**What is Spring Security?**

Spring Security provides comprehensive security services for J2EE-based enterprise software applications. There is a particular emphasis on supporting projects built using The Spring Framework, which is the leading J2EE solution for enterprise software development. If you're not using Spring for developing enterprise applications, we warmly encourage you to take a closer look at it. Some familiarity with Spring - and in particular dependency injection principles - will help you get up to speed with Spring Security more easily.

People use Spring Security for many reasons, but most are drawn to the project after finding the security features of J2EE's Servlet Specification or EJB Specification lack the depth required for typical enterprise application scenarios. Whilst mentioning these standards, it's important to recognise that they are not portable at a WAR or EAR level. Therefore, if you switch server environments, it is typically a lot of work to reconfigure your application's security in the new target environment. Using Spring Security overcomes these problems, and also brings you dozens of other useful, customisable security features.

As you probably know two major areas of application security are “authentication” and “authorization” (or “access-control”). These are the two main areas that Spring Security targets. “Authentication” is the process of establishing a principal is who they claim to be (a “principal” generally means a user, device or some other system which can perform an action in your application). “Authorization” refers to the process of deciding whether a principal is allowed to perform an action within your application. To arrive at the point where an authorization decision is needed, the identity of the principal has already been established by the authentication process. These concepts are common, and not at all specific to Spring Security.

At an authentication level, Spring Security supports a wide range of authentication models. Most of these authentication models are either provided by third parties, or are developed by relevant standards bodies such as the Internet Engineering Task Force. In addition, Spring Security provides its own set of authentication features. Specifically, Spring Security currently supports authentication integration with all of these technologies:

* HTTP BASIC authentication headers (an IEFT RFC-based standard)
* HTTP Digest authentication headers (an IEFT RFC-based standard)
* HTTP X.509 client certificate exchange (an IEFT RFC-based standard)
* LDAP (a very common approach to cross-platform authentication needs, especially in large environments)
* Form-based authentication (for simple user interface needs)
* OpenID authentication
* Authentication based on pre-established request headers (such as Computer Associates Siteminder)
* JA-SIG Central Authentication Service (otherwise known as CAS, which is a popular open source single sign on system)
* Transparent authentication context propagation for Remote Method Invocation (RMI) and HttpInvoker (a Spring remoting protocol)
* Automatic "remember-me" authentication (so you can tick a box to avoid re-authentication for a predetermined period of time)
* Anonymous authentication (allowing every call to automatically assume a particular security identity)
* Run-as authentication (which is useful if one call should proceed with a different security identity)
* Java Authentication and Authorization Service (JAAS)
* JEE container autentication (so you can still use Container Managed Authentication if desired)
* Kerberos
* Java Open Source Single Sign On (JOSSO) \*
* OpenNMS Network Management Platform \*
* AppFuse \*
* AndroMDA \*
* Mule ESB \*
* Direct Web Request (DWR) \*
* Grails \*
* Tapestry \*
* JTrac \*
* Jasypt \*
* Roller \*
* Elastic Path \*
* Atlassian Crowd \*

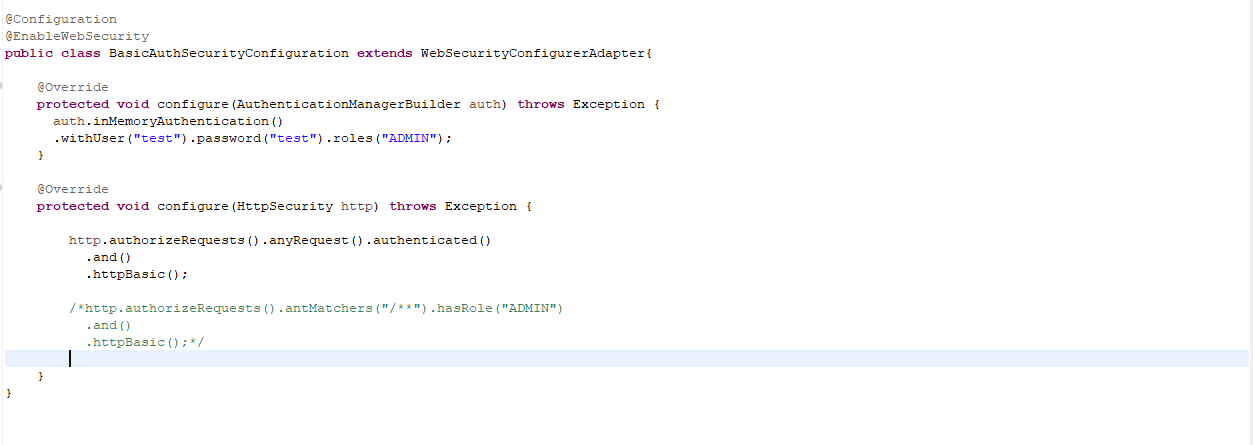
**BASIC AUTHENTICATION**

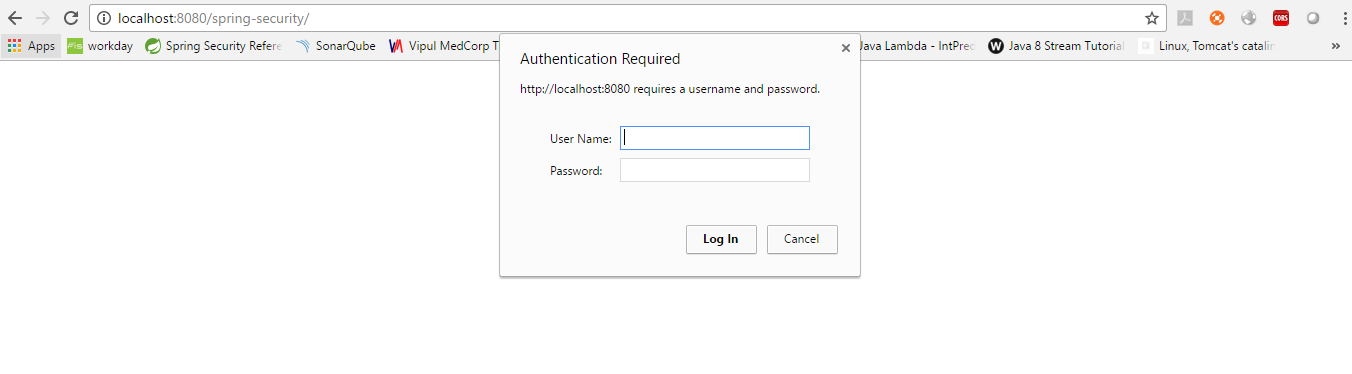
Basic authentification is a standard HTTP header with the user and password encoded in base64 : Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==.The userName and password is encoded in the format username:password. This is one of the simplest technique to protect the REST resources because it does not require cookies. session identifiers or any login pages.

In case of basic authentication, the username and password is only encoded with Base64, but not encrypted or hashed in any way. Hence, it can be compromised by any man in the middle. Hence, it is always recommended to authenticate rest API calls by this header over a ssl connection.

**BasicAuthenticationFilter in Spring**

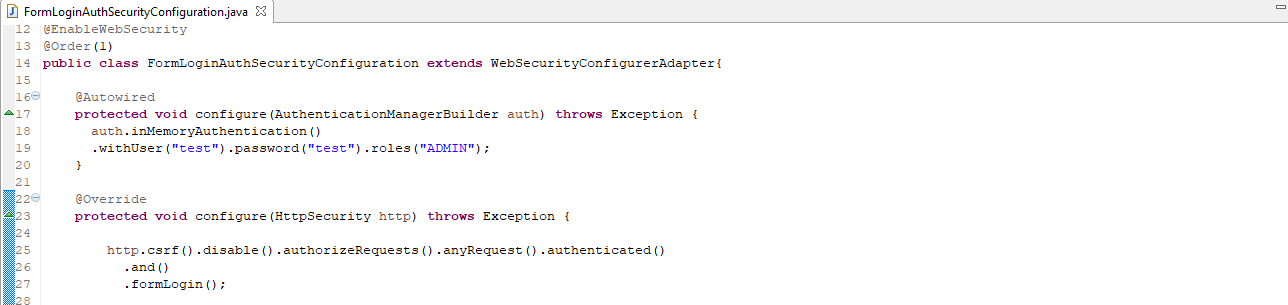
BasicAuthenticationFilter in Spring is the class which is responsible for processing basic authentication credentials presented in HTTP Headers and putting the result into the SecurityContextHolder. The standard governing HTTP Basic Authentication is defined by RFC 1945, Section 11, and BasicAuthenticationFilter confirms with this RFC.





**FORM LOGIN**

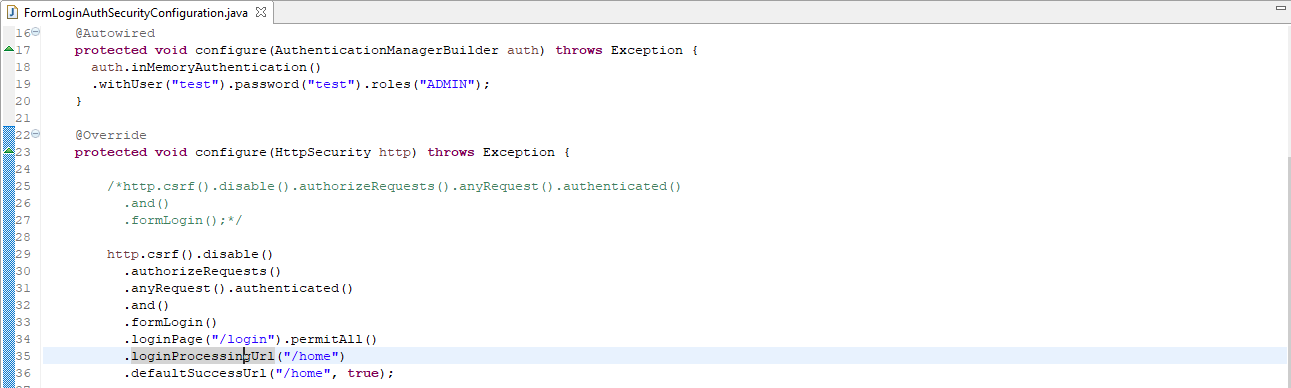
Default login page will be provided by Spring

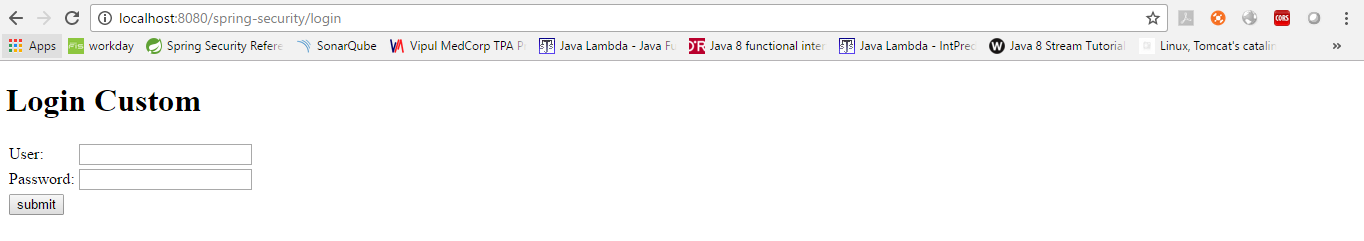


Custom login page

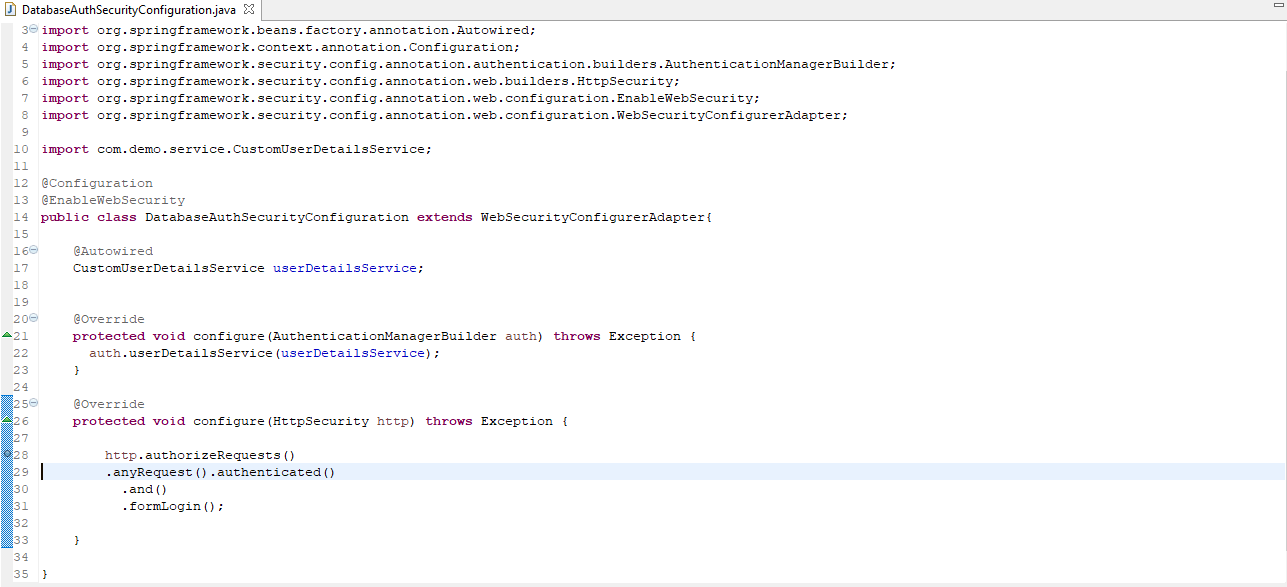
This has several methods that we can use to configure the behavior of the form login:

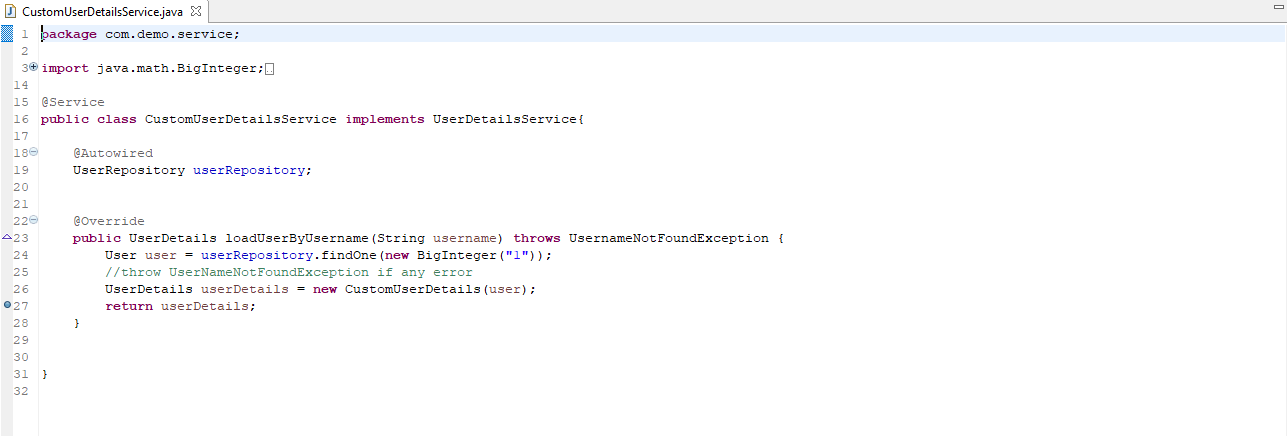
* loginPage() – the custom login page
* loginProcessingUrl() – the url to submit the username and password to
* defaultSuccessUrl() – the landing page after a successful login
* failureUrl() – the landing page after an unsuccessful login

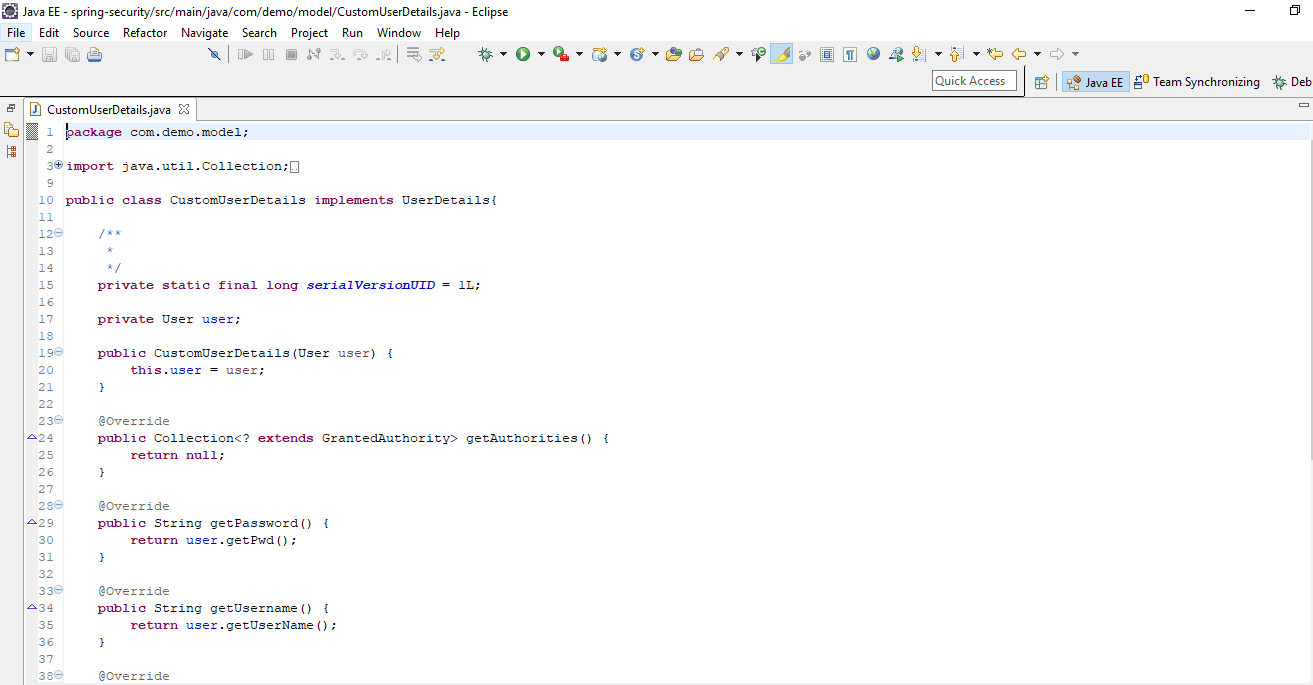




**Database Login**



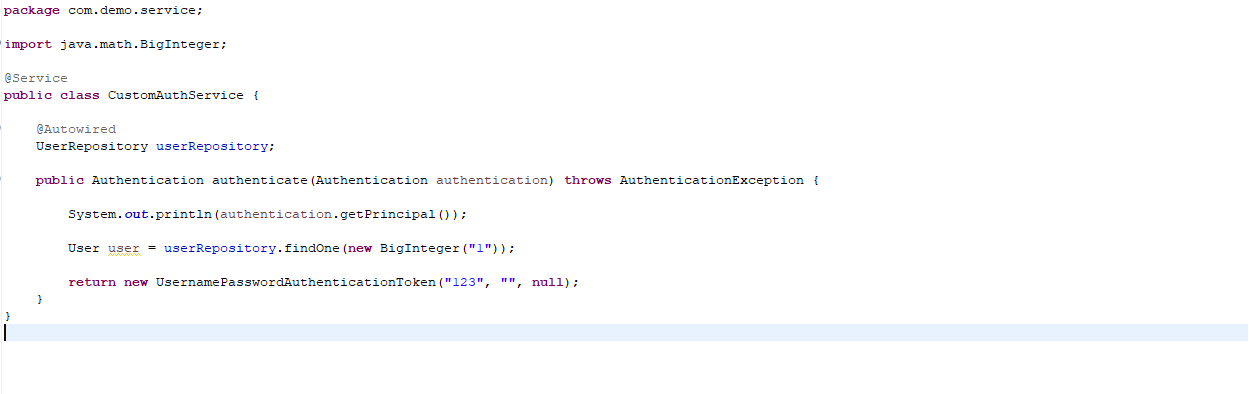




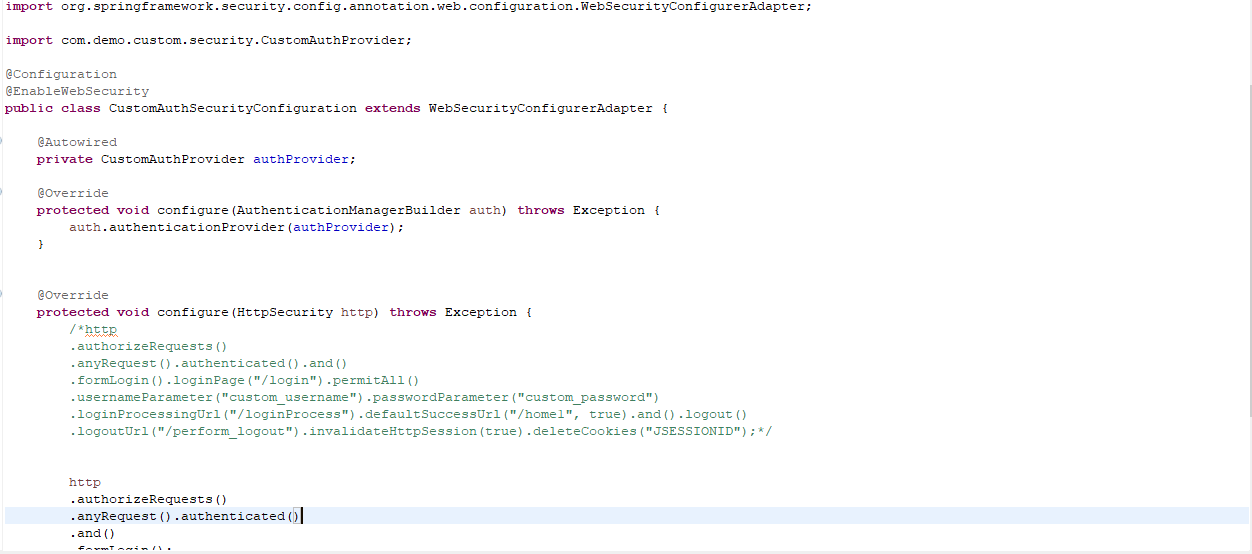
**Custom Authentication Provider**

Create custom auth provider

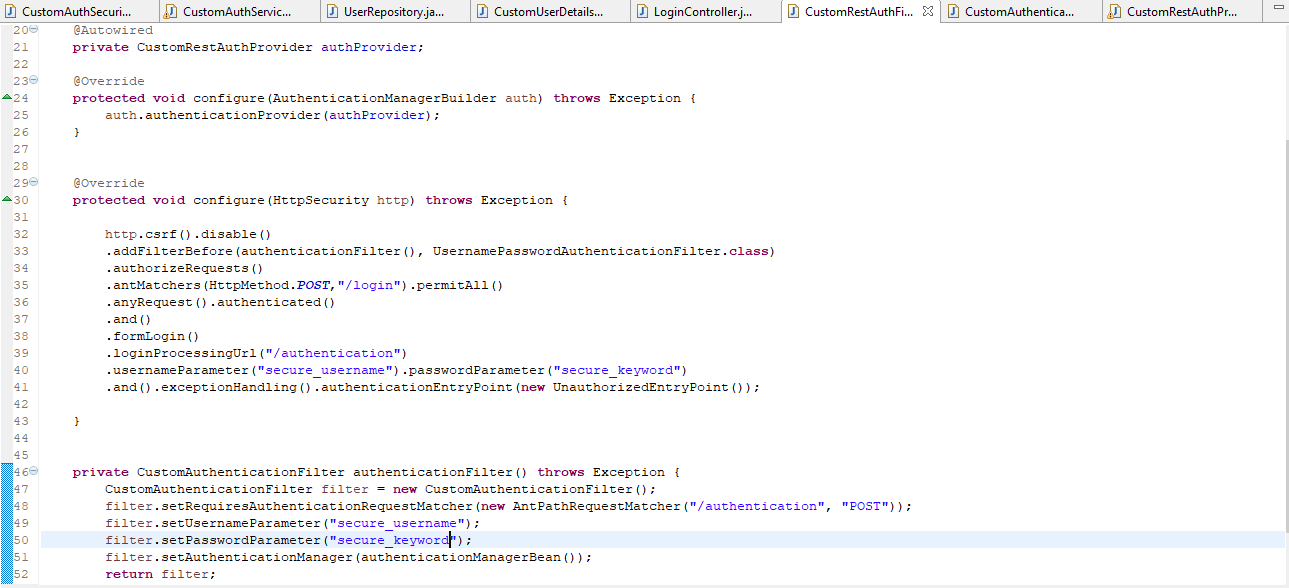


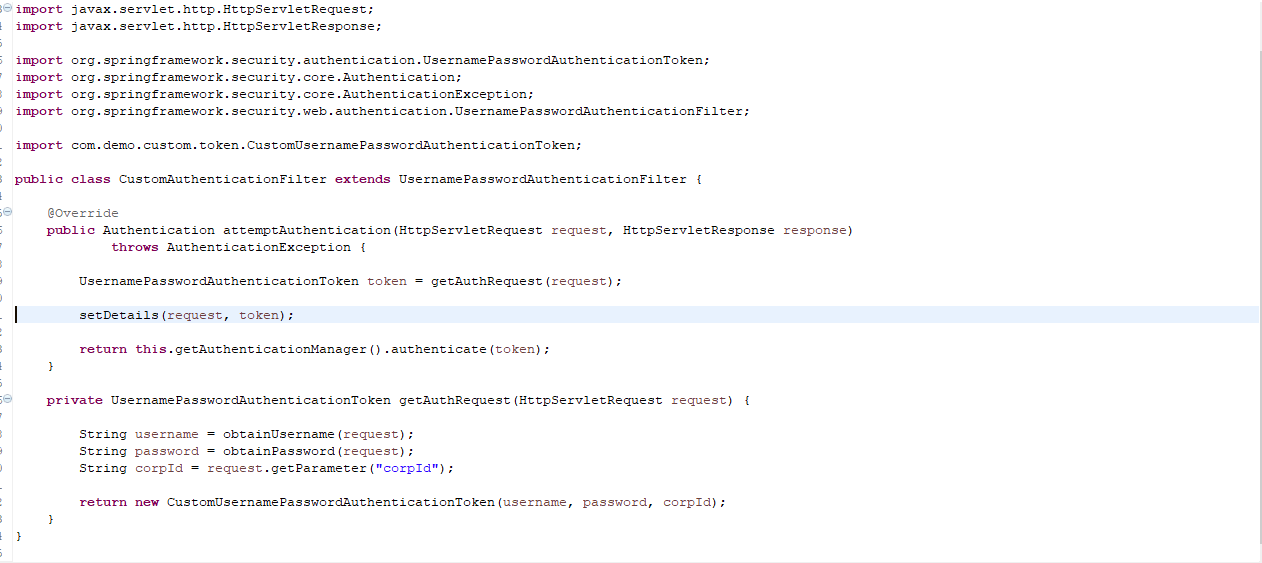


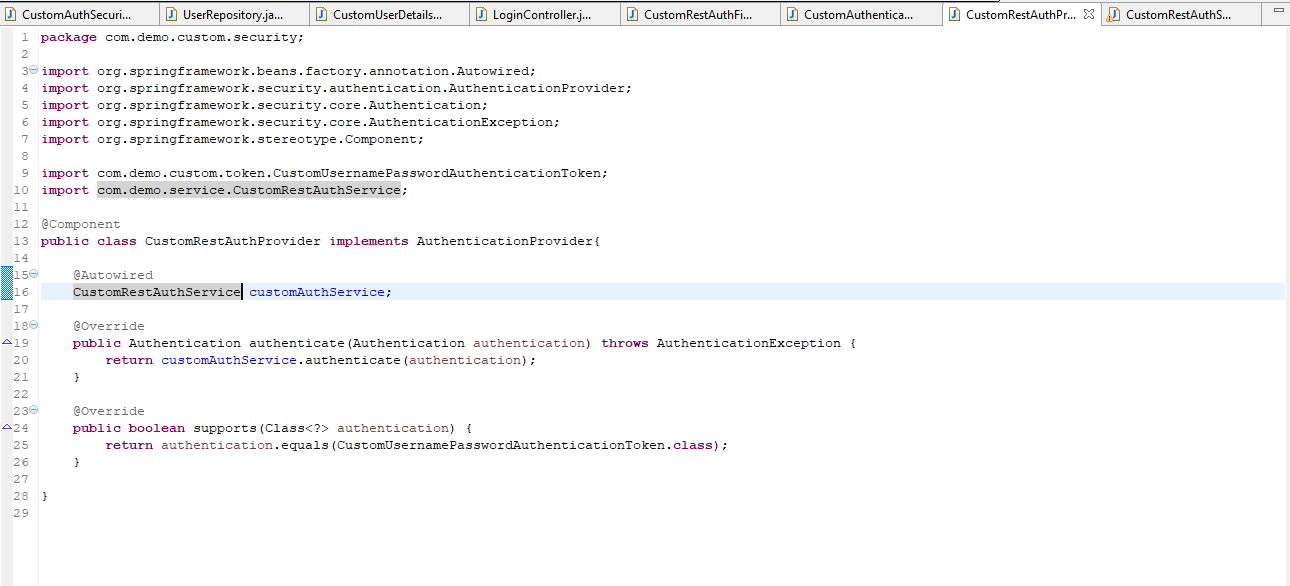
Register this auth provider

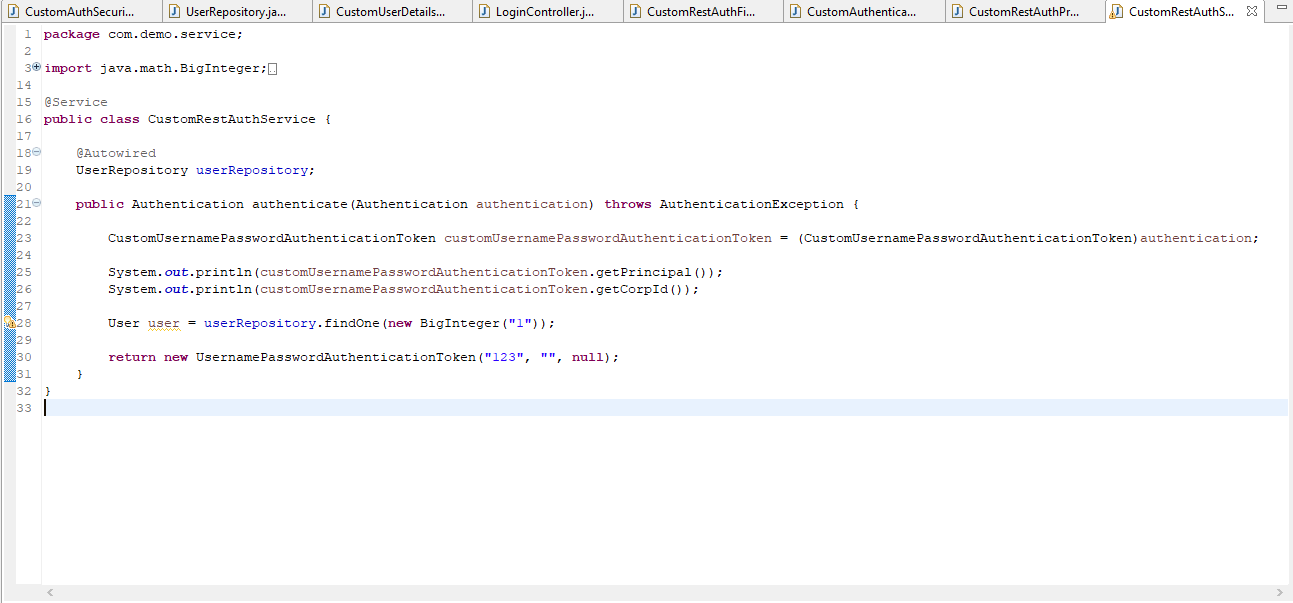


**CUSTOM AUTH FILTER AND TOKEN**

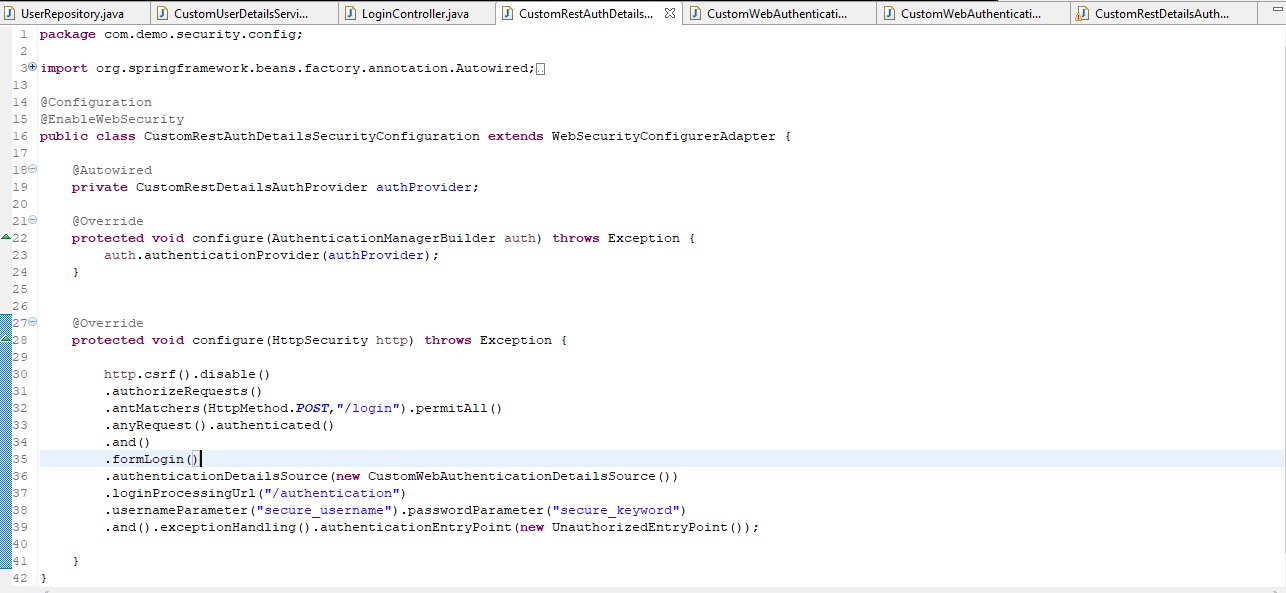


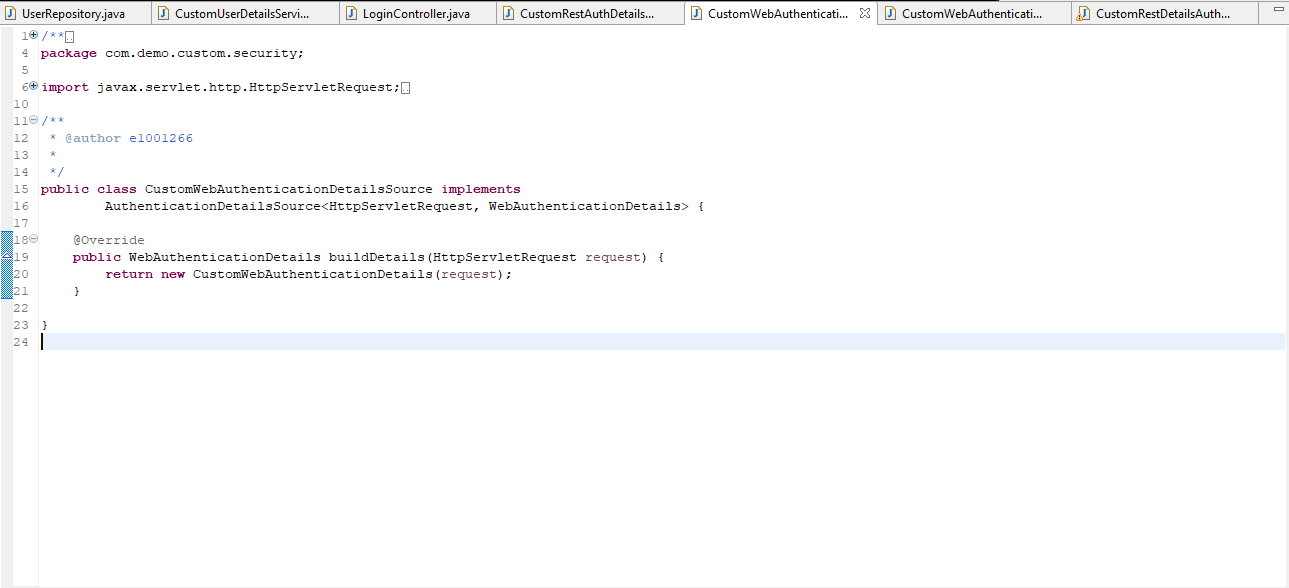


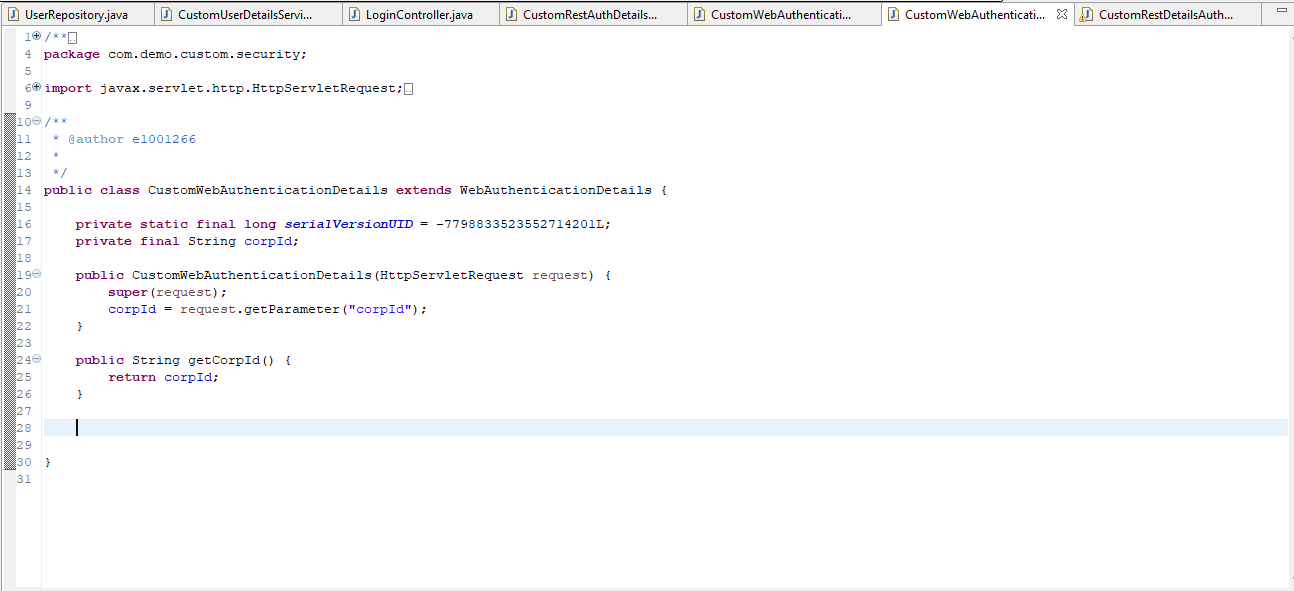




**CUSTOM AUTHENTICATION DETAILS SOURCE**



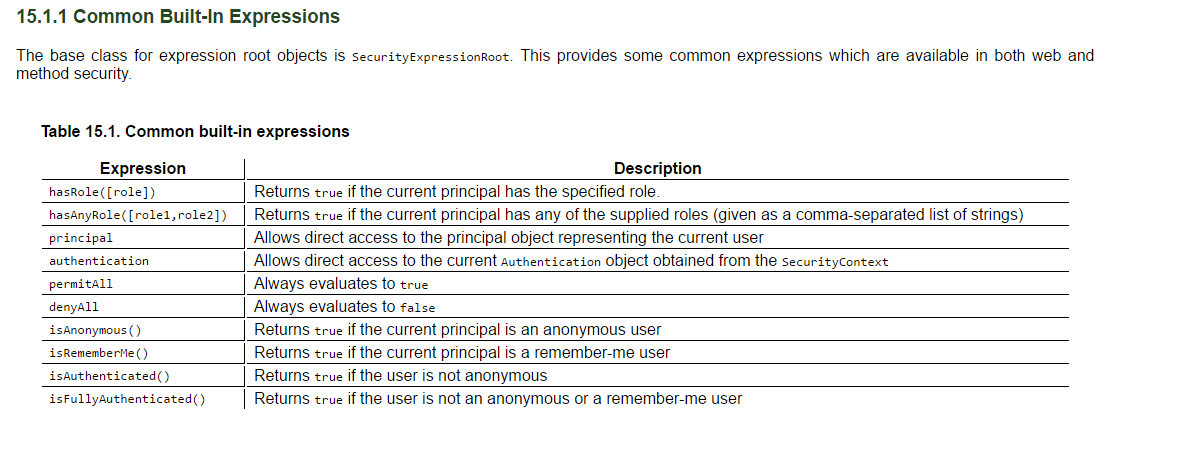


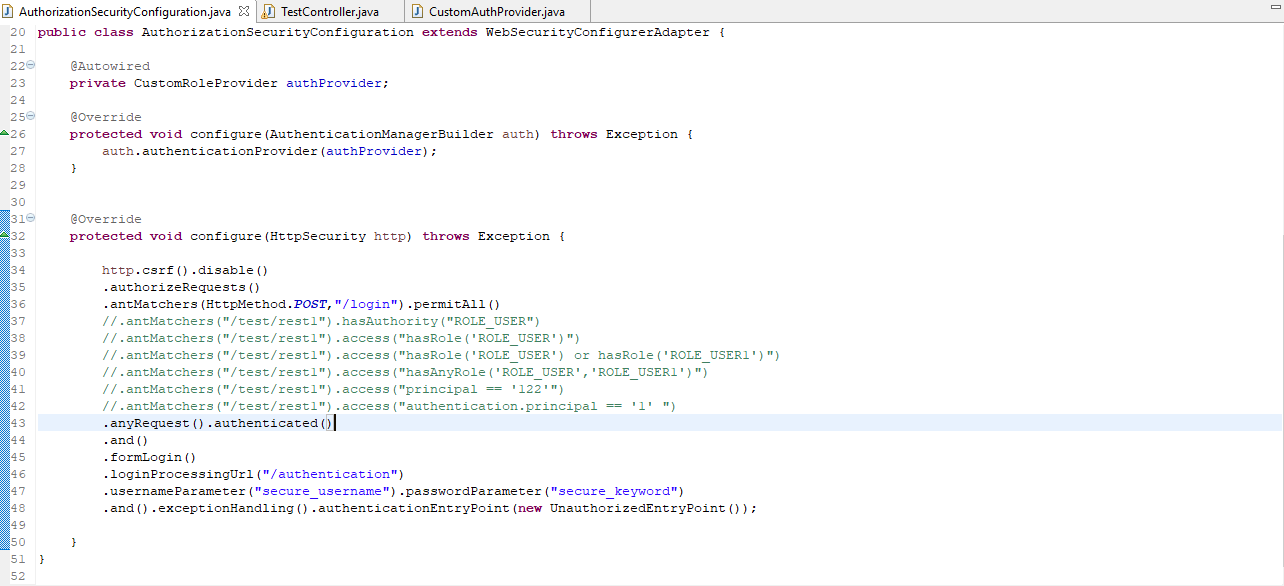




**AUTHORIZATION**

**Expression based access**





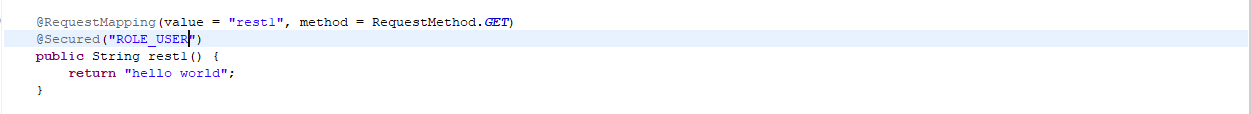
**Method Security Annotations**

Method security is a bit more complicated than a simple allow or deny rule. Spring Security 3.0 introduced some new annotations in order to allow comprehensive support for the use of expressions.

**@Secured**

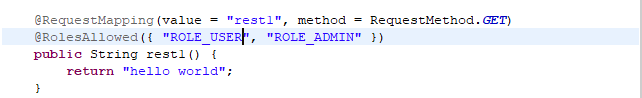
@Secured annotation is used to define a list of security configuration attributes for business methods. You can specify the security requirements[roles/permission etc] on a method using @Secured, and than only the user with those roles/permissions can invoke that method. If anyone tries to invoke a method and does not possess the required roles/permissions, an AccessDenied exception will be thrown.

@Secured is coming from previous versions of Spring. It has a limitation that it does not support Spring EL expressions.



**@RoleAllowed**

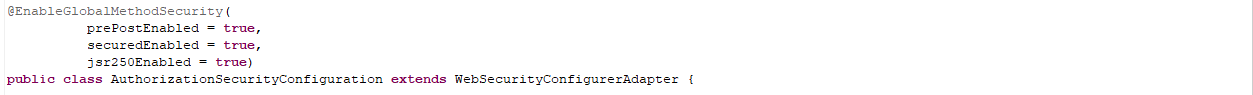
The @RoleAllowed annotation is the JSR-250’s equivalent annotation of the @Secured annotation.

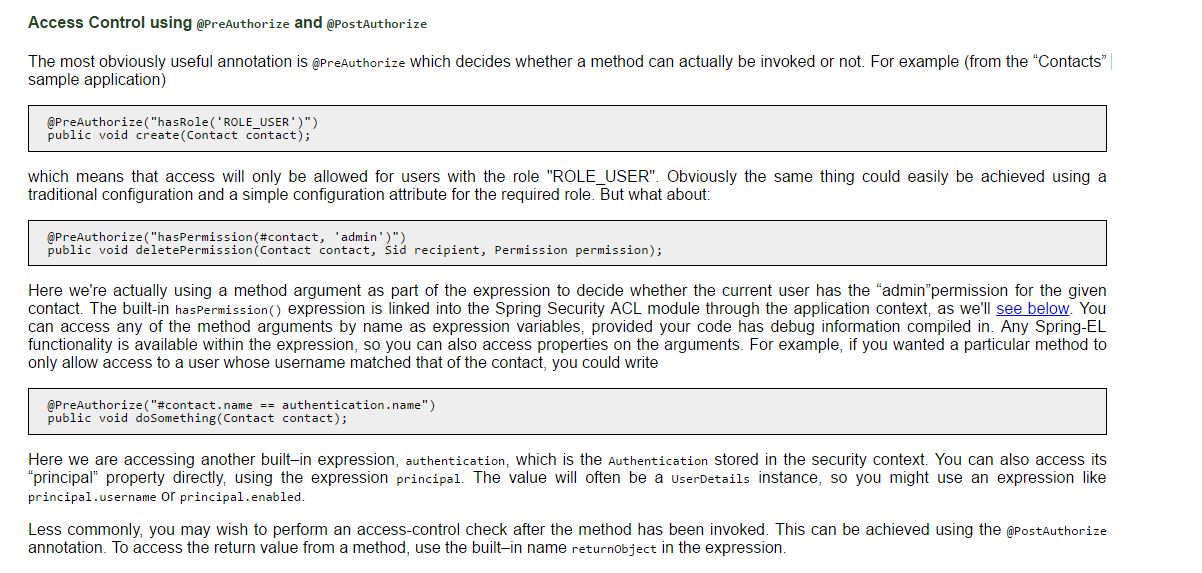


15.3.1 @Pre and @Post Annotations

There are four annotations which support expression attributes to allow pre and post-invocation authorization checks and also to support filtering of submitted collection arguments or return values. They are @PreAuthorize, @PreFilter, @PostAuthorize and @PostFilter.

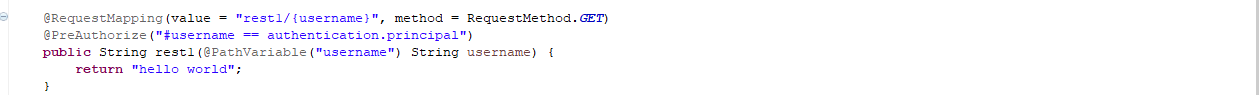
Enable them



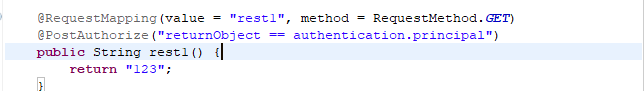


Example:

Pre Authorize

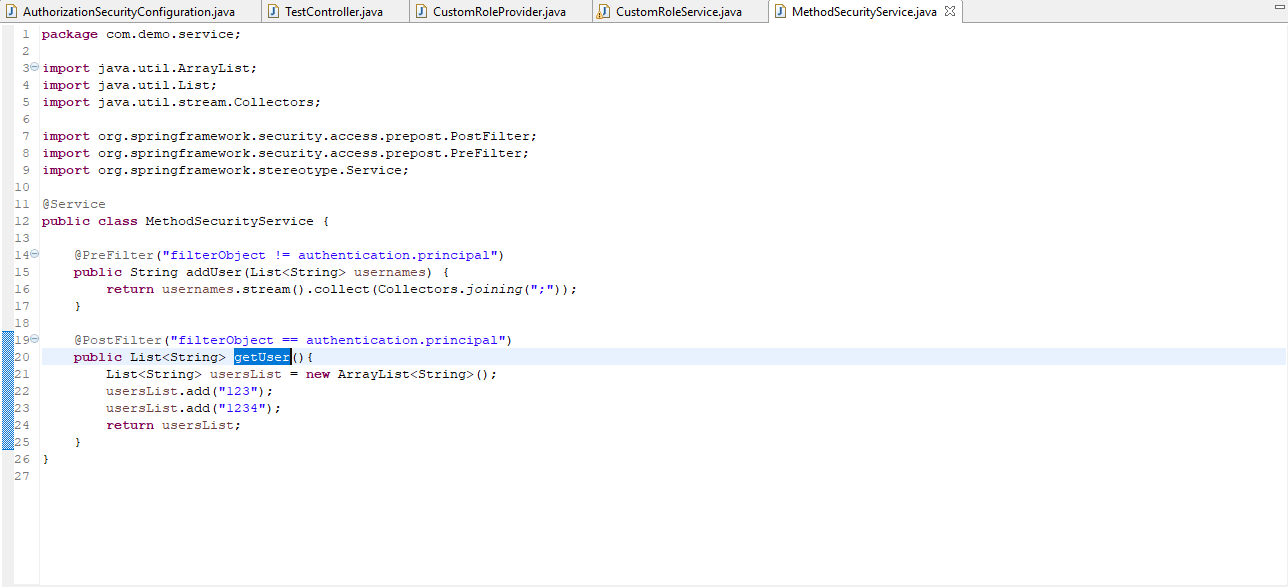


Post Authorize

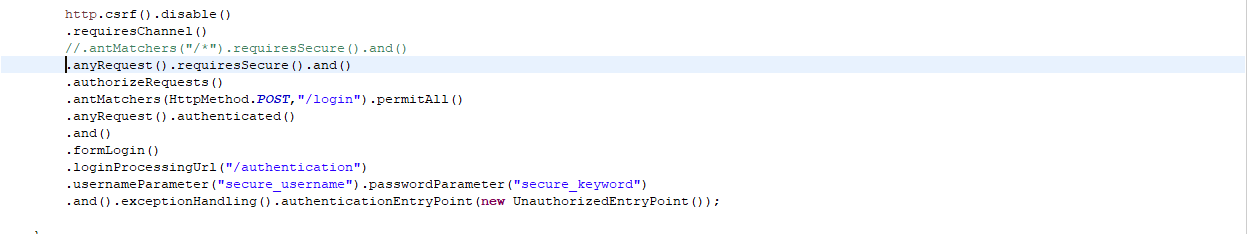


Using @PreFilter and @PostFilter Annotations

Spring Security provides the @PreFilter annotation to filter a collection argument before executing the method:



**Configure https only**



**Web Security**

configure(WebSecurity) is used for configuration settings that impact global security (ignore resources, set debug mode, reject requests by implementing a custom firewall definition). For example, the following method would cause any request that starts with /resources/ to be ignored for authentication purposes.



**Success and Failure Handler**



