SecurityContextHolder;

**public** **class** TestBankTransactions {

**private** **static** BankTransaction *bankTxn* = **null**;

@BeforeClass

**public** **static** **void** setup() {

@SuppressWarnings("resource")

ApplicationContext applicationContext = **new** ClassPathXmlApplicationContext("beans.xml");

*bankTxn* = (BankTransaction) applicationContext.getBean("bankTxn");

}

/\*\*

\* Loan can not be approved by Normal user

\*/

@Test(expected=AccessDeniedException.**class**)

**public** **void** testApproveLoanByNormalUser() {

SecurityUtil.*logout*();

SecurityUtil.*loginAs*("Deb", "pqrs", ROLETYPE.***USER***);

*bankTxn*.approveLoan("12345");

}

/\*\*

\* Loan can be approved by Admin

\*/

@Test

**public** **void** testApproveLoanByAdmin() {

SecurityUtil.*logout*();

SecurityUtil.*loginAs*("Deb", "pqrs", ROLETYPE.***ADMIN***);

*bankTxn*.approveLoan("12345");

}

/\*\*

\* Loan can not be approved by Teller

\*/

@Test(expected=AccessDeniedException.**class**)

**public** **void** testApproveLoanByTeller() {

SecurityUtil.*logout*();

SecurityUtil.*loginAs*("Deb", "pqrs", ROLETYPE.***TELLER***);

*bankTxn*.approveLoan("12345");

}

/\*\*

\* Loan can be approved by Manager

\*/

@Test

**public** **void** testApproveLoanByManager() {

SecurityUtil.*logout*();

SecurityUtil.*loginAs*("Deb", "pqrs", ROLETYPE.***MANAGER***);

*bankTxn*.approveLoan("12345");

}

/\*\*

\* Loan can not be approved by unknown user

\*/

@Test(expected = AuthenticationCredentialsNotFoundException.**class**)

**public** **void** testApproveLoanAnonymously() {

SecurityUtil.*logout*();

System.***out***.println("----->"+SecurityContextHolder.*getContext*().getAuthentication());

*bankTxn*.approveLoan("12345");

}

//--------------------

//All bank employee suggest a customer for a free credit card

@Test

**public** **void** testSuggestForCreditCardLoanByManager() {

SecurityUtil.*logout*();

SecurityUtil.*loginAs*("Deb", "pqrs", ROLETYPE.***MANAGER***);

*bankTxn*.suggestCreditCard("John Abraham");

}

@Test

**public** **void** testSuggestForCreditCardLoanByAdmin() {

SecurityUtil.*logout*();

SecurityUtil.*loginAs*("Deb", "pqrs", ROLETYPE.***ADMIN***);

*bankTxn*.suggestCreditCard("John Abraham");

}

@Test

**public** **void** testSuggestForCreditCardLoanByTeller() {

SecurityUtil.*logout*();

SecurityUtil.*loginAs*("Deb", "pqrs", ROLETYPE.***TELLER***);

*bankTxn*.suggestCreditCard("John Abraham");

}

@Test(expected=AccessDeniedException.**class**)

**public** **void** testSuggestForCreditCardLoanByUser() {

SecurityUtil.*logout*();

SecurityUtil.*loginAs*("Deb", "pqrs", ROLETYPE.***USER***);

*bankTxn*.suggestCreditCard("John Abraham");

}

//Overdraft facility can only be approved by Bank Manager only and not by any other

@Test(expected=AccessDeniedException.**class**)

**public** **void** testOverDraftFacilityByUser() {

SecurityUtil.*logout*();

SecurityUtil.*loginAs*("Deb", "pqrs", ROLETYPE.***USER***);

*bankTxn*.approveOverDraftFacility("John Abraham");

}

@Test(expected=AccessDeniedException.**class**)

**public** **void** testOverDraftFacilityByTeller() {

SecurityUtil.*logout*();

SecurityUtil.*loginAs*("Deb", "pqrs", ROLETYPE.***TELLER***);

*bankTxn*.approveOverDraftFacility("John Abraham");

}

@Test(expected=AccessDeniedException.**class**)

**public** **void** testOverDraftFacilityByAdmin() {

SecurityUtil.*logout*();

SecurityUtil.*loginAs*("Deb", "pqrs", ROLETYPE.***ADMIN***);

*bankTxn*.approveOverDraftFacility("John Abraham");

}

@Test

**public** **void** testOverDraftFacilityByManager() {

SecurityUtil.*logout*();

SecurityUtil.*loginAs*("Deb", "pqrs", ROLETYPE.***MANAGER***);

*bankTxn*.approveOverDraftFacility("John Abraham");

}

//A user can open account but bank employees should not open in the same bank

@Test

**public** **void** testOpenAccountByUser() {

SecurityUtil.*logout*();

SecurityUtil.*loginAs*("Deb", "pqrs", ROLETYPE.***USER***);

*bankTxn*.openAccount("John Abraham");

}

@Test(expected=AccessDeniedException.**class**)

**public** **void** testOpenAccountByAdmin() {

SecurityUtil.*logout*();

SecurityUtil.*loginAs*("Deb", "pqrs", ROLETYPE.***ADMIN***);

*bankTxn*.openAccount("John Abraham");

}

@Test(expected=AccessDeniedException.**class**)

**public** **void** testOpenAccountByManager() {

SecurityUtil.*loginAs*("Deb", "pqrs", ROLETYPE.***MANAGER***);

*bankTxn*.openAccount("John Abraham");

}

// @Test(expected=AccessDeniedException.class)

**public** **void** testOpenAccountByTeller() {

SecurityUtil.*logout*();

// SecurityUtil.loginAs("Deb", "pqrs", ROLETYPE.TELLER);

SecurityUtil.*loginAs*("Deb", "pqrs", ROLETYPE.***USER***);

*bankTxn*.openAccount("John Abraham");

}

//Rest user's Internet banking password

@Test(expected=AccessDeniedException.**class**)

**public** **void** testResetNetBankingPasswordByTeller() {

SecurityUtil.*loginAs*("Deb", "pqrs", ROLETYPE.***TELLER***);

*bankTxn*.resetNetBankingPassword("John Abraham");

}

@Test(expected=AccessDeniedException.**class**)

**public** **void** testResetNetBankingPasswordByUser() {

SecurityUtil.*loginAs*("Deb", "pqrs", ROLETYPE.***USER***);

*bankTxn*.resetNetBankingPassword("John Abraham");

}

@Test

**public** **void** testResetNetBankingPasswordByManager() {

SecurityUtil.*loginAs*("Deb", "pqrs", ROLETYPE.***MANAGER***);

*bankTxn*.resetNetBankingPassword("John Abraham");

}

@Test

**public** **void** testResetNetBankingPasswordByAdmin() {

SecurityUtil.*loginAs*("Deb", "pqrs", ROLETYPE.***ADMIN***);

*bankTxn*.resetNetBankingPassword("John Abraham");

}

//Account status can be checked by Bank Manager or Teller

@Test

**public** **void** testCheckAccountStatusByAdmin() {

SecurityUtil.*loginAs*("Deb", "pqrs", ROLETYPE.***ADMIN***);

*bankTxn*.checkAccountStatus("12345");

}

@Test

**public** **void** testCheckAccountStatusByManager() {

SecurityUtil.*loginAs*("Deb", "pqrs", ROLETYPE.***MANAGER***);

*bankTxn*.checkAccountStatus("12345");

}

@Test

**public** **void** testCheckAccountStatusByTeller() {

SecurityUtil.*loginAs*("Deb", "pqrs", ROLETYPE.***TELLER***);

*bankTxn*.checkAccountStatus("12345");

}

@Test(expected=AccessDeniedException.**class**)

**public** **void** testCheckAccountStatusByUser() {

SecurityUtil.*loginAs*("Deb", "pqrs", ROLETYPE.***USER***);

*bankTxn*.checkAccountStatus("12345");

}

}

# Stateless Vs Stateful in Spring Security ([Stateless with cookie vs statefull](http://stackoverflow.com/questions/26705347/stateless-with-cookie-vs-statefull))

<http://stackoverflow.com/questions/26705347/stateless-with-cookie-vs-statefull>

In a purely stateless environment you really dont need sessions or cookies.

Both sessions and cookies are used to maintain state. The only question is where. Cookies maintain the state on the client while sessions maintain state on the server.

Also from [wiki](http://en.wikipedia.org/wiki/Representational_state_transfer)

" Important to note is that the session state can be transferred by the server to another service such as a database to maintain a persistent state for a period and allow authentication. "

So typically in a stateless design, yes there is no state bewtween client requests. Every client request will have sufficient info to perform the requested action.  
However, you still need authentication / authorization so who the client is identified from request headers (typically).

<https://eimg.wordpress.com/2008/05/11/different-between-session-and-cookie/>

[HTTP](http://en.wikipedia.org/wiki/HTTP)**is a**[stateless](http://en.wikipedia.org/wiki/Stateless_server)**protocol**

*HTTP is stateless protocol. A stateless server is a serve that treats each request as an independent transaction that is unrelated to any previous request.*

That mean, the request you make doesn’t associate in any way with the previous one. So, how about the request we want to make frequently, like user name or id? As you know, we could store our data in COOKIE. When we store data in COOKIE, the browser will send the cookie data to server for each request. We already could use SESSION for this kind of task. So, what is difference between SESSION and COOKIE?

**COOKIE**  
A cookie is a text-only string that takes a place in the memory of user’s browser. If the lifetime of the cookie is set to be longer than the time user spends at that site, then this string is saved to file for future reference. User could be disabled the cookie in their browser setting.

**SESSION**  
Session values are store in server side not in user’s machine. A session is available as long as the browser is opened. User couldn’t be disabled the session. We could store not only strings but also objects in session.

**The Differences**  
We got three differences in general. The key difference would be cookies are stored in client side and sessions are stored in server side. The second difference would be cookies can only store strings. We can store our objects in sessions. Storing objects in sessions were really useful according to my experience. Another difference was that we could be save cookie for future reference, but session couldn’t. When users close their browser, they also lost the session.