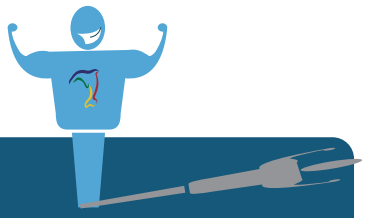
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

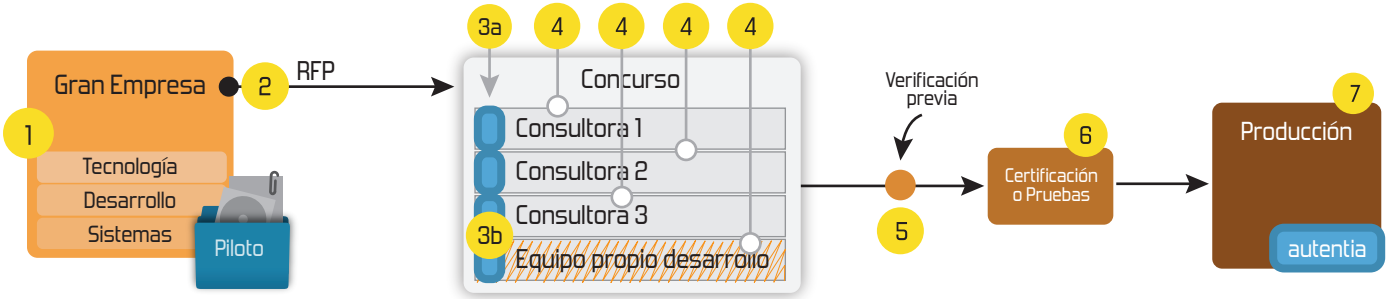
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

Spring MVC, JSF-PrimeFaces /RichFaces, HTML5, CSS3, JavaScript-jQuery

Gestor portales (Liferay)
Gestor de contenidos (Alfresco)
Aplicaciones híbridas

Control de autenticación y acceso (Spring Security)
UDDI
Web Services
Rest Services
Social SSO
SSO (Cas)

JPA-Hibernate, MyBatis
Motor de búsqueda empresarial (Solr)
ETL (Talend)

Dirección de Proyectos Informáticos.
Metodologías ágiles
Patrones de diseño
TDD

Tareas programadas (Quartz)
Gestor documental (Alfresco)
Inversión de control (Spring)

BPM (jBPM o Bonita)
Generación de informes (JasperReport)
ESB (Open ESB)

Compartimos nuestro conocimiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

DESARROLLADO POR:
Francisco Javier Martínez Páez

Consultor tecnológico de desarrollo de proyectos informáticos.

Ingeniero Técnico en Telecomunicaciones

Puedes encontrarme en Autentia: Ofrecemos servicios de soporte a desarrollo, factoría y formación

Somos expertos en Java/J2EE

Ver tutoriales de Francisco Javier Martínez Páez

Fecha de publicación del tutorial: 2011-12-14

83

Apache Camel, primeros pasos

Lo primero, os dejo el enlace a los: fuentes de este tutorial

Introducción

Si hay algo manido, tratado, documentado y reinventado en el mundo de la informática es todo lo relacionado con la integración de aplicaciones. Si le pegas una patada a una piedra elegida aleatoriamente, probablemente saldrán un par de libros que hablan acerca de este tema. Por eso no me voy a extender demasiado en explicaciones teóricas, y voy a tratar de pasar a la acción lo antes posible.
He tenido la fortuna de elegir "Apache Camel" para un proyecto en el que estado involucrado en estos últimos meses, y la verdad es que ha resultado todo un éxito su elección, y como suelo decir siempre (sin dejar de sorprenderme), completamente "gratis".
Como agradecimiento a los desarrolladores de este framework (y a todos aquellos que hacen software libre) escribo este tutorial (por sirve de algo).
También agradezco a Carlos León que en una charla de café me dijese: "i Anda !, para eso que quieres hacer a lo mejor te viene bien Camel (tenía razón)."

¿ Qué es Apache Camel ?

Los que saben explicarlo bien, son los que lo han construido: Web de Apache Camel. No obstante, me voy a atrever a explicar en pocas palabras lo que es y sus características principales:

- Framework Java que implementa los principales Patrones EIP (Enterprise Integration Patterns)
- Muy sencillo de comprender y de utilizar (si conoces Spring, mucho mejor), y muy versátil
- Incorpora un gran soporte para pruebas automáticas (lo que es de agradecer en este tipo de frameworks)
- Contiene una gran cantidad de componentes "de serie" y una gran cantidad de "conectores" o "transport" (en dialecto camel)

- Puede ser un componente muy útil dentro de una arquitectura SOA (como parte del ESB), como mediador entre aplicaciones, o enrutador o ... . Algunos ESBs lo incluyen como parte de su core (como por ejemplo ServiceMix)

Instalando Apache Camel

Error!! No hay nada que instalar. No es una aplicación, sino que es un framework que se integra en cualquier aplicación (web, standalone ...).

Planteando el problema

Para tratar de entender cuales son las capacidades de Apache Camel, os voy a proponer un problema que trataremos de resolver usando este framework.
Resulta que estamos trabajando para una empresa que vende cacharricos electrónicos por Internet usando para ello una pasarela de pago contra "Acme". Cada vez que se realiza un pago, "Acme" envía un mensaje de confirmación del pago o una notificación de error, si se ha producido un error en el proceso.
Nuestro objetivo, es recoger estos mensajes y procesarlos de la siguiente manera:

- Si el mensaje es una notificación de pago correcto, nuestro objetivo será hacerle llegar el pago al departamento financiero, así como registrar la venta.
- Si el mensaje es una notificación de pago incorrecto, nuestro objetivo será notificar el error al departamento de atención al cliente, y enviar un correo al usuario indicándole el problema que se ha producido.

Al lío.

Vamos a suponer que las notificaciones de ACME nos vienen a través de un POST http, enviando únicamente dos datos: El identificador del usuario, el identificador de la transacción y el resultado de la operación (recordad que es únicamente un ejemplo).
Vamos a crear una aplicación Web (estoy usando Tomcat 7) donde embeberemos toda esta funcionalidad (usando camel), y una jsp que nos servirá para enviar notificaciones de venta.
También vamos a usar colas jms como mecanismo de comunicación. Yo tengo instalado Apache ActiveMQ 5.5.0. Si no queréis usar un gestor de colas, podéis usar el componente SEDA de Camel (son colas en memoria. Bueno, es algo más que eso, pero para el caso es suficiente).
Lo primero es mostraros el pom.xml (únicamente lo interesante...):

```
    ...
    <!-- Camel y módulos -->
        <dependency>
            <groupId>org.apache.camel</groupId>
            <artifactId>camel-core</artifactId>
```

El web.xml:

```
<!DOCTYPE web-app PUBLIC
 "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd" >

<web-app>
```

El jsp (index.jsp):

```
<html>
<body>
<h2>Notifications Generator</h2>

<form method="post" action="<%=request.getContextPath()%>/camel/acme">
```

Y lo realmente importante, el fichero de configuración de camel (si se parece sospechosamente a un fichero de spring, y es porque es un fichero de configuración de Spring, aunque esta no es la única manera de configurar camel). Mostramos lo más interesante unicamente:

```
    ...
    <!-- Datasource para registrar los Pagos (No me uses en producción)-->
    <bean id="dataSource"
        class="org.springframework.jdbc.datasource.DriverManagerDataSource">
        <property name="driverClassName" value="com.mysql.jdbc.Driver" />
```

Básicamente, lo que acabamos de crear es nuestra primera "ruta" (route). La verdad es que se explica por si sola, pero por si acaso:

- from: ¿ Donde escucha esta ruta ? Pues hemos usado el componente servlet de camel. Básicamente, cada vez que llegue una petición HTTP al servlet, comenzará la ruta.
- to: Lo primero que hacemos con los mensajes que llegan, es convertirlos a XML. Para esa conversión estamos usando un "transformer" que se basa en plantillas de velocity. Este transformador, recogerá los parámetros del POST y conformará un mensaje XML. Esta es la plantilla de velocity que estamos usando:

```
<?xml version="1.0" encoding="UTF-8"?>
<sale>
    <result>${headers.result}</result>
    <user>${headers.user}</user>
    <transaction>${headers.transaction}</transaction>
```

- to: A continuación, estamos usando un "Content Based Router" usando XPath, es decir, en función de la información del mensaje lo enviaremos a un sitio o a otro. En este caso, si el mensaje es de un pago correcto (result=OK), entonces, enviaremos el mensaje a la cola de ActiveMQ "SALES.OK". Si el mensaje es de un pago incorrecto, enviaremos el mensaje a la

Técnicas básicas con Mybatis

CSS3 Media Queries o cómo hacer un diseño adaptativo según el terminal

Últimos Tutoriales del Autor

Patrón Intérprete

MongoDB, primeros pasos

Spring 3 Java Config Style

Apache Cassandra, ¿Qué es esto que tanto ruido hace?

JEE6, haciéndolo fácil.

Síguenos a través de:

Últimas ofertas de empleo

2011-09-08
Comercial - Ventas - MADRID.

2011-09-03
Comercial - Ventas - VALENCIA.

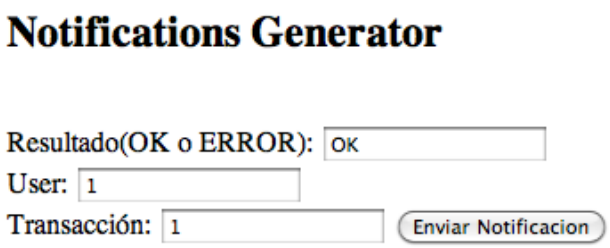
2011-08-19
Comercial - Compras - ALICANTE.

2011-07-12
Otras Sin catalogar - MADRID.

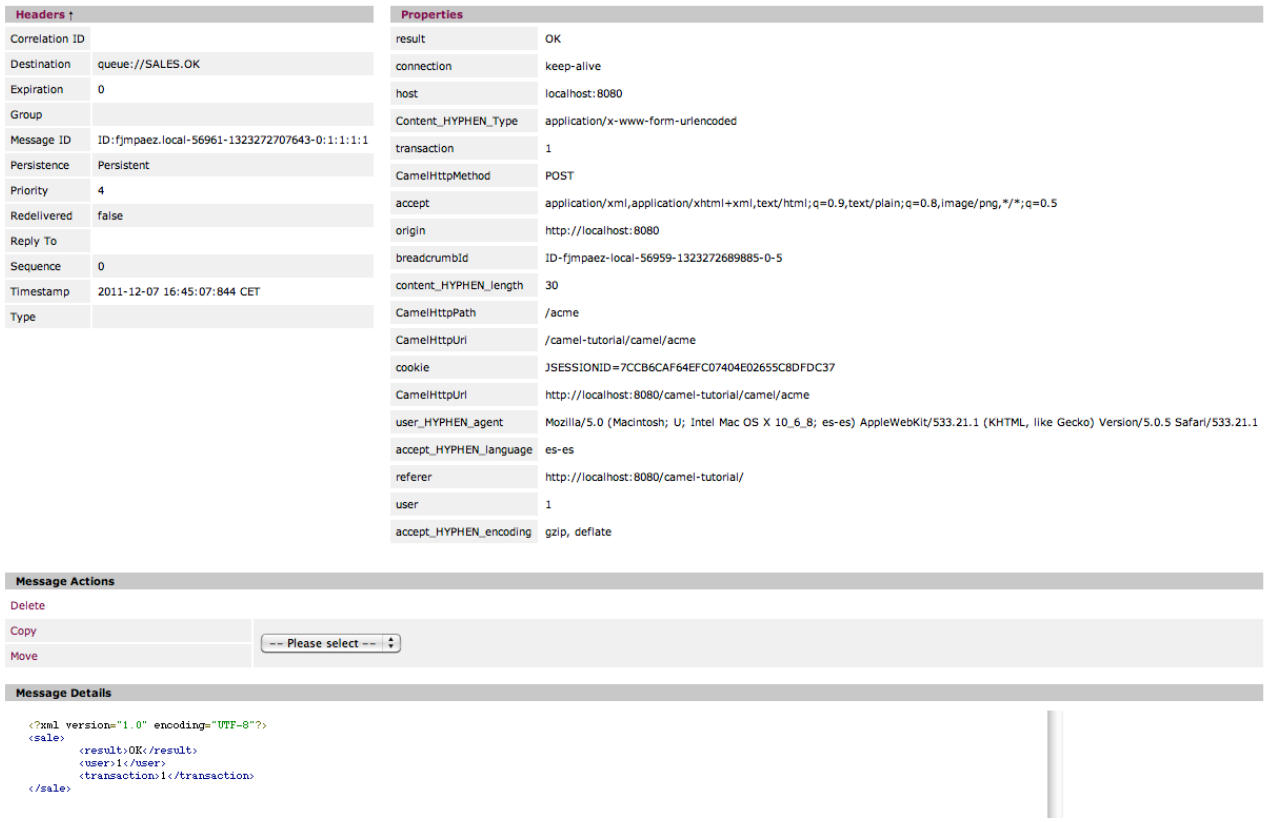
2011-07-06
Otras Sin catalogar - LUGO.

cola de ActiveMQ "SALES.ERROR". En el caso de que llegue un mensaje sin "result", entonces, escribiremos un fichero en la ruta indicada (tutorial/unknown). InOnly significa que no esperamos respuesta.

Vamos a comprobar que esto funciona.
Arrancamos la aplicación y nos vamos al index.jsp para enviar notificaciones de prueba:



Comprobamos la cola SALES.OK usando la consola de Apache ActiveMQ y vemos que efectivamente ha llegado un mensaje y además, con el formato indicado en la plantilla de velocity (mola!!):



Headers ↑	
Correlation ID	
Destination	queue://SALES.OK
Expiration	0
Group	
Message ID	ID:fjmpaez.local-56961-1323272707643-0:1:1:1:1
Persistence	Persistent
Priority	4
Redelivered	false
Reply To	
Sequence	0
Timestamp	2011-12-07 16:45:07:844 CET
Type	

Properties	
result	OK
connection	keep-alive
host	localhost:8080
Content_HYPHEN_Type	application/x-www-form-urlencoded
transaction	1
CamelHttpMethod	POST
accept	application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
origin	http://localhost:8080
breadcrumbId	ID-fjmpaez-local-56959-1323272689885-0-5
content_HYPHEN_length	30
CamelHttpPath	/acme
CamelHttpUri	/camel-tutorial/camel/acme
cookie	JSESSIONID=7CCB6CAF64EFC07404E02655C8DFDC37
CamelHttpUrl	http://localhost:8080/camel-tutorial/camel/acme
user_HYPHEN_agent	Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_6_8; es-es) AppleWebKit/533.21.1 (KHTML, like Gecko) Version/5.0.5 Safari/533.21.1
accept_HYPHEN_language	es-es
referer	http://localhost:8080/camel-tutorial/
user	1
accept_HYPHEN_encoding	gzip, deflate

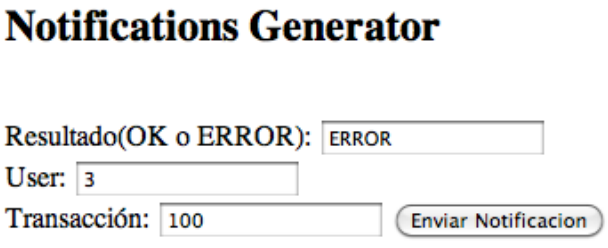
Message Actions

Delete
Copy
Move
-- Please select --

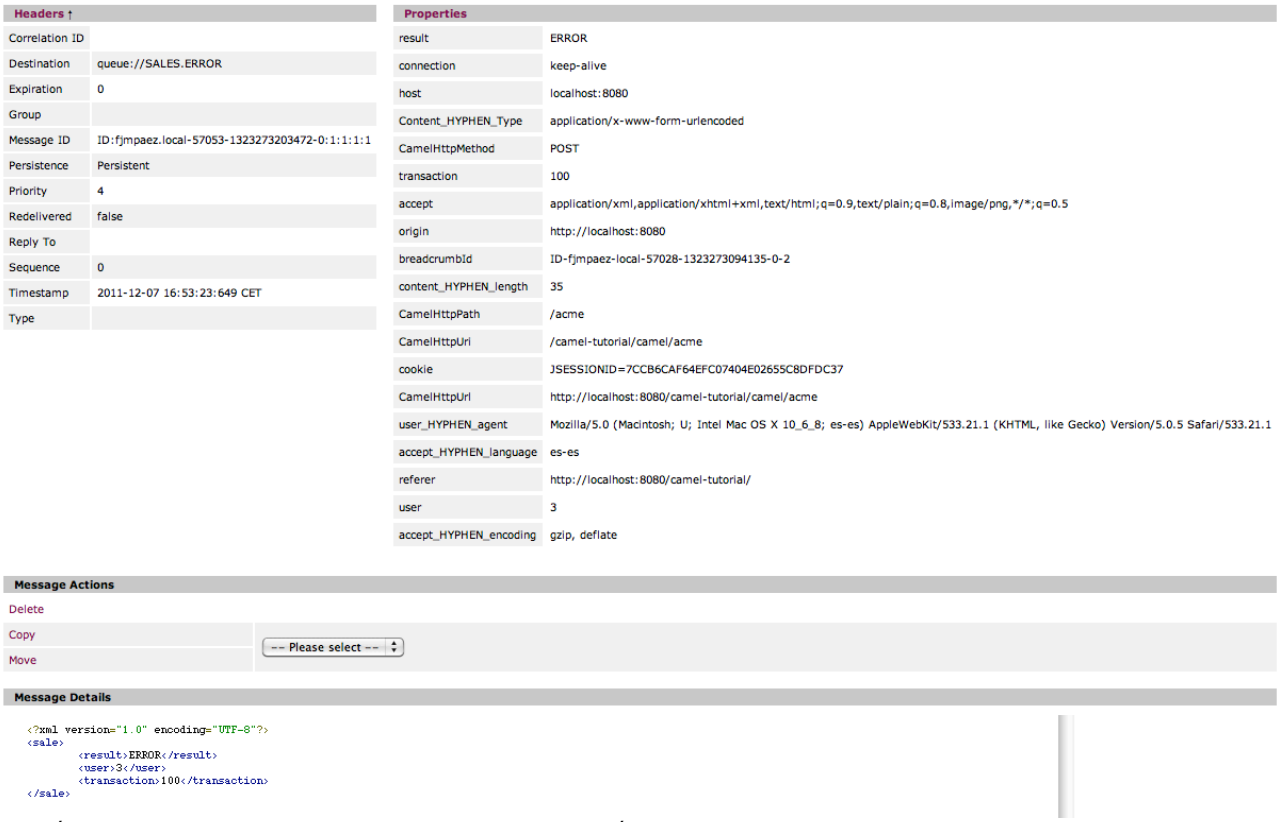
Message Details

```
<?xml version="1.0" encoding="UTF-8"?>
<sale>
        <result>OK</result>
        <user>1</user>
        <transaction>1</transaction>
</sale>
```

Probemos un mensaje de error:



Comprobamos la cola SALES.ERROR usando la consola de Apache ActiveMQ y vemos que efectivamente ha llegado un mensaje:



Headers ↑	
Correlation ID	
Destination	queue://SALES.ERROR
Expiration	0
Group	
Message ID	ID:fjmpaez.local-57053-1323273203472-0:1:1:1:1
Persistence	Persistent
Priority	4
Redelivered	false
Reply To	
Sequence	0
Timestamp	2011-12-07 16:53:23:649 CET
Type	

Properties	
result	ERROR
connection	keep-alive
host	localhost:8080
Content_HYPHEN_Type	application/x-www-form-urlencoded
CamelHttpMethod	POST
transaction	100
accept	application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
origin	http://localhost:8080
breadcrumbId	ID-fjmpaez-local-57028-1323273094135-0-2
content_HYPHEN_length	35
CamelHttpPath	/acme
CamelHttpUri	/camel-tutorial/camel/acme
cookie	JSESSIONID=7CCB6CAF64EFC07404E02655C8DFDC37
CamelHttpUrl	http://localhost:8080/camel-tutorial/camel/acme
user_HYPHEN_agent	Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_6_8; es-es) AppleWebKit/533.21.1 (KHTML, like Gecko) Version/5.0.5 Safari/533.21.1
accept_HYPHEN_language	es-es
referer	http://localhost:8080/camel-tutorial/
user	3
accept_HYPHEN_encoding	gzip, deflate

Message Actions

Delete

Copy

Move

-- Please select --

Message Details

```
<?xml version="1.0" encoding="UTF-8"?>
<sale>
        <result>ERROR</result>
        <user>3</user>
        <transaction>100</transaction>
</sale>
```

Si probáis un mensaje con result diferente a OK o ERROR, veréis que se escribe un fichero en la ruta especificada.

Bueno, no está mal por ahora. Sin pagar un duro y sin "tirar" ni una línea de código, ya tenemos hecho la mitad de lo que nos han pedido.
Sigamos ahora con nuestro negocio. Empecemos por los pagos correctos. Esto es lo que nos dijeron: "Si el mensaje es una notificación de pago correcto, nuestro objetivo será hacerle llegar el pago al departamento financiero, así como registrar la venta".
Tras hablar con el departamento financiero, ellos necesitan que les notifiquemos vía correo electrónico (cutre, pero es un ejemplo). Por otro lado, registrar la venta es guardar un registro en una base de datos...
Vamos a añadir una nueva ruta para configurar lo que necesitamos.

```
...
<route id="paymentsOK">
      <from uri="activemq:queue:SALES.OK" />
      <camel:setHeader headerName="paymentUser">
            <xpath resultType="java.lang.String">/sale/user</xpath>
```

Vamos a explicar esta nueva ruta:

- From: Escucharemos los mensajes que lleguen a la cola SALES.OK
- A continuación, usando XPath, recogeremos los valores del XML que nos interesan (user y transaction) y los guardaremos como cabeceras del mensaje (paymentUser y paymentTransaction). Estas cabeceras permaneceran en el mensaje durante toda la ruta.
- Multicast: Usamos un enrutador multicast para:
    1. Almacenar en base de datos el registro de la venta. Para eso usaremos un POJO configurado con Spring (paymentRegister)
    2. Enviar un correo electrónico al departamento financiero. Pero antes usaremos velocity como plantilla del correo.

Para almacenar en Base de Datos el pago, he usado hibernate:
La entidad:

```
@Entity
public class Payment implements Serializable {

      private static final long serialVersionUID = 1L;
```

El DAO:

```
public interface PaymentDao {

      @Transactional
      public abstract void storePayment(Payment paymentToStore);
```

```
@Repository
public class PaymentDaoHibernateImpl extends HibernateDaoSupport implements PaymentDao {

      @Autowired
      public PaymentDaoHibernateImpl(SessionFactory sessionFactory) {
```

Y ahora, el componente que registra el Pago:

```
import org.apache.camel.Exchange;
import org.apache.camel.Processor;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
```

Para enviar el correo, primero usamos velocity para generarlos usando la plantilla:

```
Dear Financial Department. This is a new Payment:
User: ${headers.paymentUser}
Transaction: ${headers.paymentTransaction}
```

Si lanzamos un nuevo ejemplo, podéis comprobar como se registra el pago y se envía el correo (siempre y cuando esté todo bien configurado).

```
ysql>
ysql> select * from Payment;
----+-------------+------+
 id | transaction | user |
----+-------------+------+
  1 | 1           | paco |
----+-------------+------+
 row in set (0.00 sec)
```

El tratamiento de los pagos erróneos os lo dejo a vosotros.

Conclusiones

Espero que este tutorial os haya servido para comprender un poco que es esto de Camel y donde podría encajar. Mi experiencia personal con Camel ha sido muy buena.
En mi caso, se ha utilizado para la catalogación masiva de ficheros y metainformación asociada a esos ficheros. Los ficheros y su metainformación se recogía vía FTP. Todos los ficheros y su metainformación eran validados y cargados en distintos repositorios de información, haciendoles pasar un workflow basado en rutas de camel (y hasta ahí puedo leer).

COMENTARIOS