# A qui va dirigit

Aquest how-to va dirigit als perfils tècnics (desenvolupadors i arquitectes) que vulguin desenvolupar una aplicació Canigó sense frontend.

# Versió de Canigó

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

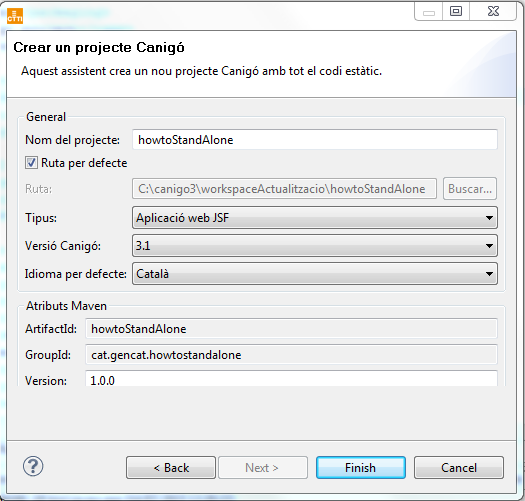
# Introducció

Canigó està preparat per al desenvolupament d’aplicacions webs, però potser per necessitat és té el requeriment de crear un backend sense interfície d’usuari, ja sigui web o d’escriptori. Per exemple, un backend que requereixi integracions amb sistemes Gencat com la PICA, Antivirus, etc, i/o un planificador de tasques que executi de forma periòdica certes funcionalitats. Tot i que Canigó no ha estat concebut per aquest tipus d’aplicacions, és possible fer-ho.

En aquest how-to expliquem com generar una aplicació d’aquest tipus a partir de l’aplicació base generada pel plugin de Canigó per a Eclipse. En aquesta aplicació d’exemple integrarem una tasca planificada cada 5 segons.

# Crear el Projecte

Amb el plugin d’Eclipse es crea una aplicació Canigó:



Eliminem les parts que no es necessiten:

* src/main/webapp
* src/main/java/cat/gencat/howtostandalone/bean/LogBean.java
* src/main/resources/assemble
* src/main/resources/urlrewrite
* src/main/resources/canigo.skin.properties

Al pom.xml eliminem les referències a la part web:

* canigo.web.jsf
* jaxb-impl
* jaxb-api
* spring-web
* primefaces
* javax.servlet-api

**pom.xml**

…

<properties>

<junit.version>4.11</junit.version>

<canigo.core.version>[3.1.0,3.2.0)</canigo.core.version>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<org.springframework.version>4.1.0.RELEASE</org.springframework.version>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>${junit.version}</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-test</artifactId>

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

<scope>test</scope>

</dependency>

<dependency>

<groupId>cat.gencat.ctti</groupId>

<artifactId>canigo.core</artifactId>

<version>${canigo.core.version}</version>

</dependency>

</dependencies>

…

# Scheduler

Per utilitzar el scheduler hem d’afegir al pom les següents dependències:

<!-- SERVEI DE TASQUES -->

<dependency>

<groupId>org.quartz-scheduler</groupId>

<artifactId>quartz</artifactId>

<version>2.2.1</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context-support</artifactId>

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

</dependency>

<dependency>

<groupId>org.apache.commons</groupId>

<artifactId>commons-collections4</artifactId>

<version>4.0</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-tx</artifactId>

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

</dependency>

<!-- FI SERVEI DE TASQUES -->

I a Spring hem de definir les tasques

**src/main/resources/spring/quartz-config.xml**

…

<bean id=*"SchedulerService"* class=*"cat.gencat.howtostandalone.service.SchedulerService"* />

<bean id=*"FirstTaskJobDetail"* class=*"org.springframework.scheduling.quartz.MethodInvokingJobDetailFactoryBean"*>

<property name=*"targetObject"* ref=*"SchedulerService"* />

<property name=*"targetMethod"* value=*"testScheduler"* />

</bean>

<bean id=*"FirstSimpleTrigger"* class=*"org.springframework.scheduling.quartz.SimpleTriggerFactoryBean"*>

<property name=*"jobDetail"* ref=*"FirstTaskJobDetail"* />

<property name=*"repeatInterval"* value=*"5000"* />

<property name=*"startDelay"* value=*"1000"* />

</bean>

<bean class=*"org.springframework.scheduling.quartz.SchedulerFactoryBean"*>

<property name=*"jobDetails"*>

<list>

<ref bean=*"FirstTaskJobDetail"* />

</list>

</property>

<property name=*"triggers"*>

<list>

<ref bean=*"FirstSimpleTrigger"* />

</list>

</property>

</bean>

…

Al fitxer quartz-config.xml es crea una tasca que s’executarà cada 5 segons. La tasca que s’executarà és crea al fitxer SchedulerService

**src/main/java/cat/gencat/howtostandalone/service/SchedulerService.java**

**package** cat.gencat.howtostandalone.service;

**import** java.util.Date;

**import** org.apache.commons.logging.Log;

**import** org.apache.commons.logging.LogFactory;

**public** **class** SchedulerService {

**private** **static** **final** Log *log* = LogFactory.*getLog*(SchedulerService.**class**);

**public** **void** testScheduler(){

System.*out*.println((**new** Date()) + " Executat metode de test de scheduler");

}

}

En aquest exemple la tasca escriurà per pantalla cada cop que s’executi.

# Main

Per a poder executar l’aplicació és necessari una classe Main. En aquesta classe carregarem el canigo. core i el servei de planificació de tasques que hem creat:

**src/main/java/cat/gencat/howtostandalone/Main.java**

**package** cat.gencat.howtostandalone;

**import** org.apache.commons.logging.Log;

**import** org.apache.commons.logging.LogFactory;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**public** **class** Main {

**private** **static** **final** Log *log* = LogFactory.*getLog*(Main.**class**);

/\*\*

\* **@param** args

\*/

**public** **static** **void** main(String[] args) **throws** Exception{

**new** ClassPathXmlApplicationContext("cat/gencat/ctti/canigo/arch/core/config/canigo-core.xml");

**new** ClassPathXmlApplicationContext("spring/quartz-config.xml");

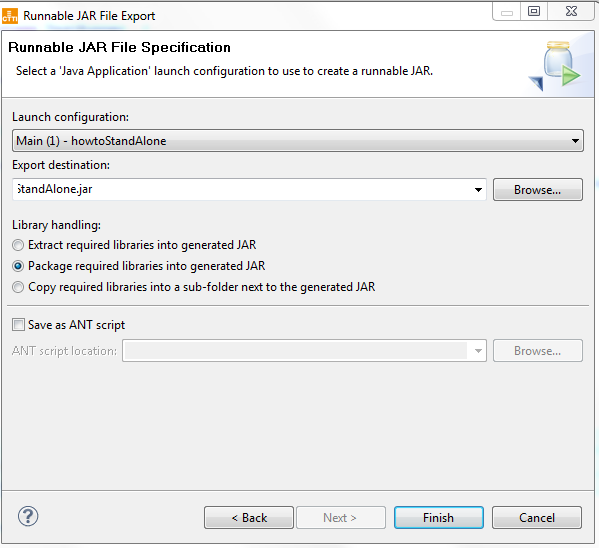
}

}

# Generar el Jar

Per a poder executar la nostra aplicació hem de generar el jar. Des d’Eclipse això es pot fer de la següent manera:

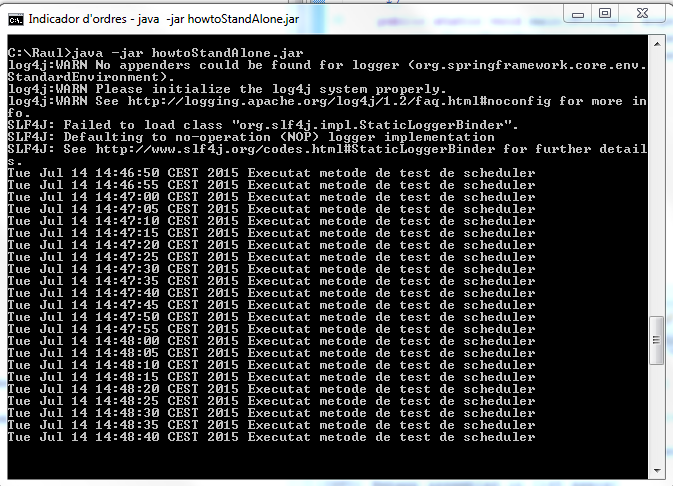
* Botó dret al projecte i prémer Exportar
* Selecionar Java/Runnable Jar File
* Indicar la classe que s’executarà, el nom del fitxer de destí i com volem deixar les llibreries



# Resultat

Per a executar el backend Canigó només cal fer la següent crida:

**java –jar howtoStandAlone.jar**



Com es pot observar, per executar aquest tipus de backend, no cal cap servidor d’aplicacions ni contenidor de servlets.