Configuració GICAR/CORS amb Apache en aplicacions REST

A qui va dirigit

Aquest how-to va dirigit a tots aquells usuaris que utilitzin Apache amb Spring Security a una aplicació Canigó 3.1 REST.

Versió de Canigó

Els passos descrits en aquest document apliquen a la darrera versió del Framework Canigó 3.1.x.

Introducció

En aquest howto s'explica com tenir configurat el servei de seguretat (en aquest howto s'utilitza GICAR com a provider) en una aplicació REST amb el filtre CORS per a utilitzar un Apache com a darrere proxy invers.

Configuració GICAR/CORS amb Apache en aplicacions REST

Servei de seguretat

Hem d'afegir les dependències a canigo-security a la nostra aplicació:

pom.xml

```
…
<canigo.security>[1.1.0,1.2.0)</canigo.security>
…
   <dependency>
        <groupId>cat.gencat.ctti</groupId>
        <artifactId>canigo.security</artifactId>
        <version>${canigo.security}</version>
        </exclusions>
     </dependency>
…
```

Al fitxer web.xml hem d'afegir els filtres que utilitza el servei de seguretat

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>
<filter>
       <filter-name>UrlRewriteFilter</filter-name>
       <filter-
class>cat.gencat.ctti.canigo.arch.web.core.filters.urlrewrite.UrlRewriteFilter<
/filter-class>
       <init-param>
              <param-name>confPath</param-name>
              <param-value>classpath:urlrewrite/urlrewrite.xml</param-value>
       </init-param>
</filter>
<filter-mapping>
       <filter-name>UrlRewriteFilter</filter-name>
       <url-pattern>/*</url-pattern>
</filter-mapping>
…
```

Per últim hem de configurar el servei de seguretat a Spring. Crearem un fitxer anomenat app-custom-security.xml a la ruta src/main/resources/spring

Configuració GICAR/CORS amb Apache en aplicacions REST

app-custom-security.xml

```
…
<!--  Secure patterns -->
<security:http use-expressions="true">
	<security:intercept-url pattern="/**" access="permitAll"
method="OPTIONS" />
    <security:intercept-url pattern="/**" access="hasRole('ROLE_ADMIN')" />

	<security:form-login login-processing-url="/j_spring_security_check"
login-page="/j_spring_security_check" />
	<security:custom-filter ref="proxyUsernamePasswordAuthenticationFilter"
before="FORM_LOGIN_FILTER" />
</security:http>

<security:authentication-manager alias="authenticationManager">
	<security:authentication-provider ref="gicarProvider"/>
</security:authentication-manager>

<bean id="proxyUsernamePasswordAuthenticationFilter"
class="cat.gencat.ctti.canigo.arch.security.authentication.ProxyUsernamePasswor
dAuthenticationFilter">
	<property name="siteminderAuthentication" value="true" />
	<property name="authenticationManager" ref="authenticationManager" />
	<property name="authenticationFailureHandler" ref="failureHandler" />
</bean>

<bean id="failureHandler"
class="org.springframework.security.web.authentication.SimpleUrlAuthenticationF
ailureHandler">
	<property name="defaultFailureUrl" value="/gicar-error.html" />
</bean>

<bean id="gicarProvider"
class="cat.gencat.ctti.canigo.arch.security.provider.siteminder.SiteminderAuthe
nticationProvider">
	<description>GICAR Provider</description>
	<property name="userDetailsService" ref="gicarUserDetailsService"/>
</bean>

<bean id="gicarUserDetailsService"
class="cat.gencat.ctti.canigo.arch.security.provider.gicar.GICARUserDetailsServ
iceImpl">
	<description>User Detail service implementation for GICAR
provider</description>
	<property name="httpGicarHeaderUsernameKey"
value="${security.gicar.httpGicarHeaderUsernameKey:NIF}"/>
	<property name="authoritiesDAO" ref="authoritiesDAO"/>
</bean>

<bean id="authoritiesDAO"
class="cat.gencat.ctti.canigo.arch.security.provider.sace.authorities.Authoriti
esDAOImpl">
	<description>
	Authorities DAO implementation for SACE. Gets granted authorities for
specified user
	</description>
	<property name="dataSource" ref="dataSource"/>
</bean>
…
```

Configuració GICAR/CORS amb Apache en aplicacions REST

Per a evitar que el servei de seguretat intercepti les operacions OPTIONS s'afegeix la línia:

```
<security:intercept-url pattern="/**" access="permitAll" method="OPTIONS" />
```

A l'exemple interceptem totes les altres peticions amb:

```
<security:intercept-url pattern="/**" access="hasRole('ROLE_ADMIN')" />
```

Amb la propietat method es pot especificar si només es vol interceptar les operacions PUT, GET, etc...

Configuració GICAR/CORS amb Apache en aplicacions REST

Cors Filter

Per a configurar el CORS Filter a la nostra aplicació l'afegim al web.xml

web.xml

```
…
<filter>
       <filter-name>CorsFilter</filter-name>
       <filter-class>cat.gencat.serveisrest.filters.CorsFilter</filter-class>
</filter>
<filter-mapping>
  <filter-name>CorsFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
…
```

En aquest exemple hem creat la clase cat.gencat.serveisrest.filters.CorsFilter. Aquí s'ha de indicar el path del vostre CorsFilter.

En aquesta classe s'ha d'especificar el domini de la vostra aplicació:
response.addHeader("Access-Control-Allow-Origin", domini);

Un exemple de com quedaria el fitxer:

Configuració GICAR/CORS amb Apache en aplicacions REST

cat.gencat.serveisrest.filters.CorsFilter

```
package cat.gencat.serveisrest.filters;

import java.io.IOException;

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;


public class CorsFilter implements Filter {


	public void init(final FilterConfig filterConfig) throws
ServletException {
	}

	private void doFilter(final HttpServletRequest request,
			final HttpServletResponse response, final FilterChain
chain)
			throws IOException, ServletException {

		response.addHeader("Access-Control-Allow-Origin", domini);
		response.addHeader("Access-Control-Allow-Headers",
				"origin, content-type, accept, authorization");
		response.addHeader("Access-Control-Allow-Credentials", "true");
		response.addHeader("Access-Control-Allow-Methods",
				"GET, POST, PUT, DELETE, OPTIONS, HEAD");
		response.addHeader("Access-Control-Max-Age", "1209600");

		chain.doFilter(request, response);
	}

	public void doFilter(final ServletRequest request,
			final ServletResponse response, final FilterChain chain)
			throws IOException, ServletException {

		if (request instanceof HttpServletRequest
				&& response instanceof HttpServletResponse) {
			doFilter((HttpServletRequest) request,
					(HttpServletResponse) response, chain);
		} else {
			throw new ServletException(
					"Cannot filter non-HTTP requests/responses");
		}
	}
	public void destroy() {
	}

}
```

Alternativa - Cors Filter

Una configuració del Cors Filter alternativa i que ens proporcionaria la opció de tenir configurat el domini de la nostra aplicació segons l'entorn és el següent:

Traiem la configuració del Cors Filter del web.xml i implementem la configuració al fitxer app-custom-security.xml:

src/main/resources/spring/app-custom-security.xml

```
…
<bean id="corsFilter" class="cat.gencat.serveisrest.filters.CorsFilter" >
        <property name="domini" value="${domini}"/>
</bean>

<!--  Secure patterns -->
<security:http use-expressions="true">
        …
        <security:custom-filter ref="corsFilter" before="HEADERS_FILTER"/>
</security:http>
…
```

I al fitxer CorsFilter el modifiquem per afegir la propietat **domini**

```
…
private String domini;

public String getDomini() {
        return domini;
}
public void setDomini(String domini) {
        this.domini = domini;
}

private void doFilter(final HttpServletRequest request,
        final HttpServletResponse response, final FilterChain chain)
                throws IOException, ServletException {

        response.addHeader("Access-Control-Allow-Origin", getDomini());
        response.addHeader("Access-Control-Allow-Headers",
                        "origin, content-type, accept, authorization");
        response.addHeader("Access-Control-Allow-Credentials", "true");
        response.addHeader("Access-Control-Allow-Methods",
                        "GET, POST, PUT, DELETE, OPTIONS, HEAD");
        response.addHeader("Access-Control-Max-Age", "1209600");
        chain.doFilter(request, response);
}
…
```

Configuració GICAR/CORS amb Apache en aplicacions REST

Apache

Configuració Virtual Host

```
<VirtualHost *:80>
        ServerName <domini>
        ServerAdmin <domini>

        DocumentRoot "<path_contingut_estatic>"
        <Directory />
            Options FollowSymLinks
            AllowOverride None
        </Directory>
        <Directory <path_contingut_estatic>>
            DirectoryIndex index.html
            Options Indexes FollowSymLinks MultiViews
            AllowOverride None
            Require all granted
            Allow from all
        </Directory>

        ##### Configuració <Location> #####

</VirtualHost>
```

On la configuració de la "Location" ha de ser la següent depenent del mòdul d'Apache utilitzat:

mod_proxy

```
ProxyPass /<app>/<context_contingut_dinamic>
http://<ip_servidor_aplicacions>:<port_connector_http_servidor_aplicacions>/<app>
/<context_contingut_dinamic>
ProxyPassReverse /<app>/<context_contingut_dinamic>
http://<ip_servidor_aplicacions>:<port_connector_http_servidor_aplicacions>/<app>
/<context_contingut_dinamic>
```

mod_proxy_ajp

```
ProxyPass /<app>/<context_contingut_dinamic>
ajp://<ip_servidor_aplicacions>:<port_connector_ajp_servidor_aplicacions>/<app>/<
context_contingut_dinamic>
ProxyPassReverse /<app>/<context_contingut_dinamic>
ajp://<ip_servidor_aplicacions>:<port_connector_ajp_servidor_aplicacions>/<app>/<
context contingut dinamic>
```

mod_wl_24

```
<Location /<app>/<context_contingut_dinamic>>
        WLSRequest On
</Location>
```

I en la configuració del mòdul

```
<IfModule mod_weblogic.c>
        WebLogicHost <ip_servidor_aplicacions>
        WebLogicPort <port_servidor_aplicacions>
</IfModule>
```

Configuració GICAR/CORS amb Apache en aplicacions REST

L'aplicació en el servidor d'aplicacions s'ha de desplegar amb context-root "/<app>/<context_contingut_dinamic>".
Típicament el <context_contingut_dinamic> es "AppJava".