# A qui va dirigit

Aquest how-to va dirigit als perfils tècnics (desenvolupadors i arquitectes) que desenvolupin aplicacions Canigó i vulguin utilitzar websockets per a tenir un canal de comunicació servidor/client sense haver d’obrir nous ports de comunicació.

# Versió de Canigó

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

# Introducció

En aquest HowTo s’explica com configurar un websocket a servidor i realitzar la connexió des de client.

A l’exemple s’escriu al socket cada 10 segons, i es pinta a una plana html.

La integració de la tecnologia Websockets a l’aplicació es realitza amb “spring-websockets” per la part del servidor i “SockJS per la part del client.

Els passos descrits han estat realitzats a partir d’una aplicació amb arquitectura REST+HTML5/JS generada amb el plugin de Canigó per a Eclipse.

# Configuració a servidor (Spring websockets)

Primer de tot s’han d’afegir les llibreries necessàries al “pom.xml”, per aquest howto són necessàries *spring-websocket* i *spring-messaging.*

Al **pom.xml** afegim les dependències:

*…*

*<dependency>*

*<groupId>org.springframework</groupId>*

*<artifactId>spring-websocket</artifactId>*

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

*</dependency>*

*<dependency>*

*<groupId>org.springframework</groupId>*

*<artifactId>spring-messaging</artifactId>*

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

*</dependency>*

*…*

Per a configurar el websocket amb Spring hi ha dos possibilitats, mitjançant anotacions o amb un fitxer xml.

Configuració amb anotacions

**WebSocketConfig.java**

***package*** *cat.gencat.websockets.config;*

***import*** *org.springframework.context.annotation.Configuration;*

***import*** *org.springframework.messaging.simp.config.MessageBrokerRegistry;*

***import*** *org.springframework.web.socket.config.annotation.AbstractWebSocketMessageBrokerConfigurer;*

***import*** *org.springframework.web.socket.config.annotation.EnableWebSocketMessageBroker;*

***import*** *org.springframework.web.socket.config.annotation.StompEndpointRegistry;*

*@Configuration*

*@EnableWebSocketMessageBroker*

***public******class*** *WebSocketConfig* ***extends*** *AbstractWebSocketMessageBrokerConfigurer {*

*@Override*

***public******void*** *registerStompEndpoints(StompEndpointRegistry registry) {*

*registry.addEndpoint("/watch").withSockJS();*

*}*

*@Override*

***public******void*** *configureMessageBroker(MessageBrokerRegistry registry) {*

*registry.enableSimpleBroker("/topic");*

*registry.setApplicationDestinationPrefixes("/app");*

*}*

}

El fitxer “WebSocketConfig.java” s’ha de ficar en un path on Spring escanegi les anotacions, en aquest cas caldrà afegir al nostre *app-custom-beans.xml* la següent configuració:

***<context:component-scan base-package="cat.gencat.websockets.config" />***

Configuració amb fitxer xml

Dintre de la carpeta resources/spring s’ha de crear el fitxer “websockets.xml” (o el nom que es desitgi).

**websockets.xml**

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

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

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

*xsi:schemaLocation="*

*http://www.springframework.org/schema/beans*

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

*http://www.springframework.org/schema/websocket*

*http://www.springframework.org/schema/websocket/spring-websocket-4.0.xsd">*

*<websocket:message-broker application-destination-prefix="/app">*

*<websocket:stomp-endpoint path="/watch" >*

*<websocket:sockjs/>*

*</websocket:stomp-endpoint>*

*<websocket:simple-broker prefix="/topic"/>*

*</websocket:message-broker>*

*</beans>*

Amb aquesta configuració s’obre un websocket al path /watch i un broker a /topic.

Els missatges s’escriuran a /topic/XXX on XXX és el nom que es desitgi. Amb aquesta configuració de websockets es pot escriure en diversos tòpics (Ex. /topic/XXX, /topic/YYY, /topic/ZZZ, ...).

El client s’ha de connectar per escoltar al path /watch i subscriure’s al tòpic /topic/XXX que es desitgi escoltar.

Escriure el missatge

Per mostrar el funcionament del websocket en aquest howto hem creat un servei que escriu al websocket la hora cada 10 segons.

**WebsocketService.java**

***package*** *cat.gencat.websockets.service;*

***import*** *java.util.Date;*

***import*** *java.util.concurrent.Executors;*

***import*** *java.util.concurrent.ScheduledExecutorService;*

***import*** *java.util.concurrent.ScheduledFuture;*

***import*** *java.util.concurrent.TimeUnit;*

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

***import*** *org.springframework.messaging.simp.SimpMessagingTemplate;*

***import*** *org.springframework.stereotype.Service;*

*@Service("websocketService")*

***public******class*** *WebsocketService {*

*@Autowired*

***private*** *SimpMessagingTemplate template;*

***final*** *Runnable writeMessage =* ***new*** *Runnable() {*

***public******void*** *run() {*

*template.convertAndSend("/topic/howto", "" +* ***new*** *Date());*

*}*

*};*

***final*** *ScheduledExecutorService scheduler = Executors.newScheduledThreadPool(1);*

***final*** *ScheduledFuture<?> writeHandle = scheduler.scheduleAtFixedRate(writeMessage, 10, 10, TimeUnit.****SECONDS****);*

*}*

En aquest exemple s’escriu al tòpic /topic/howto

# Configuració a client (SockJS)

Per a realitzar el client utilitzem SockJS + Stomp i JQuery.

Al nostre html inclourem les dependències a aquestes llibreries, dos botons (un per connectar-se i un altre per desconnectar-se) i un div on s’aniran escrivint els missatges tant bon punt arribin.

**Index.html**

*<!DOCTYPE html>*

*<html xmlns:ng="http://angularjs.org" lang="en">*

*<head>*

*<meta http-equiv="content-type" content="text/html; charset=UTF-8">*

*<script src="https://ajax.googleapis.com/ajax/libs/jquery/2.2.2/jquery.min.js"></script>*

*<script src="https://cdnjs.cloudflare.com/ajax/libs/sockjs-client/1.0.3/sockjs.js"></script>*

*<script src="https://cdnjs.cloudflare.com/ajax/libs/stomp.js/2.3.3/stomp.js"></script>*

*<script src="js/main.js"></script>*

*</head>*

*<body id="ng-app" >*

*<div class="container">*

*<div class="navbar">*

*<div class="navbar-inner">*

*<div class="container">*

*<a class="brand" href="index.html">*

*<img src="http://www.gencat.cat/img2/temp/escudo\_tcm32-28377.gif"></a>*

*</div>*

*</div>*

*</div>*

*</div>*

*<input type="button" id="connectar" value="Connectar" >*

*<input type="button" id="desconnectar" value="Desconnectar" >*

*<div id="websocketContent"></div>*

*</body>*

*</html>*

Amb javascript activem la connexió quan es premi al botó *connectar*, quan arribi el missatge l’afegirem al div websocketContent i si volem deixar d’escoltar el websocket es pot prémer el botó *desconnectar*

**js/main.js**

*$(document).ready(****function****() {*

***var*** *socket;*

*$("#connectar").click(****function****() {*

*socket =* ***new*** *SockJS('http://localhost:8081/websockets/api/watch');*

*stompClient = Stomp.over(socket);*

*stompClient.connect({},* ***function****(frame) {*

*stompClient.subscribe('/topic/howto',* ***function****(content){*

*addMessage(content.body);*

*});*

*});*

*});*

*$("#desconnectar").click(****function****() {*

*socket.close();*

*});*

***function*** *addMessage(message) {*

*$websocketContent = $('#websocketContent');*

***var*** *autoscroll = ($websocketContent.scrollTop() + $websocketContent.innerHeight()) === $websocketContent[0].scrollHeight;*

***var*** *$contentDiv = $('<div></div>').html(message);*

*$websocketContent.append($contentDiv);*

***if*** *(autoscroll) {*

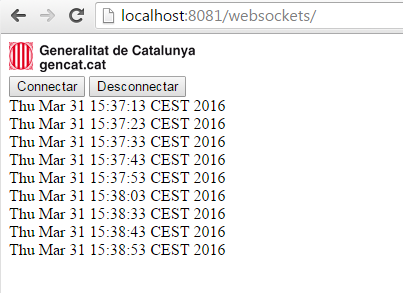
*$websocketContent.scrollTop($websocketContent[0].scrollHeight);*

*}*

*}*

*});*

# Resultat



Després del missatge a les 15:37:53 hem desconnectat, per tornar a connectar i continuar escrivint el missatge rebut