# A qui va dirigit

Aquest how-to va dirigit als perfils tècnics (desenvolupadors i arquitectes) que hagin de desenvolupar aplicacions Canigó 3 basades en arquitectura REST+HTML5/JS.

# Versió de Canigó

Els passos descrits en aquest document apliquen a la versió del framework Canigó 3.

# Introducció

En aquest howto s’explica com configurar el servei de seguretat (en aquest howto s’utilitza GICAR com a provider) de Canigó 3 en una aplicació REST+HTML5/JS desplegada darrere un Apache configurat com a proxy invers. També s’inclou la configuració del filtre CORS.

# 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 el filtre de Spring Security:

**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>

…

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

**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.ProxyUsernamePasswordAuthenticationFilter"*>

<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.SimpleUrlAuthenticationFailureHandler"*>

<property name=*"defaultFailureUrl"* value=*"/gicar-error.html"* />

</bean>

<bean id=*"gicarProvider"* class=*"cat.gencat.ctti.canigo.arch.security.provider.siteminder.SiteminderAuthenticationProvider"*>

<description>GICAR Provider</description>

<property name=*"userDetailsService"* ref=*"gicarUserDetailsService"*/>

</bean>

<bean id=*"gicarUserDetailsService"* class=*"cat.gencat.ctti.canigo.arch.security.provider.gicar.GICARUserDetailsServiceImpl"*>

<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.AuthoritiesDAOImpl"*>

<description>

Authorities DAO implementation for SACE. Gets granted authorities for specified user

</description>

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

</bean>

…

Per a evitar que el servei de seguretat no autentiqui el mètode HTTP “OPTIONS”, en el cas d’AngularJS s’utilitza en les peticions de preflight, s’afegeix la línia:

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

I interceptem totes les altres peticions per autoritzar el seu accés:

<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...

# 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 (aquest filtre es proporcionarà en properes versions de Canigó).

En el filtre s’ha d’especificar el domini de l’aplicació:

*response.addHeader("Access-Control-Allow-Origin", <domini>);*

Un exemple de com quedaria el filtre:

**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() {

}

}

# Configuració CORS Filter multi-entorn

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 la següent:

Treiem 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 Apache per proxy invers

A continuació s’especifiquen les diferents configuracions d’Apache per a funcionar com a proxy invers que es pot trobar als CPDs corporatius

<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>

Configuració Virtual Host:

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 (Weblogic 12c i Apache 2.4.x)**

<Location /<app>/<context\_contingut\_dinamic>>

WLSRequest On

</Location>

I en la configuració del mòdul d’Apache de WL:

<IfModule mod\_weblogic.c>

WebLogicHost <ip\_servidor\_aplicacions>

WebLogicPort <port\_servidor\_aplicacions>

</IfModule>

És molt important en tots els casos que l'aplicació en el servidor d'aplicacions es desplegui amb context-root "/<app>/<context\_contingut\_dinamic>". Típicament el <context\_contingut\_dinamic> es configura amb valor "AppJava".