# Instalación

Instalar ffmpeg, software necesario para procesar las imágenes de los avatars.

Crear la base de datos con los scripts correspondientes (mysql o sqlserver).

Crear los wars con el build provisto.

Copiar el war al directorio de deploy de la configuración.

Copiar el datasource correspondiente al directorio de deploy de la configuracion.

### Generacion de los wars

1) Ejecutar el build.xml

Esto compilara los fuentes y generara los archivos mysql/LOJACK\_WEB.war y sqlserver/LOJACK\_WEB.war

En los pasos de creación de base de datos, usuario y copiado de datasources cambiar los datos de la base y usuario de creerlo necesario.

### Crear la base de datos y el usuario:

1) Mysql: mysql/create-mysql.sql

2) Sqlserver: sqlserver/db-sqlserver.sql

### Crear los objetos con el usuario:

1) Mysql: mysql/create-mysql.sql

2) Sqlserver: sqlserver/create-sqlserver.sql

### Copiar la definición de datasource correspondiente al directorio de deploy:

1) Mysql: mysql/mysql-ds.xml

2) sqlserver: sqlserver/sqlserver-ds.xml

### Copiar el jar del driver al directorio lib de la configuración de jboss:

1) Mysql: mysql/mysql-connector-java-5.1.13-bin.jar

2) sqlserver: sqlserver/sqljdbc4.jar

## Configuración

Toda la configuración de la aplicación se realiza a través de datos de la tabla SYSPROPERTIES.

#### A continuación se listan las propiedades y su uso:

**proxy.http:** Configuracion del proxy a usar para requerimientos http. Esta configuracion es global, luego puede activar o desactivarse para cada servicio. Debe configurarse como server:port, ejemplo, localhost:9090.

**proxy.https:** Configuracion del proxy a usar para requerimientos https. Esta configuracion es global, luego puede activar o desactivarse para cada servicio. Debe configurarse como server:port, ejemplo, localhost:9090.

**proxy.socks:** Configuracion del proxy a usar para requerimientos socks. Debe configurarse como server:port, ejemplo, localhost:9090.

**front.server:** Se usa para generar links a la aplicación que son absolutos, debe tener el nombre con el cual el server que hostea el front se ve desde otros equipos, por ejemplo: www.lojackhome.com

**thalamus.server:** url del server de thalamus, por ejemplo http://localhost:8280/ThalamusWeb/ o https://testa-lojack-rest.thalamuslive.com/lojack

**thalamus.touchpoint.code:** código asignado para thalamus (REAL-LIFE-SMARTPHONE-APP)

**thalamus.touchpoint.token:** token asignado para thalamus

(k8exyct1v6edf9q50fryuq3r02upn0m98twst4etlbjgc9cnsq585879gxlcbu11)

**thalamus.timeout:** timeout en milisegundos para la comunicacion con Thalamus

**thalamus.proxy:** Activa o desactiva el uso de proxy en la comunicacion con Thalamus. Si la url de thalamus es https usara la property de proxy.https, caso contrario usara proxy.http. Este valor debe ser true o false.

**guid:** Token para la comunicación con el middleware

**gis.server:** Ubicación del server de GIS, por ejemplo http://localhost:8180/GISWeb/gis/

**services.server:** Ubicación del server de servicios no GIS, por lo hablado debe ser el mismo.

**services.timeout:** Timeout para la comunicación con el middleware.

**middleware.proxy:** Activa o desactiva el uso de proxy en la comunicacion con el middleware. Si la url del middleware es https usara la property de proxy.https, caso contrario usara proxy.http. Este valor debe ser true o false.

**mw.protocol:** Valor json para emulador de middleware,soap-json para Carpathia.

**prevent.server:** Ubicación del server de prevent, por ejemplo <http://www.lojackgis.com.ar/PreventWCFServices/GIS>

**prevent.timeout:** Timeout para la comunicación con prevent

**prevent.proxy:** Activa o desactiva el uso de proxy en la comunicacion con el prevent. Si la url de prevent es https usara la property de proxy.https, caso contrario usara proxy.http. Este valor debe ser true o false.

**prevent.loginurl:** Url para hacer el login automatico a prevent

**prevent.token:** Token de seguridad de prevent

**prevent.loginurl :** Url para hacer el login automatico a pets

**prevent.mobile.loginurl :** Url para hacer el login automatico a pets mobile

**pets.token:** Token de seguridad de prevent

**prop.tmp.path:** Directorio temporal, aquí se extraerá la configuración del log4j y se escribirá el log, así como los blobs serán extraídos en dicho directorio.

**camera.proxy:** Activa o desactiva el uso de proxy en la comunicacion con las camaras.

**camera.mobile.refreshTime:** Tiempo de refresco de la versión Mobile de la cámara, representa la cantidad de milisegundos de refresco entre imagen e imagen.

**camera.mobile.mode:** Determina el acceso a la imagen en modo mobile, los valores posibles son local, proxy, external, socket.

**camera.mobile.external.url:** Url del servicio externo de bajada de imagen.

**camera.mobile.socket:** Server y puerto de la aplicación stand alone que funciona como intermediario para la camara. Por ejemplo, localhost:8800.

**camera.applet.refreshTime:** Tiempo de refresco de la versión applet de la cámara, representa la cantidad de milisegundos de refresco entre imagen e imagen.

**camera.connectTimeOut:** Timeout para la conexión con la camara, en milisegundos

**camera.readTimeOut: Timeout para la lectura de la camara, en milisegundos**

**camera.cache:** Tiempo durante el cual el front cachea las imagenes y devuelve la misma sin acceder a la camara, en milisegundos

**camera.cacheSize:** Tamaño maximo del cache de imagenes, se cachea una imagen por url de camara

Como recomendación, camera.mobile.refreshTime debe ser mayor a camera.connectTimeOut y camera.readTimeOut.

**job.refresh.time:** Tiempo de refresco del proceso del front que consulta los estados de los jobs asincrónicos. Se expresa en milisegundos.

**job.abort.time:** Tiempo pasado el cual se asume que un job no terminara. Pasado ese tiempo no se consultara más dicho job. Se expresa en milisegundos.

**job.client.refresh.time:** Tiempo de refresco del ajax del cliente que consulta los estados de los jobs. Se expresa en milisegundos.

**front.login.deplay:** Milisegundos de retraso del login.

**contactform.email:** Dirección de email a la cual se enviaran los formularios de contacto

**video.car:** URL de youtube del video de car

**video.home:** URL de youtube del video de home

**video.pets:** URL de youtube del video de pets

**video.loapp:** URL de youtube del video de loapp

**video.mobile.car:** URL de youtube del video de car para mobile

**video.mobile.home:** URL de youtube del video de home para mobile

**video.mobile.pets:** URL de youtube del video de pets para mobile

## Mails

La configuración para envío de emails se encuentra en la tabla SYSPROPERTIES con el prefijo mail.

Se provee una configuración ejemplo que utiliza gmail para el envío (cuenta test.lojack.front@gmail.com/t3stl0j4k)

## Logging

La configuración del log de la aplicación se realiza a través de log4j. La aplicación extrae automáticamente el log4j al directorio prop.tmp.path, luego puede ser editado para configurarlo.

Si bien cada clase tiene su log, existen tres clases que pueden ser particularmente importantes y son las que acceden a servicios externos:

**com.tdil.thalamus.client.core.**

**ThalamusClient:** log de la clase que maneja la comunicación con thalamus. En modo debug muestra todas las invocaciones, incluyendo parámetros y resultados. En modo info muestra los tiempos de las invocaciones.

**com.tdil.lojack.prevent.PreventConnector:** log de la clase que maneja la comunicación con prevent. En modo debug muestra todas las invocaciones, incluyendo parámetros y resultados. En modo info muestra los tiempos de las invocaciones.

**com.tdil.lojack.gis.LoJackServicesConnector:** log de la clase que maneja la comunicación con el middleware. En modo debug muestra todas las invocaciones, incluyendo parámetros y resultados. En modo info muestra los tiempos de las invocaciones. Se recomienda quitar el uso de console appender en el ambiente productivo.

# CameraServer

Camera server es la aplicación java stand alone que funciona de intermediario entre la web app y las cámaras.

Esta aplicación abre un socket al cual la web app se conecta y ejecuta los comandos, ya sea de pedido de imagen como de movimiento. Cada comando ejecuta en un thread independiente en camera server.

La aplicación se exporta al directorio camera-server. En el mismo se encuentra el jar con la logica, un archivo de properties ejemplo, un archivo de logging ejemplo y un sh también de ejemplo para levantarla.

A continuación se lista los parametros configurables:

server.port= Puerto en el cual escucha requerimientos

camera.connectTimeOut= Timeout de conexión con la camara

camera.readTimeOut= Timeout de lectura con la camara

camera.proxy.https= Proxy para accesos https localhost:9092

camera.proxy.http=Proxy para accesos http localhost:9092

Estos últimos dos parametros pueden dejarse vacíos en cuyo caso no utiliza proxy. Algo importante es que el proxy debe ser SOCKs.

En cuanto al logging, si se deja la categoría com.tdil.lojack.camera.CameraServer en debug loguea los pedidos incluyendo los parámetros, si esta en info loguea solo los pedidos, en error solo los errores.

No se provee la funcionalidad de bajar la app, se deberá ejecutar un kill al proceso.

# Administracion

La web app expone casos de uso de administrarla desde el website.

El acceso a la misma es traves de la pagina /admin/login.jsp

Desde esta pagina, dado un usuario autorizado, se puede acceder a ver, modificar y recargar las sys properties asi como a la configuracion del sistema de logging y bajado del log completo.

Los usuarios de estas funcionalidades de administracion residen en la tabla SYSTEMUSER.

Dicha tabla debe ser mantenida por fuera del sistema.

Cada registro tiene el username, el password (este campos es el sha1 del password), y si tiene acceso a la configuracion de logs y/o de system properties (loggingAccess 0 o 1, syspropAccess 0 o 1).

Pantallas







