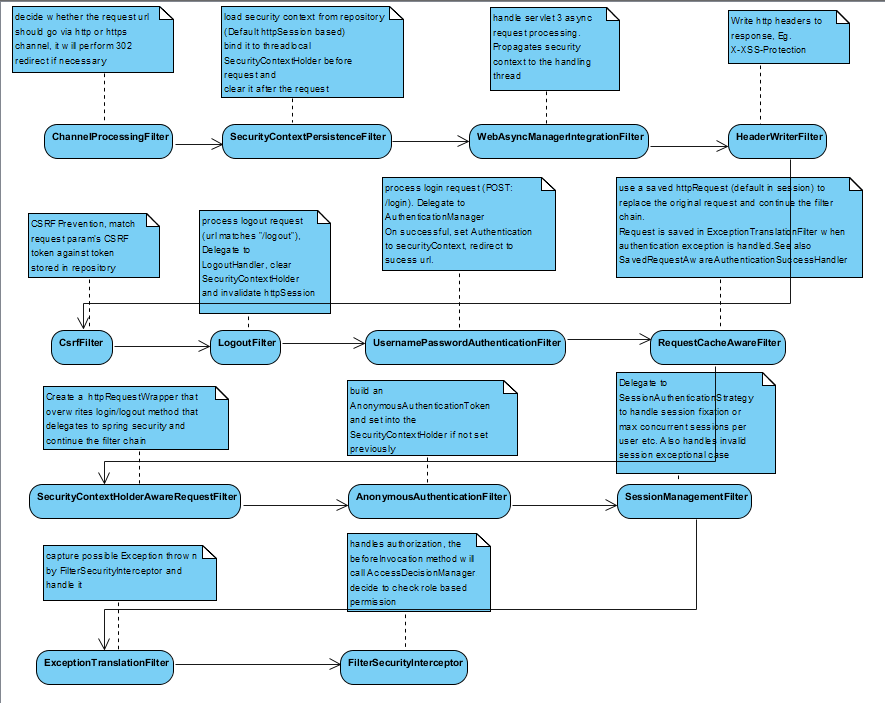
## Spring security filter chain:



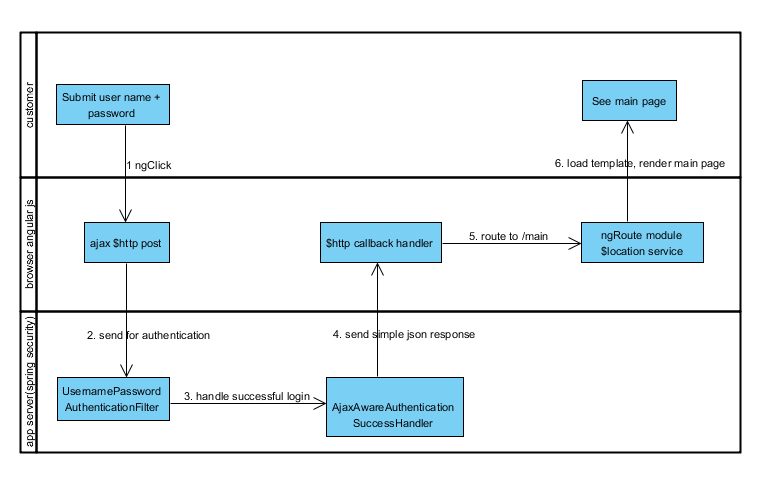
## Basic authentication and authorization with angular js application:

Key issue: default spring security uses redirection to handle various security exceptions (such as wrong user name/password). For ajax request, it is hard for the browser to follow the redirection, since redirection will only happen within the ajax response.

Client JS should supply a special http header for the server to differentiate that the request is sent from ajax. Eg. X-Requested-With: XMLHttpRequest

For angular js implementation, a global http interceptor is used to interpret the http status code. If it sees a special http status that indicate exceptional security condition for ajax request, it will broadcast an event for angular js listener to handle. The listener will extract url from even message and then follow the redirect.

Note, if you use angularjs route module, it is best to put the event listener for security exception event as a global one (registered in **anguleModule.run** method), this is to avoid redefine listener in each controller



* Supplying wrong user name/password

UsernamePasswordAuthenticationFilter.attemptAuthentication throws an AuthenticationException(BadCredentialsException) and delegate to AuthenticationFailureHandler.

Therefore, we need a special *AjaxAwareUrlAuthFailureHandler* that extends *SimpleUrlAuthenticationFailureHandler*. If the ajax http header exists, the handler will call **response.sendError** that uses a special error code (901) and sends a special json format error page to browser ajax callback, otherwise, it will perform redirect as per normal

*<security:form-login login-page="/html/login.html"*

*authentication-success-handler-ref="ajaxAwareUrlAuthSuccessHandler"*

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

*<bean id="ajaxAwareUrlAuthSuccessHandler" class="org.wsipersd.core.security.web.AjaxAwareAuthenticationSuccessHandler">*

*</bean>*

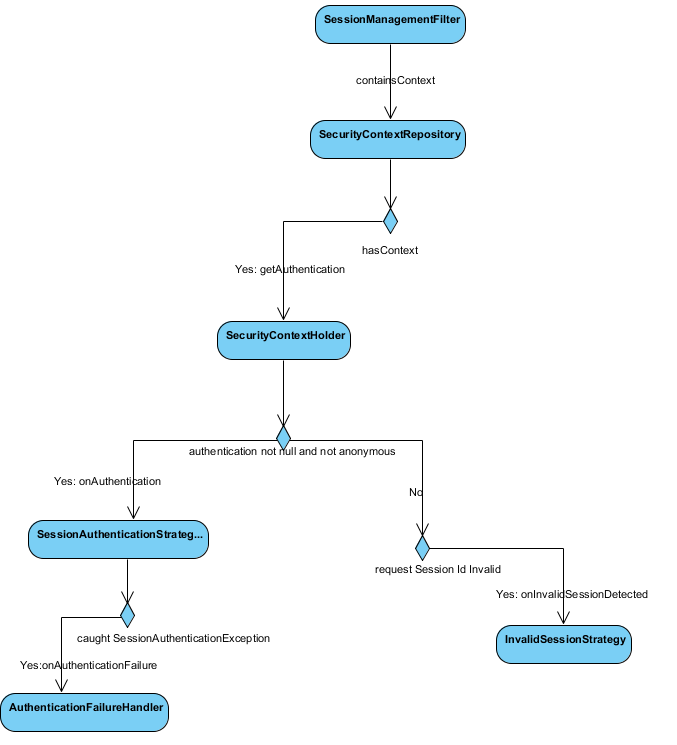
*<bean id="ajaxAwareUrlAuthFailureHandler" class="org.wsipersd.core.security.web.AjaxAwareUrlAuthFailureHandler">*

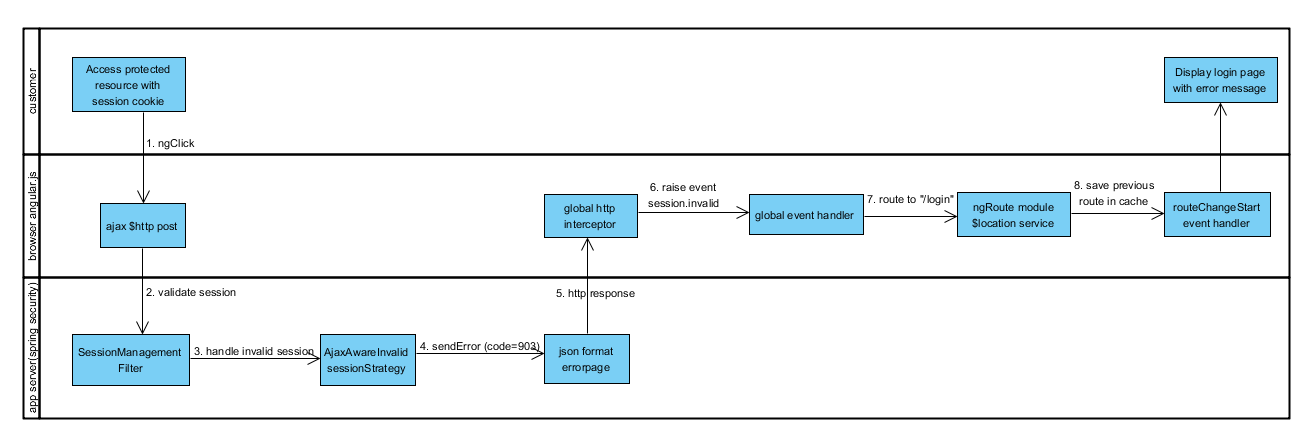
*<constructor-arg index="0" type="java.lang.String" value="/html/login.html"></constructor-arg>*

*<property name="ajaxResponseErrorCode" value="901"/>*

*</bean>*

* Session timeout





SessionManagementFilter will see that there is no authentication context in repository and request and request sessionId is not valid. It will delegate to InvalidSessionStrategy.

Therefore, we need a special AjaxAwareInvalidSessionStrategy. The logic is exactly the same as *AjaxAwareUrlAuthFailureHandler*, only that a different error code (903) is used.

*<security:session-management session-fixation-protection="migrateSession" invalid-session-strategy-ref="ajaxAwareInvalidSessionStrategy" />*

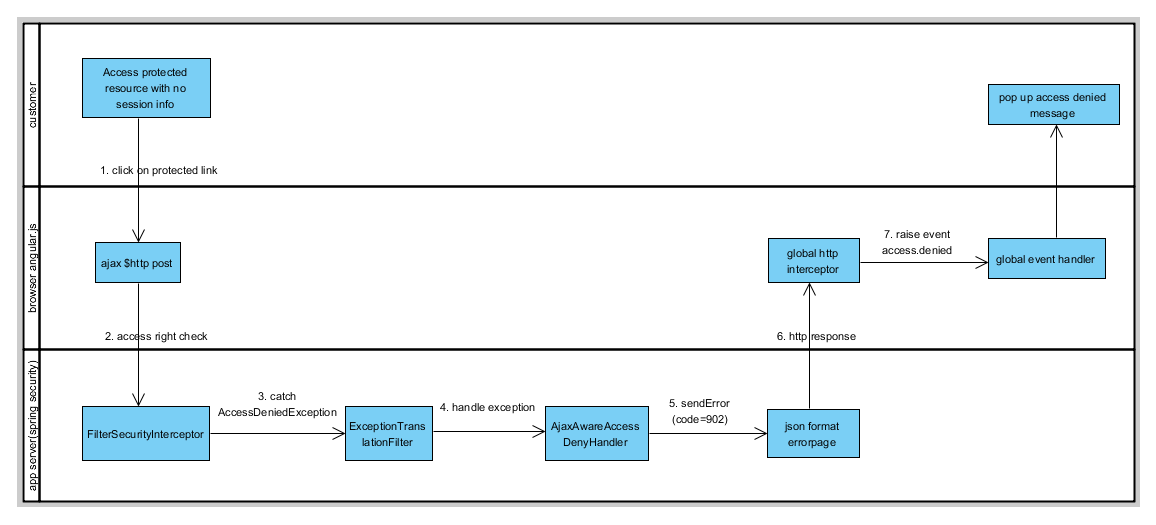
*<bean id="ajaxAwareInvalidSessionStrategy" class="org.wsipersd.core.security.web.AjaxAwareInvalidSessionStrategy">*

*<property name="destinationUrl" value="/html/login.html"/>*

*<property name="ajaxResponseErrorCode" value="903"/>*

*</bean>*

* Trying to access resources without permission



AccessDeniedException is thrown by FilterSecurityInterceptor and caught by ExceptionTranslationFilter. The ExceptionTranslationFilter delegates to AccessDeniedHandler that registered in spring configuration file. A special AjaxAwareAccessDenyHandler is used. The logic is exactly the same as *AjaxAwareUrlAuthFailureHandler*, only that a different error code (902) is used and the redirect url is an access denied page instead of the login page.

*<security:access-denied-handler ref="ajaxAwareAccessDenyHandler"/>*

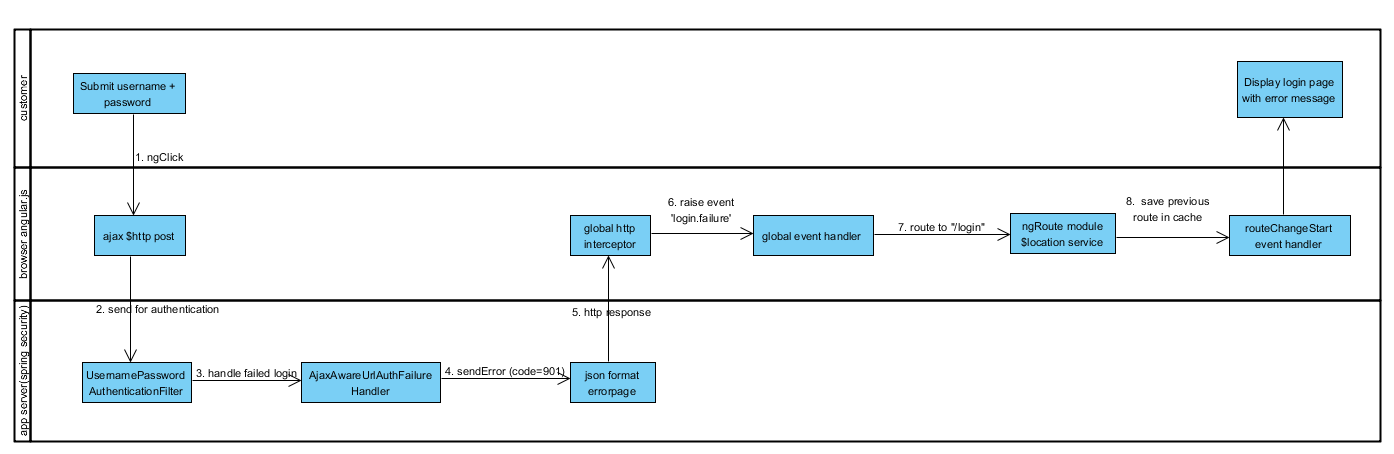
*<bean id="ajaxAwareAccessDenyHandler" class="org.wsipersd.core.security.web.AjaxAwareAccessDenyHandler">*

*<property name="errorPage" value="/static/accessDenied.html"/>*

*<property name="ajaxResponseErrorCode" value="902"/>*

*</bean>*

* Trying to access resources without log in



There are again two possible cases here:

Access a protected page: For angularjs routing module, you can gain access to a page by using “#’ notion. Eg. <https://localhost:8443/springsec/html/template.html#!/main>. Similar to the third case, AccessDeniedException is thrown by FilterSecurityInterceptor and caught by ExceptionTranslationFilter. However, handleSpringSecurityException method checks that if the authentication is anonymous, it will save the current request url and redirect to AuthenticationEntryPoint. (sendStartAuthentication method). The default AuthenticationEntryPoint is configured in <security:form-login **login-page="/html/login.html”**. On subsequent successful login, SavedRequestAwareAuthenticationSuccessHandler will retrieve the previously saved url from request cache and forward. For angular js, this does not work well, because the url server sees is not really the route path. It is better to let angular js itself remember the path and redirect on successful login.

Access a protected rest resource via ajax: On server side, the processing should be the same as the previous case. On client side, however, there is no previous route to remember by angular js, so it simply redirects to login page, and after login, the default main page is displayed.

So here, the following changes are implemented:

AjaxAwareAuthenticationEntryPoint replaces the default LoginUrlAuthenticationEntryPoint. For ajax request, it sends a special error code (901) to browser just like an unsuccessful login request.

*<security:http* ***entry-point-ref="ajaxAwareAuthenticationEntryPoint"****>*

*<bean id="ajaxAwareAuthenticationEntryPoint" class="org.wsipersd.core.security.web.AjaxAwareAuthenticationEntryPoint">*

*<constructor-arg index="0" type="java.lang.String" value="/html/login.html"></constructor-arg>*

*<property name="ajaxResponseErrorCode" value="901"/>*

*</bean>*

AjaxAwareHttpSessionRequestCache replaces the default HttpSessionRequestCache. It does not save request if it is an ajax one.

*<security:request-cache ref="ajaxAwareHttpSessionRequestCache"/>*

*<bean id="ajaxAwareHttpSessionRequestCache" class="org.wsipersd.core.security.web.AjaxAwareHttpSessionRequestCache">*

*</bean>*

On angularjs side, if it is a login error, we use $cacheProvider to remember previous routing path in routeChangeStart event handler and load the path from cache and use $location.path to redirect on successful login.

## Things to note:

After spring security 4, CSRF protection is enabled by default, you have to turn it off by setting

<csrf disabled=”true”/>

By default, all css/js request also goes through the security chain, this is undesirable from performance perspective

<security:http pattern="/js/\*\*" security="none"/>

Creates an empty security chain for js files

From spring security 4:

UsernamePasswordAuthenticationFilter’s post url has been changed from “/j\_spring\_security\_check” to “/login”. Username and password parameter has been changed from “j\_username” and “j\_password” to simply “username” and “password”

For tomcat 8, the session is persisted/restored after server shuts down. You can change the behavior by setting in context.xml

<Manager pathname="" />

See <https://tomcat.apache.org/tomcat-8.0-doc/config/manager.html#Persistence_Across_Restarts>