E.T.S. de Ingeniería Industrial, Informática y de Telecomunicación

Entorno Fake y pruebas de integraciones

Grado en Ingeniería Informática

Trabajo Fin de Grado

Kepa Choperena Conde
Federico Fariña Figueredo
Pamplona, 01/07/2016

Objeto y Desarrollo

Este trabajo fin de grado ha sido realizado durante las prácticas curriculares llevadas a cabo en la empresa Openbravo.

En este trabajo fin de grado se aborda el desarrollo de un entorno “Fake” que emula el comportamiento de un sistema completo, el sistema de la empresa BUT, de modo que sea posible desarrollar y probar los elementos necesarios para realizar un proceso de importación de esta empresa a los sistemas de Openbravo. El objetivo consiste en poder desarrollar la infraestructura necesaria para la integración de BUT en Openbravo a pesar de que Openbravo no tiene acceso a las aplicaciones de BUT. También se aborda la construcción de serie de test, tanto unitarios como de performance, que prueben el funcionamiento de un proceso de importación.

Se realiza una breve introducción que sitúe el contexto. Se analizan los requisitos del entorno Fake. Se describen los casos de uso, diagramas de clases y el diseño de los test. Se detalla la implementación y el despliegue realizado. Se explican los resultados obtenidos en las pruebas de performance. Finalmente, se mencionan las conclusiones obtenidas del TFG.

Keywords: Entorno Fake, testing, performance, Openbravo.

ÍNDICE

1. Introducción

Este proyecto se ha realizado en la empresa Openbravo durante las prácticas en empresas cursadas en el octavo semestre del Grado de Ingeniería Informática facilitadas por la Universidad Pública de Navarra y la Fundación Universidad-Sociedad. En primer lugar realizare una pequeña presentación de la empresa, así como del cliente al que esta destinado el proyecto que abarca este trabajo fin de grado. Posteriormente, se detallarán los aspectos del diseño, la implementación y el despliegue del proyecto personal realizado.

OPENBRAVO

Openbravo es un líder mundial en el ámbito del software de código abierto comercial que pretende ofrecer a los minoristas un producto que permita gestionar los continuos cambios del negocio. Cuenta con sedes en diferentes países como México, India, Francia y España además de una red oficial de empresas colaboradoras, llamado programa de *partners*, con más de 150 *partners* a nivel mundial. Los *partners* o socios se encargan de vender, implementar y desplegar soluciones basadas en los productos de Openbravo.

La compañía ofrece una plataforma de comercio que permite a los clientes tener un amplio y ágil control de sus negocios gracias a la adaptabilidad que proporciona. Existen dos tipos de Suite. La *Openbravo Commerce Suite,* destinada a los *retailers* o minoristas ágiles, y la *Openbravo Business Suite* que se destina a las empresas con un gran tamaño.

Ante la pregunta, ¿Por que Openbravo? la empresa responde con las siguientes características generales (véase figura 1):

Figura 1: Características de la plataforma

- Diseñado para ser único: es un sistema de gestión empresarial en código libre y en entorno web que cubre todas las funcionalidades que requieren las pymes con gran flexibilidad y facilidad de uso.

- Arquitectura revolucionaria: la aplicación está basada en una arquitectura revolucionaria que consigue mejorar la manera de desarrollar las aplicaciones de gestión. Combinando las metodologías MVC (Model-View-Controller) y MDD (Model-Driven Development) se facilita el proceso de implantación, el desarrollo de nuevas funcionalidades e incluso de nuevos verticales demandados por los clientes.

- Disponible sin coste: la aplicación está accesible sin ningún tipo de coste. Adicionalmente, la documentación técnica y funcional está disponible libremente.

- Altamente adaptable: Openbravo puede configurarse fácilmente según las necesidades y particularidades de cada cliente. Debido a que es un software de código libre, el usuario puede modificarlo y ajustarlo aún más al uso que le vaya a dar.

- Siempre disponible: los usuarios pueden acceder al sistema en función de sus permisos de acceso con sólo tener un navegador instalado. Su arquitectura tecnológica garantiza el máximo nivel de seguridad, con la posibilidad de alojarse en sus propias instalaciones o remotamente.

- Oportunidad de negocio: las características del producto lo hacen realmente atractivo para empresas de pequeño tamaño. Openbravo permite a este tipo de empresas empezar por algo sin tener que realizar un gran desembolso. Además la empresa ofrece el soporte mínimo necesario para tener éxito en el negocio.

ARQUITECTURA DEL SISTEMA

Actualmente la última revisión estable y que contiene la aprobación del equipo de QA (quality assurance) es la 3.0RR16Q1.1. En esta versión del ERP (Enterprise Resource Planning), la arquitectura pasa a ser de tipo RIA (Rich Internet Application).

La arquitectura RIA nos permite separar las solicitudes de datos de la interfaz de usuario, además de que la carga de datos en el cliente se realiza una única vez y solo solicita los datos necesarios para continuar con el funcionamiento. De esta forma se aligera el trafico entre el servicio y el cliente.

La arquitectura actual facilita mucho el trabajo a los desarrolladores. Por ejemplo, pueden añadir nuevas ventanas, pestañas y campos sin necesidad de parar o reconstruir la aplicación por completo. La arquitectura del ERP se puede dividir en los siguientes bloques:

- Openbravo Core: contiene la funcionalidad común, sin ningún tipo de detalle del modelo de negocio de los clientes.
- JSON Module: provee de servicios web de tipo JSON REST (Representational State Transfer). Permite la transferencia de datos entre cliente-servidor así como con servicios externos.
- Weld Module: provee la inyección de dependencias y la gestión de los componentes.
- Kernel Module: es el encargado de las tareas de infraestructura. Realiza el procesamiento de solicitudes, la gestión de eventos, manejo de cache,…
- Datasource Module: utiliza el módulo JSON para proporcionar una capa de más alto nivel para el manejo de datos.
- Smartclient Module: provee la interfaz de usuario de la librería *Smartclient*.
- Application Module: contiene la implementación de las ventanas, pestañas y campos además del código de cliente y servidor.

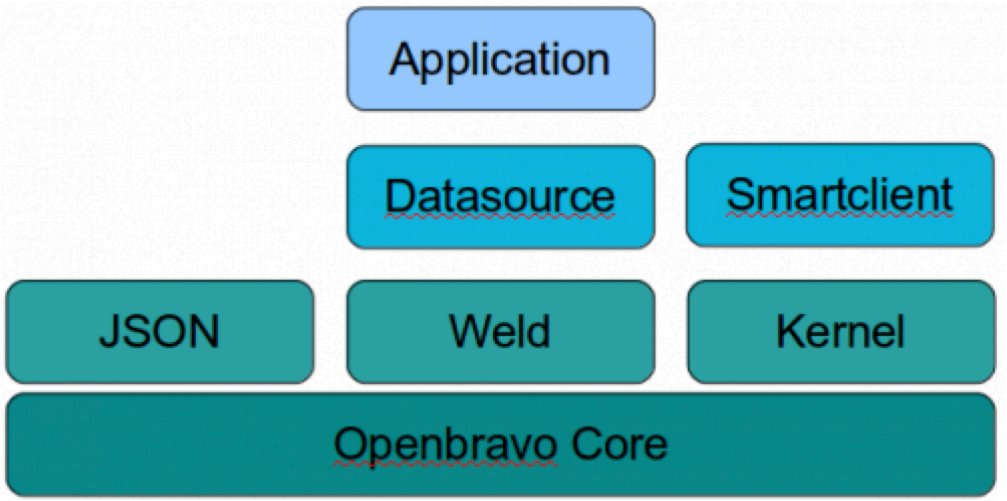

Figura 2: Arquitectura del ERP

La plataforma incluye un cliente Web POS o punto de venta que añade movilidad y agilidad al sistema. El sistema se añade extendiendo el módulo de *core* empleando el módulo Openbravo WebPOS Module, que aprovecha todos los *frameworks* de Openbravo para permitir utilizar cualquier dispositivo móvil o pantalla táctil. La visión general de la arquitectura del cliente Web POS queda de la siguiente forma.

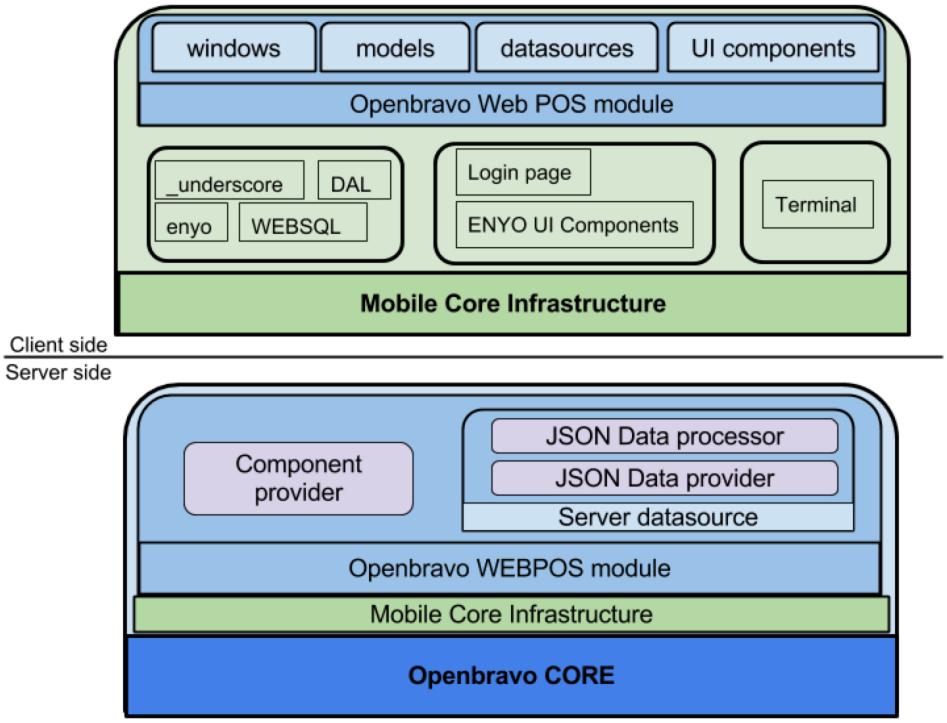

Figura 3: Arquitectura del Web POS

CLIENTE DEL PROYECTO: BUT

La empresa BUT (veasé logo en la figura 4) es una de las últimas empresas que se ha unido a Openbravo con el objetivo de mejorar su eficiencia y eficacia, además de ser uno de los clientes de mayor envergadura de Openbravo. BUT es una cadena francesa de tiendas especializadas en muebles para el hogar, electrodomésticos, imagen y sonido. En 2015 alcanzó el 13,1% de cuota del mercado de muebles detrás de Ikea y Conforama, convirtiéndolo así en una de las empresas líderes en el sector a nivel global.

Figura 4: Logo empresa BUT

Como aplicación central para la gestión, actualmente BUT utiliza una cantidad bastante alta de entornos propios que se comunican entre ellos mediante servicios web, dando como resultado una aplicación muy distribuida, con una alta complejidad de manejo, actualización y depuración.

La empresa, mediante una serie de acuerdos, ha decidido utilizar el sistema ERP de Openbravo como futura inversión dando lugar al proyecto BUT. El proyecto, pretende abarcar la integración de los sistemas que utiliza a día de hoy la compañía y cualquier tipo de desarrollo *custom* o mejora que requiera el cliente. De este modo, podrá disfrutar de un sistema centralizado, fácil de manejar, actualizar y depurar, que le permita controlar de forma más extensa los flujos funcionales creados en el negocio. Openbravo también proporcionará el soporte contratado.

Debido al tamaño de la compañía así como a la complejidad de la integración con sus sistemas, el proyecto BUT pasa a ser uno de los más costosos llevados a cabo por Openbravo, al que le asigna un tercio de la plantilla. En la siguiente imagen se puede apreciar la complejidad del mapa de sistemas actual y el coste que puede suponer la integración completa.

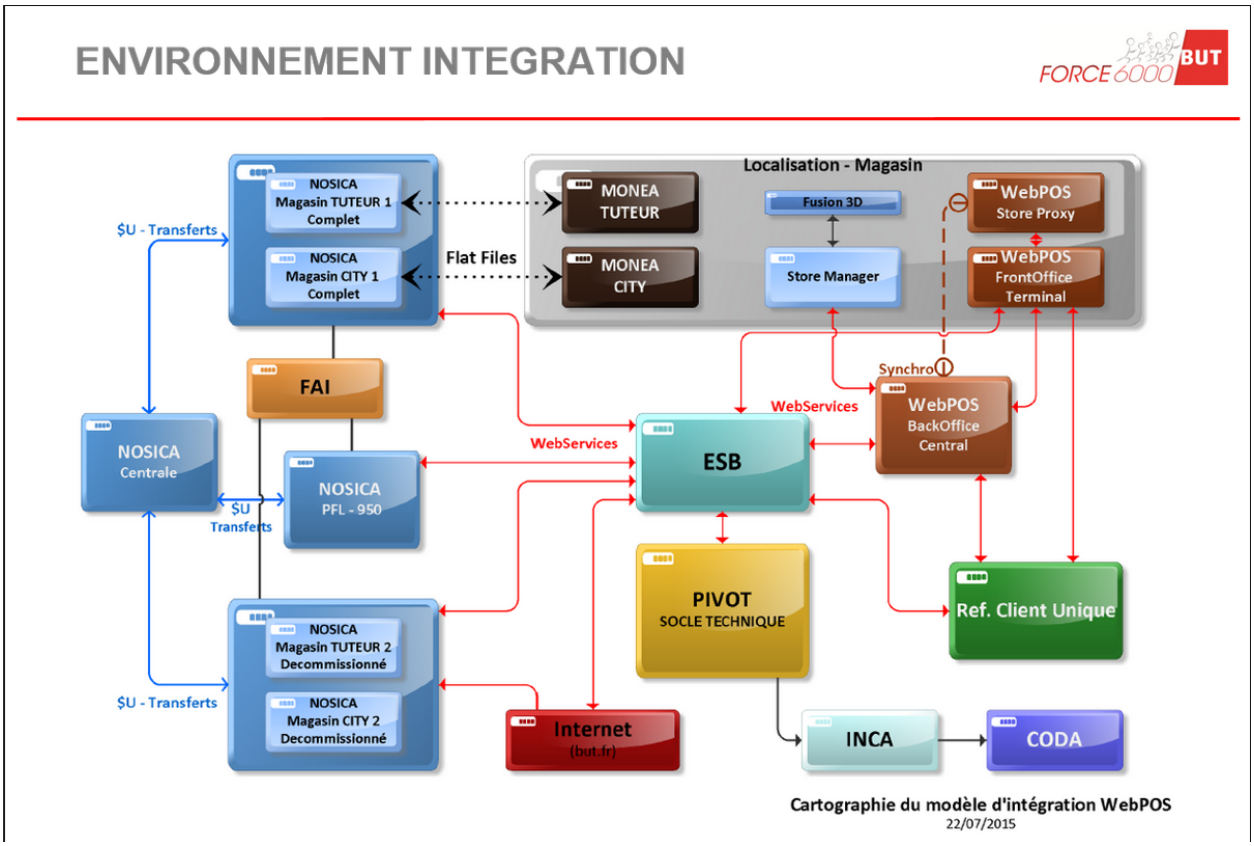

Figura 5: Mapa de sistemas actual de BUT

PROPÓSITO DEL PROYECTO BUT

La idea del proyecto nace de la necesidad del cliente de poseer un sistema que le permita gestionar los datos de la empresa de manera más centralizada y simple. Como objetivo del proyecto se pretende crear una plataforma de integración que permita integrar los servicios web externos al sistema de Openbravo. Asimismo, pretende integrar varios desarrollos *custom* propios del proyecto BUT, para importar entidades dentro del sistema.

INTEGRATION PLATFORM

La plataforma de integración o *integration platform* es un módulo que forma parte de la plataforma de desarrollo de Openbravo y que tiene como función proveer a los desarrolladores de unas funcionalidades necesarias para desarrollar las integraciones fácilmente. El desarrollo consiste en la integración de servicios web externos que permitan tanto la entrada como la salida de información utilizando un formato JSON estandarizado. Para la configuración y la visualización de la correcta ejecución de los procesos, dentro del mismo ERP tendremos la ventana llamada *Data In*, que mostrará las llamadas que se han hecho a los servicios web de Openbravo, y la ventana *Data Out*, que mostrará las llamadas que hayan sido hechas a partir de los servicios web de Openbravo. Finalmente la ventana *Entities Configuration*, se utiliza para configurar la URL del servicio web correspondiente a su respectiva entidad siguiendo el siguiente formato:

http://<serveraddress>/openbravo/ws/com.openbravo.but.integrationplatform.integration/*EntityName*

Además de la configuración de la URL es obligatorio respetar ciertos requisitos para la integración de las entidades mediante la plataforma. El formato del fichero JSON a enviar o recibir es uno de ellos. El formato básico a seguir es el siguiente, aunque dependiendo de la entidad este tendrá unos atributos u otros:

```
{"data": [{"SearchKey": "UnSearchKey", "Name": "UnName" }]}
```

El formato básico tiene como objetivo ser utilizado por un desarrollador que desea comprobar si la configuración ha sido realizada correctamente, ya que como antes he mencionado, esta parte del proyecto esta orientada a facilitar el trabajo a los desarrolladores que pretenden realizar integraciones mediante servicios web en el ERP.

DESARROLLOS CUSTOM

Los desarrollos *custom* se componen de una serie de módulos propios del proyecto cuyo desarrollo ha sido necesario para realizar la integración de las entidades. Se pueden dividir en tres conjuntos:

- Master Data Integration: consiste en la carga de los datos necesarios para realizar el arranque inicial del ERP. Se compone de la integración de las entidades *Services*, *Products*, *Product Assortments*, *Product Caracteristics*, *Product Prices*, *Customer* y *Nomenclature*.

- Transactional Data Integration: permite comunicar datos entre los sistemas de BUT y las transacciones de la plataforma Openbravo. Se compone de la integración de las entidades *Contramarque*, *StockMD*, *Customer*, *Orders*, *Returns*, *Shipments*, *Invoices*, *Payments*, *Cashups* y *GiftCards*. Los casos *StockMD* y *Customer* requieren de una integración especial debido a que se comunican con diferentes sistemas respecto a las demás entidades. En el caso de Stock MD, este se comunica con un sistema llamado NOSICA para el control de stock y reservas de manera directa desde el Web POS. En cambio *Customer*, se comunica con el sistema *RCU* (Ref. Unique Client) también desde el Web POS.

- Configuration Data Integration: consiste en la integración de la información necesaria para la configuración del ERP. Se compone de la integración de las entidades *Organization* / *Store Model*, *User Passwords* (Employee), *Roles* & *Permissions*, *Taxes*, *Warehouses* y *Stores*.

PLAN DEL PROYECTO

Para abarcar un desarrollo de tal magnitud ambas compañías han llegado al acuerdo inicial de realizar el proyecto en tres ciclos, aunque debido a los cambios de requisitos que se irán generando durante el transcurso es posible que acabe teniendo alguno más. Los ciclos previstos son:

1. Con una duración estimada de 10 semanas pretende abarcar los desarrollos requeridos en el *setup* de la plataforma Openbravo así como los que incluyen una entidad de mayor importancia funcional en la misma. También se incluyen algunas integraciones iniciales o básicas como son la de *RCU*(*Customer*) y *StockMD*.

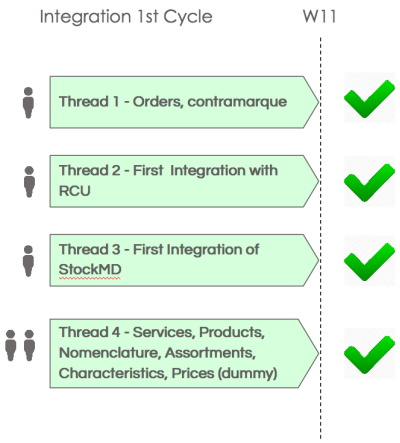

Figura 6: Integraciones del ciclo 1

2. Se estima una duración de 6 semanas y se pretende llevar a cabo modificaciones sobre lo realizado en el ciclo 1, además de la segunda fase de las integraciones de *RCU* y *StockMD* y la integración de las siguientes entidades:

    - *Warehouses*
    - *CashUps*
    - *Payments*
    - *Gift Cards*
    - *Taxes*
    - *Returns*
    - *Store Information*

3. El tercer ciclo no tiene aun una estimación prevista de duración aunque el plan pretende abarcar los siguiente puntos:

    - Tercera fase de integración de *RCU* y *StockMD*
    - Integración de *Shipments* e *Invoices*
    - Posibles modificaciones en *Store Information*
    - Test y automatización
    - Documentación técnica

OBJETIVO Y ALCANCE PERSONAL DENTRO DEL PROYECTO BUT

En lo que a mi respecta, en este documento solo se refleja la participación en el proyecto hasta el segundo ciclo, debido a que las fechas manejadas en el tercero están fuera del plazo de entrega del TFG. Tampoco se darán detalles sobre los procesos que no interfieren con las tareas detalladas a continuación dentro del mismo.

La integración se llevara a cabo de forma paralela o simultanea entre ambas organizaciones para disminuir el coste en tiempo del proyecto. Para ello, debido a que Openbravo no tiene acceso directo sobre los sistemas de BUT es necesario desarrollar un entorno *Fake* que simule la respuesta que obtendríamos de su sistema. De esta forma, se podrá realizar el desarrollo de las integraciones.

El desarrollo consistirá en servicios web que permitan emular la información que proporciona el sistema del cliente dependiendo de la necesidades de datos del ERP en sus flujos funcionales. El intercambio de dichos datos se realizará mediante servicios web de tipo REST que transferirán un fichero con un determinado formato JSON dependiendo de la entidad a transferir. Una de las tareas ha realizar y expuestas en el documento será el desarrollo de dicho entorno.

El entorno no manejará peticiones de todas las entidades mencionadas hasta el momento debido a la madurez o funcionamiento de cada desarrollo. Únicamente se van

abarcar los servicios web de las entidades mencionadas hasta el ciclo 2 excluyendo los siguientes:

- *RCU*: debido a la complejidad del desarrollo, hasta el ciclo 3 *RCU* no se comunica con el servicio web, ya que es más prioritario desarrollar la lógica del módulo que implementar la comunicación con el servicio. Los datos necesarios se insertaran a modo de *Hard-code* (directamente en el código).

- *Returns* y *Store Information*: debido a que ambos desarrollos no han alcanzado la suficiente madurez como para requerir la respuesta del entorno *Fake*, no se han contemplado los casos.

Siguiendo con el proceso EDL (detallado posteriormente), es uno de los más importantes dentro del proyecto de integraciones. Es el encargado de importar las entidades mencionadas anteriormente mediante servicios web de tipo REST en el ERP. El proceso se ha tratado y estudiado en gran medida durante el desarrollo del entorno *Fake* ya que es el principal proceso con el que interactúa. Gracias a ello, he adquirido los conocimientos suficientes para realizar las tareas de testing relacionadas con el proceso que serán expuestas en este documento.

Sobre lo realizado comentar que únicamente se incluirá en el producto final o en la entrega al cliente las pruebas tanto de performance como las unitarias, ya que el propósito del entorno *Fake* es permitir, facilitar y mejorar los demás desarrollos que dependan de la respuesta de este pero no tiene valor alguno por si mismo para el cliente.

HARDWARE Y SOFTWARE PARA EL DESARROLLO

Hardware

En principio, cualquier equipo actual debería ser suficiente para realizar el desarrollo y las correspondiente configuraciones, aunque es recomendable un equipo con al menos los siguientes requisitos hardware para el desarrollo del entorno *Fake* y la realización de las pruebas sobre el proceso EDL:

- Procesador: Intel Core i3 (2GHz)
- 4GB de memoria RAM
- Monitor con una resolución 1680x1050 o mayor.

Software

A continuación se definen los elementos software necesarios para el desarrollo del entorno, como para las pruebas de performance y unitarias.

- Maquina local además de dos entornos virtuales de tipo Amazon EC2, para la instalación del ERP, así como del entorno *Fake* y la realización de pruebas de performance y unitarias.

- Versión 3.0PR16Q.1 de Openbravo ERP. Aunque no sea estrictamente necesario, el desarrollo del proyecto se ha realizado sobre esta versión, que garantiza la estabilidad de la versión como se ha mencionado anteriormente.

- Para el servidor multiplataforma basado en JAVA donde se instalara el ERP, un SO de tipo Windows, Linux, FreeBSD, Mac OSX, Solaris,... y una arquitectura de tipo x86, x86_64, IA64, Sparc, PowerPC, AIX.

- Base de datos de tipo PostgreSQL en cualquiera de las versiones 9.1.x, 9.2.x, 9.3.x, 9.4.x, 9.5.x aunque la recomendada sea la 9.3.x. También se podría haber utilizado Oracle en las versiones 11g o 12c aunque la recomendada es la 11gR2.

- JDK (Java Development Kit) de tipo OpenJDK, Sun JDK o IBM JDK para el desarrollo y ejecución en JAVA. Las versiones soportadas son la 6,7 y 8 aunque la recomendada sea la 7.x.

- Apache Tomcat como servidor web para el ERP. Las versiones soportadas son la 6.0.x, 7.0.x y 8.0.x, aunque la recomendada sea la 7.0.x.

- Apache Ant para la automatización de las tareas de tipo “set up” o compilación, creación de base de datos o ejecución de test. Versiones soportadas: 1.7, 1.8.x, 1.9.x. Versión recomendada: 1.9.x.

- Es recomendable utilizar los paquetes Apache HTTP Server en la última 2.4.x, Apache mod_jk connector en la última 1.2.x y Apache Tomcat Native en la última 1.1.x aunque no es obligatorio tenerlos instalados.

- Se requiere alguno de los siguientes navegadores para la configuración del funcionamiento y visualización de la interfaz del ERP.
    - Firefox, versiones soportada: 38 o superior. Versión recomendada: 45 o superior.
    - Chrome, versiones soportadas: 48. Versión recomendada: 49 o superior.
    - Internet Explorer, versiones soportadas: 9 o superior. Versión recomendada: 11 o superior.
    - Edge, versiones soportadas: 12. Versión recomendada: 13 o superior.
    - Safari, versiones soportadas: 8. Versión recomendada: 9 o superior.

- Entorno de desarrollo o IDE de cualquier tipo que permita el desarrollo de aplicaciones en Java, además de soportar Apache Tomcat, siendo el recomendado Eclipse Luna.

- Mercurial como herramienta para la gestión del código del repositorio remoto, siendo necesario algún servicio de repositorios, como por ejemplo Bitbucket.

- Apache JMeter para la realización de las pruebas de performance contra los servicios web y Munin para la monitorización en gran detalle de las mismas.

- Librería JUnit para realizar las pruebas unitarias contra el proceso EDL. Versión recomendada: 4.

2. Análisis de requisitos

En el siguiente apartado se detalla el análisis de requisitos del entorno *Fake.* En este caso únicamente tenemos requisitos funcionales, ya que el desarrollo no tiene ningún requisito que imponga alguna restricción, tanto en el diseño, como en la implementación.

REQUISITOS FUNCIONALES

Los requisitos funcionales se han separado en tres grupos dependiendo del módulo de integración con el que se comunica o atiende. En este caso tenemos los siguientes módulos con sus respectivas entidades

Integration Transaction Module – Order / Contramarque

RF_Transaction1: Implementar servicio web a la escucha en la URL correspondiente.
RF_Transaction2: Implementar funciones de creación de respuestas de tipo *Standard* (especificadas anteriormente en el apartado de *Integration Platform*).
RF_Transaction3: Añadir validación de la entidad *Order* / *Contramarque*
RF_Transaction4: Añadir lógica de respuesta del servicio web que actúa en base a la validación.

Integration StockMD Module – StockMD

RF_StockMD1: Implementar servicio web a la escucha en la URL correspondiente.
RF_StockMD2: Añadir respuesta de un fichero JSON estático.

Integration Master – Services, Products, Nomenclature, Assortments, Characteristics, Prices & Warehouses

RF_Master1: Implementar cliente que envié las entidades *Services*, *Products*, *Assortments*, *Characteristics*, *Prices* y *Warehouses* en formato JSON a su correspondiente URL.
RF_Master2: Añadir funcionalidad de lectura de un fichero JSON.

RF_Master3: Añadir funcionalidad de configuración de la ip y puerto de la maquina a la que apuntar.
RF_Master4: Añadir funcionalidad para realizar peticiones repetidas.
RF_Master5: Añadir funcionalidad de envío aleatorio de ficheros JSON.
RF_Master6: Añadir validación de la entrada del usuario.
RF_Master7: Implementar servicio web a la escucha en la URL correspondiente para la entidad *Nomenclature* que devuelva el fichero JSON correspondiente.

Integration CashUps / Payments / Gift Cards

RF_Integration1: Implementar servicio web a la escucha en la URL correspondiente para cada entidad.
RF_Integration2: Añadir validación de la entidad recibida.
RF_Integration3: Añadir lógica de respuesta que actúa en base a la validación de la entidad.
RF_Integration4: Añadir funciones de creación de respuestas de tipo *Standard* (misma que RF_Transaction2)

Integration Taxes

RF_Taxes1: Implementar servicio web a la escucha en la URL correspondiente.
RF_Taxes2: Añadir funcionalidad de respuesta de un fichero JSON de la entidad *Taxes*.
RF_Integration4: Añadir funciones de creación de respuestas de tipo *Standard* (misma que RF_Transaction2)

3. Diseño

En este apartado se detallarán los aspectos del diseño del entorno *Fake*, además del análisis de flujos tanto de los test unitarios como los de rendimiento. En primer lugar se tratará el entorno *Fake* tanto a nivel de arquitectura como a nivel funcional analizando sus casos de uso. Después, se describirá el análisis de flujos de ambos test simultáneamente ya que coinciden en ambos.

ENTORNO FAKE

El entorno se compone de un servidor y un cliente ambos implementados en java y que utilizan el protocolo HTTP. La parte del servidor pretende atender tanto a las peticiones que soliciten los datos de alguna entidad, como por ejemplo *Product Category*, como a peticiones que envían información destinada a los servidores de BUT pero que requieren una respuesta. La función principal del entorno es simular la respuesta del sistema PIVOT ESB. El siguiente gráfico nos permite tener una visión más general de lo que se requiere diseñar.

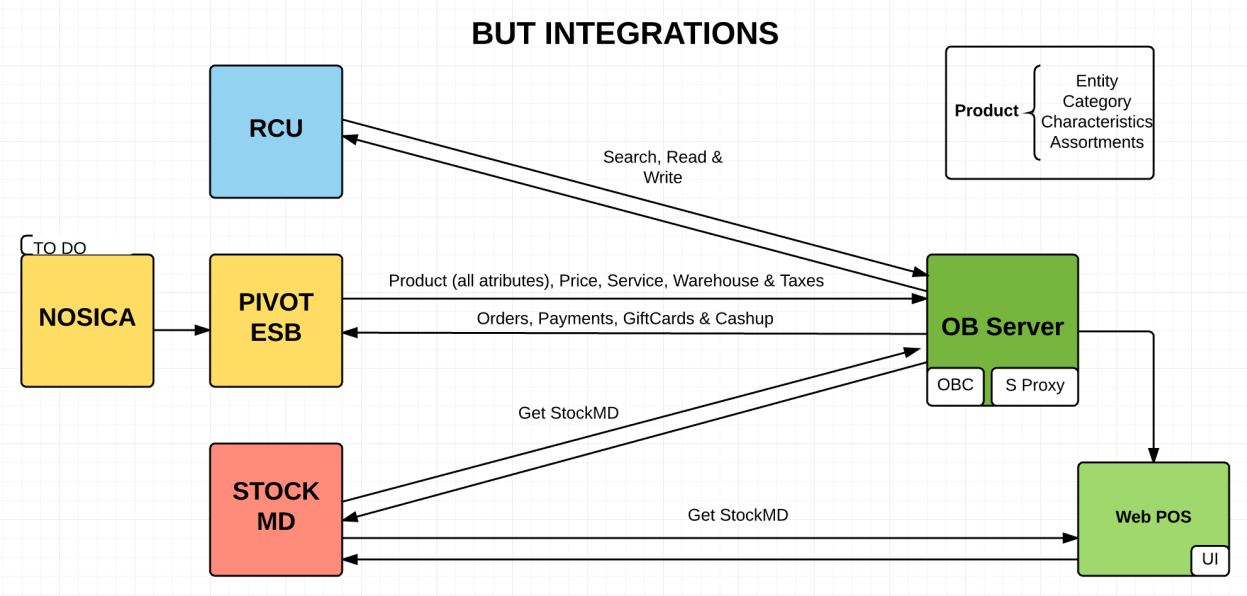

Figura 7: Visión general del Fake Environment

En el grafico, observamos tres sistemas que se comunican con el ERP mediante el servidor de Openbravo, RCU, PIVOT ESB y STOCK MD. Aunque en el grafico solo STOCK MD se comunica de manera directa con el Web POS, todos los servicios que se conectan con PIVOT ESB a través del OB Server también lo hacen. Con RCU ocurre lo mismo como hemos comentado anteriormente, aunque parte no se incluye en el documento ya que no está completamente definida. El entorno *Fake* solo emula la respuesta de los sistemas PIVOT ESB y STOCK MD. Todos los servicios que se dirigen hacia Openbravo provienen de la parte del cliente, que en este caso se lanzará y ejecutará de manera manual. Es decir, el cliente no será un demonio que ira haciendo peticiones de manera autosuficiente, si no que se lanzará mediante una interacción con un usuario.

El servicio StockMD requiere de algún comentario adicional. Cuando se comenzó con el desarrollo de la aplicación se detectó que este servicio era extraordinariamente necesario. Esto, junto con la presión del plazo de entrega, llevo a concluir que era necesario un desarrollo muy rápido. Es por ello que se optó por desarrollar un servidor con el *framework* NodeJS de la forma más simple y rápida posible.

StockMD cuenta con un único caso de uso, *Get StockMD*. Recibe una petición y en base a ella devuelve un fichero JSON de la entidad *StockMD*. Este caso de uso se lanza desde el cliente Web POS cuando se pulsa cualquiera de los botones *Store stock*, *Other stores stock* o *Get Stock Date*. Los dos primeros se muestran en la siguiente imagen, el tercero se encuentra disponible en el menú desplegable que se encuentra en la esquina superior izquierda. Estos botones lanzan un proceso que solicita la información tanto del stock de la tienda actual, como el presente en las demás tiendas. También puede suceder que el ERP solicite la misma información, aunque en todo caso será el usuario el que inicia el proceso, no será otro proceso en *background* el que lo haga.

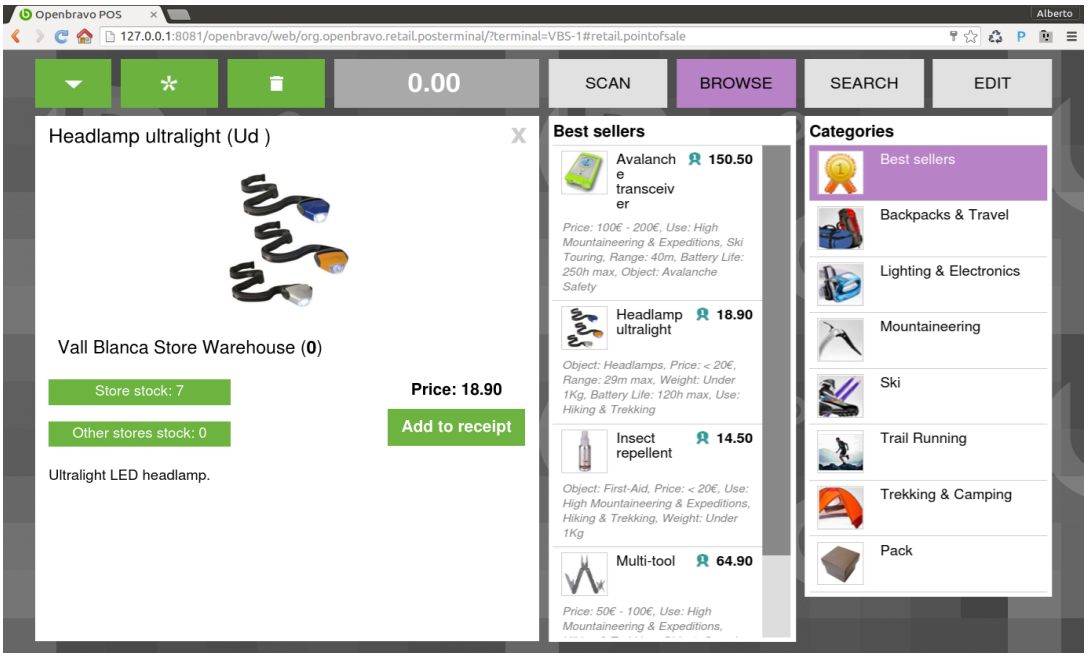

Figura 8: Página principal de un producto en el Web POS

Más adelante en el apartado de implementación, se mencionarán los aspectos de la interfaz así como su funcionamiento en más detalle, aunque debo mencionar que se trata de un servicio web que atiende peticiones HTTP de tipo POST.

CASOS DE USOS

STOCKMD

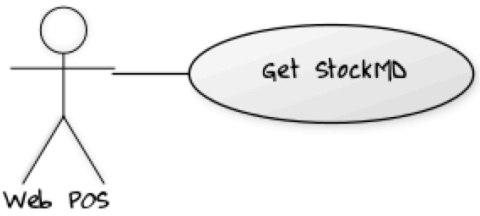

Figura 9: Diagrama de casos de uso de StockMD

SERVIDOR FAKE

El servidor tiene una mayor complejidad, tanto en la parte arquitectónica como en la funcional, que al realizado en el caso de *StockMD*. En este caso se abarca más de un servicio web, por lo que se complica la manera de gestionar las peticiones, aunque como observaremos en el diagrama de casos de uso, se pueden agrupar la mayoría de ellas en un tipo. Se ha optado por utilizar una arquitectura que soporte multihilo, proporcione concurrencia y permita clasificar las peticiones dependiendo de la dirección para luego ser tratadas. Para ello el servidor utilizará un conjunto de *handlers* de modo que las solicitudes puedan ser tratadas por separado en función de su URL. Una vez clasificada la petición se trata dentro del mismo manejador. El servidor se compone de los casos de uso expuestos en el diagrama que viene a continuación:

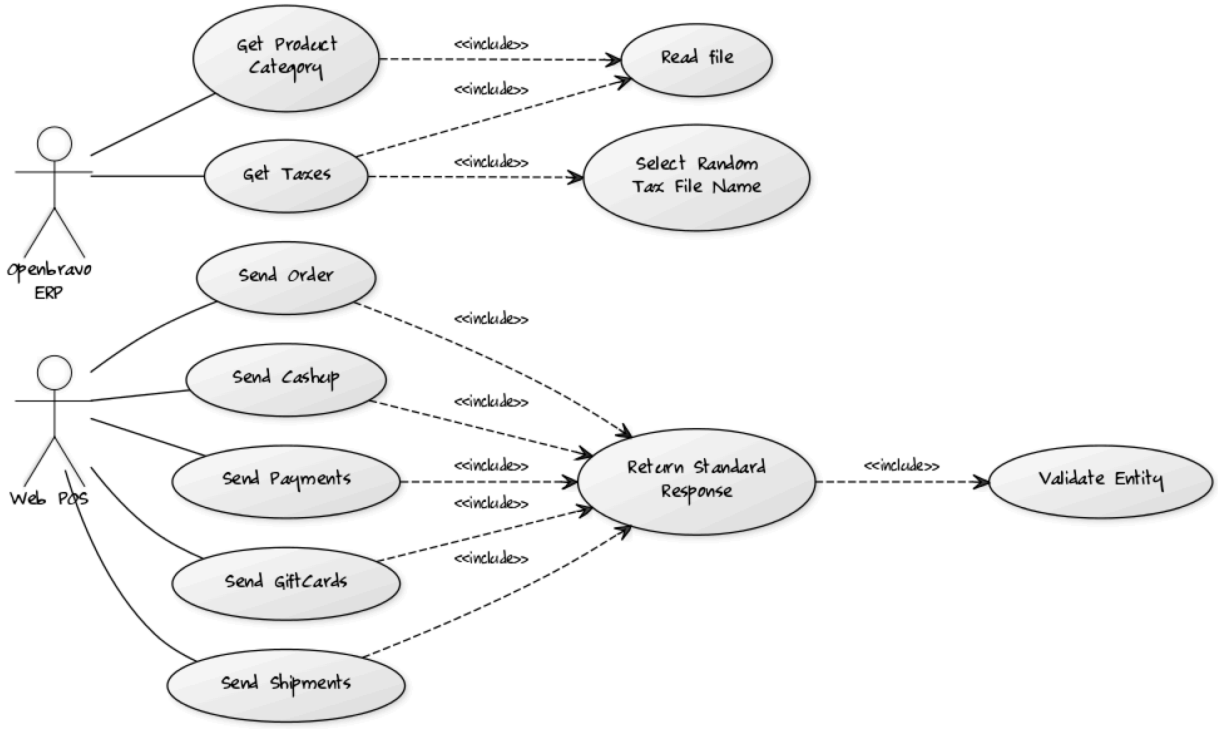

Figura 10: Diagrama de casos de uso del Fake Server

ACTORES

Debido a que se va a desarrollar un servidor, el actor es el cliente ya que es el que inicia el proceso. En este caso, la plataforma de Openbravo es el actor que arranca la comunicación con los servicios de categorías de productos y tasas mediante procesos *background* o por petición del usuario. Para los demás casos, el cliente es el Web POS ya que se comunica con los sistemas externos sin utilizar el OB Server.

CASOS DE USO

GET PRODUCT CATEGORY

Dependencias: Caso de uso *Read File*
Precondición: Se realiza la importación de un producto en el ERP cuya categoría no coincide con el árbol de categorías.
Flujo básico:

1. Para actualizar el árbol, la plataforma realiza una petición en formato JSON al servicio web del entorno *Fake*.
2. Este lee el fichero “arc-response-nomenclature.json”.
3. Devuelve un objeto JSON creado a partir del fichero mediante la conexión con la plataforma.

Flujos alternativos:

1. Petición JSON no cumple con el formato. Se detiene el proceso y finaliza la conexión.
2. Fichero “arc-response-nomenclature.json” no encontrado. Se detiene el proceso y finaliza la conexión.
3. Fallo en la conexión HTTP. Se detiene el proceso.

Postcondición: Petición atendida y fichero JSON enviado aunque pueda haberse perdido debido a algún fallo en la conexión.

GET TAXES

Dependencias: Casos de uso *Read File* y *Select Random Tax File Name*.
Precondición: Proceso en *background* que realiza llamadas al servicio web, activo o arrancado en el ERP.
Flujo básico:

1. Para actualizar todas las tasas existentes realiza una petición en formato JSON a su correspondiente servicio del entorno *Fake*.
2. Este selecciona un fichero de *Taxes* al azar.
3. Devuelve el objeto JSON creado a partir de este.

Flujos alternativos:

1. Petición JSON no cumple con el formato. Se detiene el proceso y se fuerza el final de la conexión.
2. Ficheros de tipo *Taxes* no existentes. Se detiene el proceso y se cierra la conexión.
3. Fallo en la conexión HTTP. Se detiene el proceso.

Postcondición: Petición atendida y fichero JSON enviado aunque no sea fiable.

READ FILE

Dependencias: No hay dependencias.
Precondición: Ninguna precondición.

Flujo básico: Se recibe el nombre del fichero a leer y se devuelve referencia a este.
Flujo alternativo: Fichero no existente. Finaliza el proceso.
Postcondición: Devuelto objeto fichero.

SELECT RANDOM TAX FILE NAME

Dependencias: No hay dependencias.
Precondición: Fichero de configuración rellenado con el número máximo de ficheros con el formato “arc-response-taxesX.json” presentes en el directorio “files”.
Flujo principal:

1. Se lee el número del fichero de configuración.
2. Se calcula un número aleatorio entre 0 y 1.
3. Se realiza la división 1/numeroDeFicheros y se suma esta cantidad hasta obtener un valor mayor que el número aleatorio calculado. La cantidad de sumas realizadas determinar el número del fichero seleccionado.

Flujo alternativo: Fichero de configuración no existente. Se finaliza el proceso.
Postcondición: Devuelto el nombre del fichero.

SEND ORDER

Dependencias: Caso de uso *Return Standard Response*.
Precondición: Finalización de una orden en el cliente Web POS.
Flujo Principal:

1. Se recibe la orden creada en formato JSON desde el Web POS.
2. Se valida la entidad.
3. Devuelve OK o KO en base a la validación.

Flujos alternativos:

1. Orden recibida en la petición con formato incorrecto. Se detiene el proceso y finaliza la conexión.
2. Fallo en la conexión HTTP. Se detiene el proceso.

Postcondición: Petición atendida.

SEND CASHUP

Dependencias: Caso de uso *Return Standard Response*.
Precondición: Realizar caja en el cliente Web POS.
Flujo Principal:

1. Se recibe la información sobre la caja realizada en formato JSON desde el Web POS.
2. Se valida la entidad.
3. Devuelve OK o KO en base a la validación.

Flujos alternativos:

1. Información sobre la caja recibida en la petición con formato incorrecto. Se detiene el proceso y finaliza la conexión.
2. Fallo en la conexión HTTP. Se detiene el proceso.

Postcondición: Petición atendida.

SEND PAYMENTS

Dependencias: Caso de uso *Return Standard Response*.
Precondición: Realizar un pago en el cliente Web POS.
Flujo Principal:

1. Se recibe la información sobre el pago realizada en formato JSON desde el Web POS.
2. Se valida la entidad.
3. Devuelve OK o KO en base a la validación.

Flujos alternativos:

1. Pago recibido en la petición con formato incorrecto. Se detiene el proceso y finaliza la conexión.
2. Fallo en la conexión HTTP. Se detiene el proceso.

Postcondición: Petición atendida.

SEND GIFTCARDS

Dependencias: Caso de uso *Return Standard Response*.
Precondición: Utilizar una tarjeta regalo en el cliente Web POS.
Flujo Principal:

1. Se recibe la información sobre el envió realizado en formato JSON desde el Web POS.
2. Se valida la entidad.
3. Devuelve OK o KO en base a la validación.

Flujos alternativos:

1. Tarjeta regalo recibida en la petición con formato incorrecto. Se detiene el proceso y finaliza la conexión.
2. Fallo en la conexión HTTP. Se detiene el proceso.

Postcondición: Petición atendida.

SEND SHIPMENTS

Dependencias: Caso de uso *Return Standard Response.*
Precondición: Registrar nuevo envío en el cliente Web POS.
Flujo Principal:

1. Se recibe la información sobre el envió realizado en formato JSON desde el Web POS.
2. Se valida la entidad.
3. Devuelve OK o KO en base a la validación.

Flujos alternativos:

1. Información del envió recibida en la petición con formato incorrecto. Se detiene el proceso y finaliza la conexión.
2. Fallo en la conexión HTTP. Se detiene el proceso.

Postcondición: Petición atendida.

RETURN STANDARD RESPONSE

Dependencias: Caso de uso *Validate Entity*
Precondición: Recibir una petición de alguno de los tipos que incluye el caso en el servidor Fake.
Flujo Principal:

1. Se valida la entidad recibida.
2. Devuelve OK o KO en base a la validación.

Flujo alternativo: No existe ningún flujo alternativo.
Postcondición: Petición atendida.

VALIDATE ENTITY

Dependencias: No hay dependencias.
Precondición: Recibir una entidad para validar.
Flujo Principal:

1. Se realiza una comprobación de la existencia de todos los atributos obligatorios además de su contenido.
2. Devuelve *true* o *false* en base a la validación.

Flujo alternativo: No existe ningún flujo alternativo.
Postcondición: Entidad validada.

PROCESO EDL

Antes de empezar detallando los casos de uso o como se han estructurado, conviene comentar que es el proceso EDL, ya que es la única razón de existencia del cliente *Fake*. El proceso EDL es un proceso que tiene como objetivo importar entidades al ERP mediante su recepción vía servicios web de tipo HTTP. El proceso se usará para las cargas iniciales y nuevas inserciones que realizara BUT contra el ERP. En nuestro caso, solo nos interesan las entidades *Product Category*, *Product*, *Service*, *Warehouse* y *Price*. Este proceso realiza comprobaciones sobre las entidades y, si las da por validas, las inserta en la plataforma. Debido a las dependencias existentes entre las entidades, el proceso tiene obligación de seguir una secuencia para insertarlas correctamente.

Un producto depende de las categorías, por lo que es necesario tenerlas presentes antes de insertarlo. A su vez, el precio depende de un producto. Un servicio es algo similar a un producto con varias peculiaridades, pero en esencia lo mismo. Un servicio también requiere que existan las categorías de productos.

Finalmente, un almacén no depende de las categorías de productos. En el siguiente diagrama de secuencia del proceso EDL podremos observar el orden de los pasos para importar las entidades mencionada. Comentar que el paso 1 y la integración de la entidad *Warehouse* no son dependientes, pero se ha optado por esta representación por que el proceso *Get Product Category Tree* se caracteriza por ser un proceso de *setup* del entorno.

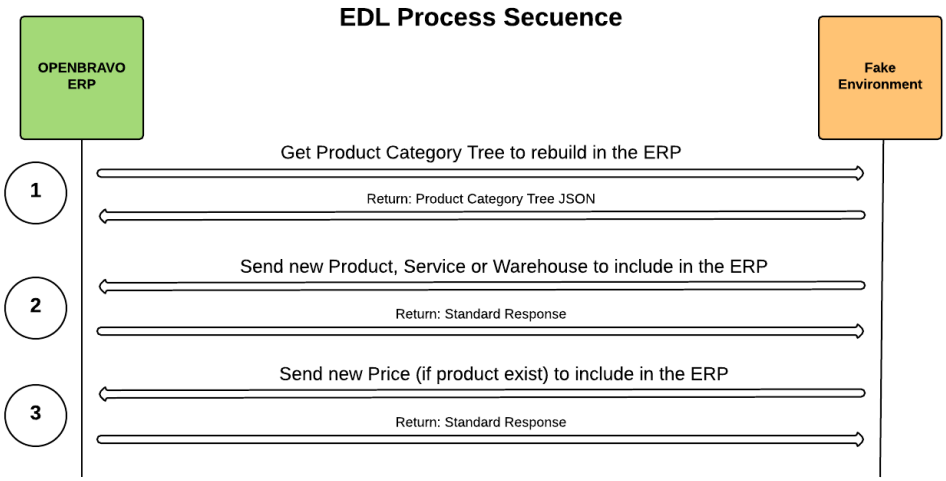

Figura 11: Diagrama de secuencia del proceso EDL

CLIENTE FAKE

El Cliente *Fake* tiene como objetivo abarcar los procesos mencionados en el gráfico anterior a excepción de *Get Product Category Tree*. Se trata de un cliente HTTP que realiza peticiones de tipo POST contra la plataforma de Openbravo. La arquitectura no tiene complejidad alguna. Todas las peticiones que se realizan tienen un flujo similar, es decir, se puede utilizar la misma función para generar todas variando parámetros como la URL o la entidad a enviar. Por lo tanto, se optara por un diseño que nos permita independizar esa parte y que nos permite reusarla.

El cliente contara con una serie de opciones para su configuración. El usuario podrá configurar la ip de por defecto, del servicio web al que mandar la petición HTTP; el puerto en caso de querer cambiarlo a alguno diferente al 80 y otras opciones referentes al envió de la petición. Podrá insertar el nombre del fichero JSON a leer para después ser enviado. También dispondrá de un pequeño sistema para la escritura de los tiempos de ejecución en un fichero para el análisis de rendimiento de la plataforma Openbravo.

Para realizar todas las funcionalidades mencionadas el cliente contará con funciones de apoyo para realizar tareas como son la lectura de un fichero JSON solicitando al usuario el nombre de este y la escritura de los resultados en un fichero. Pasemos a tratar los casos de uso presentes.

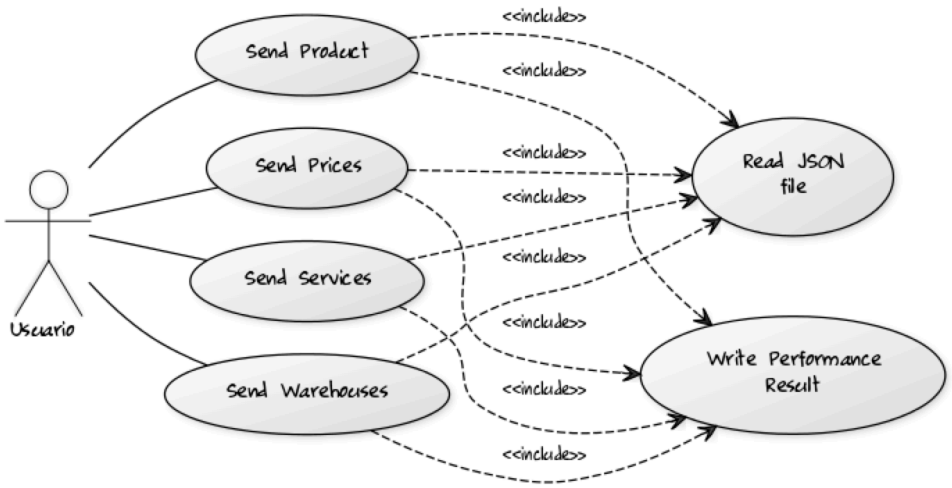

Figura 12: Diagrama de casos de uso del Fake Client

ACTORES

En este caso dispones de un único actor. La aplicación se ha construido con el fin de proporcionar a un desarrollador la simulación del servicio de BUT para realizar algún tipo de prueba sobre su desarrollo. Por lo tanto, este será nuestro único actor.

CASOS DE USO

SEND PRODUCT

Dependencias: Casos de uso *Read JSON file* y *Write Performance Result*.
Precondición: El usuario requiere la información de un producto.
Flujo Principal:

1. Usuario arranca la aplicación.
2. Configura el servicio si es necesario.
3. Selecciona la opción *Send Product JSON* en el menú.
4. introduce el nombre del fichero, si lo desea enviar repetidas veces y si es así la cantidad de repeticiones.

Flujos alternativos:

1. Fichero a enviar no encontrado. Se detiene el proceso y se vuelve al menú.

2. Fichero a enviar con un formato JSON incorrecto. Se detiene el proceso y se vuelve al menú.
3. Servicio web mal configurado. Se detiene el proceso y se muestra un mensaje comunicando al usuario que ejecute de nuevo la aplicación y cambia los parámetros de configuración como la ip y el puerto.

Postcondición: Usuario ha enviado correctamente el fichero JSON de producto.

SEND PRICES

Dependencias: Casos de uso *Read JSON file* y *Write Performance Result*.
Precondición: El usuario requiere la información del precio de un producto.
Flujo Principal:

1. Usuario arranca la aplicación.
2. Configura el servicio si es necesario.
3. Selecciona la opción *Send Prices JSON* en el menú.
4. introduce el nombre del fichero, si lo desea enviar repetidas veces y si es así la cantidad de repeticiones.

Flujos alternativos:

1. Fichero a enviar no encontrado. Se detiene el proceso y se vuelve al menú.
2. Fichero a enviar con un formato JSON incorrecto. Se detiene el proceso y se vuelve al menú.
3. Servicio web mal configurado. Se detiene el proceso y se muestra un mensaje comunicando al usuario que ejecute de nuevo la aplicación y cambia los parámetros de configuración como la ip y el puerto.

Postcondición: Usuario ha enviado correctamente el fichero JSON del precio del producto.

SEND SERVICES

Dependencias: Casos de uso *Read JSON file* y *Write Performance Result*.
Precondición: El usuario requiere la información de un servicio.
Flujo Principal:

1. Usuario arranca la aplicación.
2. Configura el servicio si es necesario.
3. Selecciona la opción *Send Services JSON* en el menú.
4. introduce el nombre del fichero, si lo desea enviar repetidas veces y si es así la cantidad de repeticiones.

Flujos alternativos:

1. Fichero a enviar no encontrado. Se detiene el proceso y se vuelve al menú.
2. Fichero a enviar con un formato JSON incorrecto. Se detiene el proceso y se vuelve al menú.

3. Servicio web mal configurado. Se detiene el proceso y se muestra un mensaje comunicando al usuario que ejecute de nuevo la aplicación y cambia los parámetros de configuración como la ip y el puerto.
4. Postcondición: Usuario ha enviado correctamente el fichero JSON de un servicio.

SEND WAREHOUSES

Dependencias: Casos de uso *Read JSON file* y *Write Performance Result*.
Precondición: El usuario requiere la información de un almacén.
Flujo Principal:

1. Usuario arranca la aplicación.
2. Configura el servicio si es necesario.
3. Selecciona la opción *Send Warehouses JSON* en el menú.
4. introduce el nombre del fichero, si lo desea enviar repetidas veces y si es así la cantidad de repeticiones.

Flujos alternativos:

1. Fichero a enviar no encontrado. Se detiene el proceso y se vuelve al menú.
2. Fichero a enviar con un formato JSON incorrecto. Se detiene el proceso y se vuelve al menú.
3. Servicio web mal configurado. Se detiene el proceso y se muestra un mensaje comunicando al usuario que ejecute de nuevo la aplicación y cambia los parámetros de configuración como la ip y el puerto.

Postcondición: Usuario ha enviado correctamente el fichero JSON de un almacén.

READ JSON FILE

Este caso se ha detallado anteriormente en la parte del servidor. Tiene el mismo funcionamiento.

WRITE PERFORMANCE RESULT

Dependencias: No hay dependencias.
Precondición: Requiere de los tiempos de inicio y final de los procesos, número de mensajes OK, KO y el total, así como el nombre del tipo de proceso ejecutado.
Flujo Principal:

1. Proceso requiere la funcionalidad.
2. Este calcula el tiempo de ejecución comparando los valores de tiempos proporcionados.
3. Escribe los datos en un fichero junto al nombre del proceso y la cantidad de envíos realizados..

Flujo alternativo: No se encuentra el fichero donde escribir los datos del test de performance. El proceso se detiene, muestra un mensaje informando al usuario y vuelve al menú.

DIAGRAMAS DE CLASES

Como antes he comentado, no se darán más detalles del caso *StockMD* debido a su simpleza, por lo que comenzaremos con la parte del servidor. A la hora de realizar la implementación de este, se ha tenido en cuenta la necesidad de separar las peticiones dependiendo de la URL, ya que el funcionamiento no es el mismo en todos los casos. La estructura se compone de un servidor que define como contexto varios *handler*-s cada uno con su propia clase y funcionalidad desarrollada en el método handle(), que es el método ejecutado al recibir una petición. Además se incluye una clase con funcionalidades comunes para todos los manejadores como son la lectura de ficheros, creación de respuestas,…

En este caso, los manejadores son de tipo HTTP. Para implementar dicha lógica, JAVA nos proporciona la clase *HttpHandler.java* que puede ser extendida. En el siguiente diagrama no se ve reflejado ese aspecto debido a que es una parte relacionada con la arquitectura de JAVA y se utiliza para poder establecer en el servidor un *handler* como contexto.

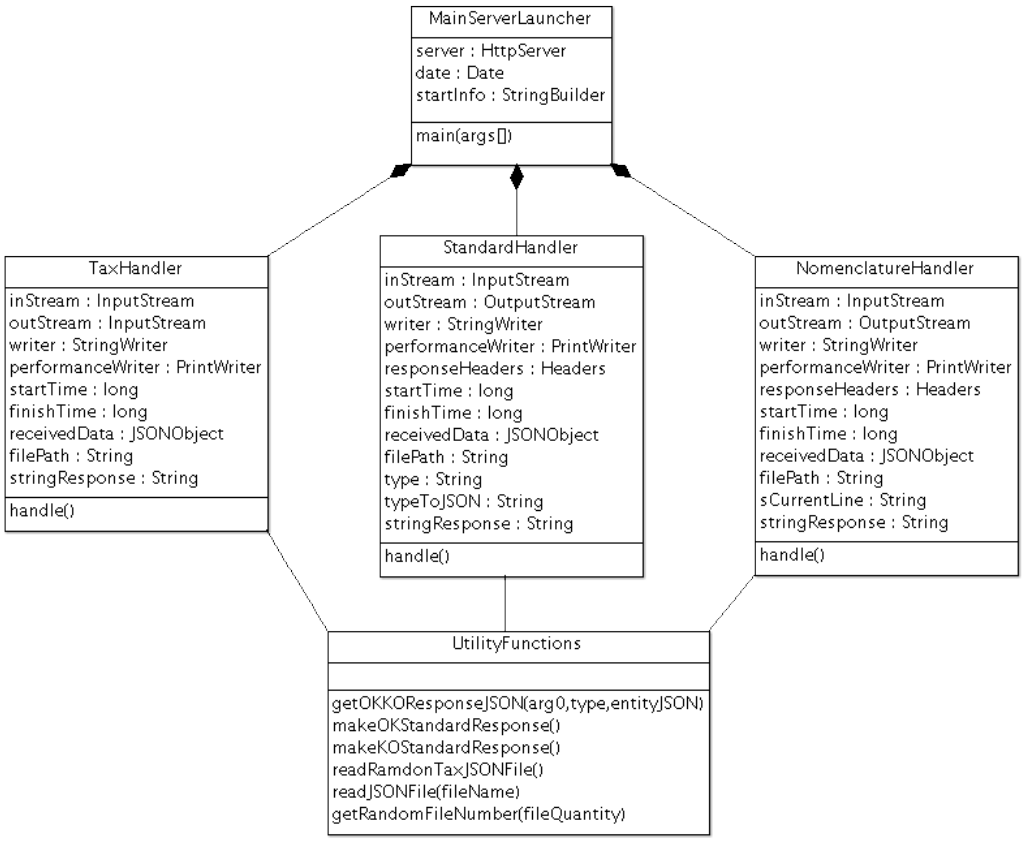

Figura 13: Diagrama de clases del Servidor Fake

En cuanto a la parte del cliente, en este caso la estructura es bastante más simple, debido a la similitud funcional de los casos de uso que lo componen. Se ha optado por crear una clase que ejerce el papel de lanzador y que permite configurar la ip y el puerto y que tendrá

una instancia de nuestra clase cliente. La clase *ButSystemClient* será la encargada de mostrar el menú y generar la petición HTTP correspondiente utilizando los métodos de la clase *UtilityFunctions*. Finalmente, en esta última clase es donde estará implementa la funcionalidad al nivel más bajo, como son las tareas de creación de conexiones, lectura de fichero, …

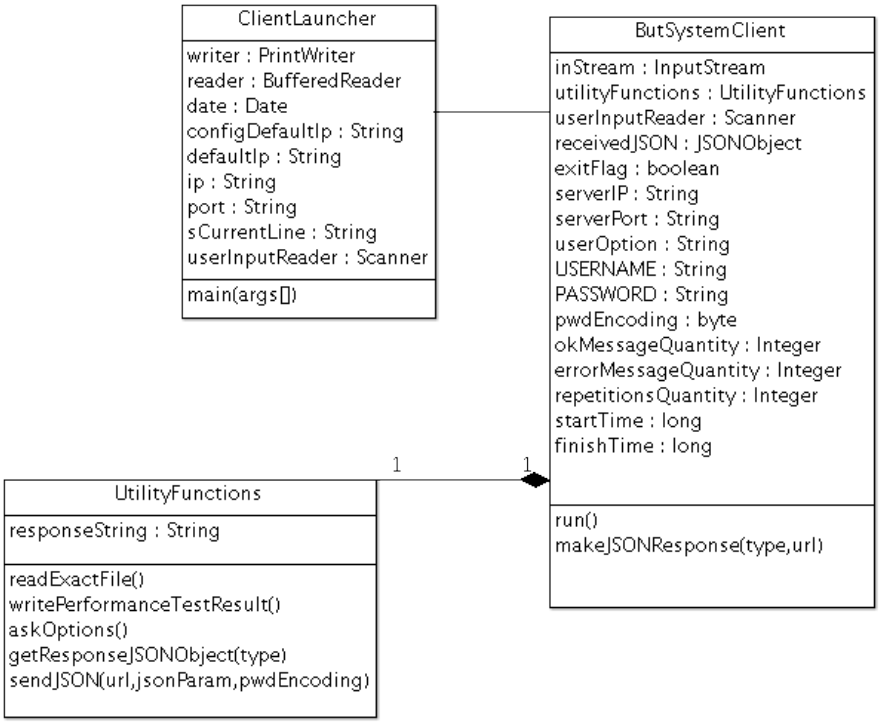

Figura 14: Diagrama de clases del Cliente Fake

TEST UNITARIOS Y DE PERFORMANCE

Los test pretenden comprobar el correcto funcionamiento y rendimiento del proceso EDL. Para ello, se ha estudiado el proceso y los flujos que pueden ser creados por las características de cada entidad recibida. Analicemos las especificaciones de *Product*, *Services*, *Prices* y *Warehouses*, la respuesta y flujos posibles del proceso ante estos.

PRODUCT

Structure:

- **Product_ID (String)**
- **Name (String)**
- **Description[ ] (String)**
- **TaxCategory**
  - **TaxeCode (String)**
  - **TaxeName (String / null)**

Example:

```
"Product_ID":"__10000000001",
"Description":[
   "description1",
   "last descriptionnnnnEND."
],
"TaxCategory":{
     "TaxeCode":"TVA",
   "TaxeName":"TVA normale",
```

        - TaxeRate (String)
        - StartDate (String / null)
- TaxDEA
    - DEA_Code (String)
    - DEA_Amount (String)
- DEEE_Amount (String/null)
- Product Category
    - Code_Famille (String)
    - Code_Sous_Famille (String)
- Code_EAN (String)
- Brand (String / null)
- Product_Type (String)
- Grouped_Product (String(1))
- Show_stock_screen (String(1))
- Editable_Price (String(1))
- Show_Characteristics_Description (String(1))
- ProductComplementary [ ]
    - Product_ID (String)
    - Product_Name (String)
    - Code_EAN (String)
- ProductCharacteristics [ ]
    - Characteristic (String)
    - Value (String)
- InAssortment (String(1))
- IsLocal (String(1))
- Product Assortment
    - Store_ID (String)
    - Store_Name (String)
    - BestSeller (String)


```
    "TaxeRate":"20.00",
    "StartDate":"2016/01/15"
  },
  "TaxDEA":{
    "DEA_Code":"CodeDEA",
    "DEA_Amount":"2.2"
  },
 "DEEE_Amount":"0.8",
  "Product Category":{
    "Code_Famille":"ssalt",
    "Code_Sous_Famille":"CANAPE"
  },
  "Code_EAN":"10000000001",
  "Brand":"SANSMARQUE",
  "Product_Type":"I",
  "Grouped_Product":"Y",
  "Show_stock_screen":"Y",
  "Editable_Price":"Y",
  "Show_Characteristics_Description":"N",
  "ProductComplementary":[],
  "ProductCharacteristics":[
    {
      "Characteristic":"COLORIS_FABRICANT",
      "Value":"NOIR"
    },
    {
      "Characteristic":"HAUTEUR",
      "Value":"40"
    }
  ],
  "InAssortment":"Y",
  "IsLocal":"N",
  "ProductAssortment":{
    "Store_ID":"1234",
    "Store_Name":"StoreName_1234",
    "BestSeller":"N"
  }
```


SERVICES

Structure:

- Product_ID (String)
- Name (String)
- TaxCategory
    - TaxeCode (String)
    - TaxeName (String / null)
    - TaxeRate (String)
    - StartDate (String / null)
- Description[]
    - Description1 (String)
    - Description2 (String)
    - …
    - Description N (String)
- ProductCategory
    - Type_ID (String / null)
    - Name (String)
- Code_EAN (String)
- IsLocal (String(1) / null)

Example:


```
"Product_ID":"991230777724960",
  "Name":"PACK SIMP. RM BUFFET CUISINE111",
  "TaxCategory":{
     "TaxeCode":"TVA",
     "TaxeName":"TVA",
     "TaxeRate":"20.00",
     "StartDate":"2016/01/15"
  },
  "Description":[
     "description1",
     " description2",
     " description3",
     " description4"
  ],
  "ProductCategory":{
     "Type_ID":"S",
     "Name":"Assurance"
  },
  "Code_EAN":"EANSERVICEPACKCUISINE",
   "IsLocal":"N",
```

- **ProductAssortment (null)**
  - **Store_ID (String / null)**
  - **Store_Name (String / null)**
  - **BestSeller (String / null)**

```
"Product Assortment":{
      "Store_ID":"247",
      "Store_Name":"BUT AUXERRE",
      "BestSeller":"N"
   }
```

PRICES

Structure:

- **Product ID (String)**
- **IsLocal (String(1))**
- **Currency (String)**
- **PVE Price (String / null)**
- **Price Includes Tax (String(1))**
- **Price List [ ]**
  - **Store Name (String)**
  - **Store ID (String)**
  - **Store Price (String)**

Example:

```
"Product_ID":"29990461238467",
"IsLocal":"Y",
"Currency":"EUR",
"PVE_Price":"123",
"Price_includes_Tax":"Y",
"PriceList":[
  {
    "Name":"NOSICA ABIERTO",
    "Store_ID":"666",
    "Store_Price":"666.66"
  }
}
```

WAREHOUSES

Structure:

- **NosWareHouseID (String)**
- **Name (String)**
- **IsStoreDefaultWareHouse (String(1))**
- **IsActif (String(1) / null)**
- **Store**
  - **NosStoreID (String)**
  - **NosStoreName (String / null)**
- **WareHouseType**
  - **ID (String)**
  - **Label (String)**
- **WareHouseAddress**
  - **Adr1 (String)**
  - **Adr2 (String)**
  - **PostalCode (String)**
  - **City (String)**
  - **Country (String)**

Example:

```
"NosWareHouseID":"1894880101250",
"Name":"350 DEPOT SALON",
"IsStoreDefaultWareHouse":"N",
"IsActif":"Y",
"Store":
  {
    "NosStoreID":"350",
    "NosStoreName":"350 BUT STRASBOURG NORD"
  },
"WareHouseType":
  {
    "ID":"2",
    "Label":"Local"
  },
"WareHouseAddress":
  {
    "Adr1":"36 RTE DE STRASBOURG",
    "Adr2":"ZAC",
    "PostalCode":"67550",
    "City":"VENDENHEIM",
    "Country":"FRANCE"
  }
```

Para todas las entidades, las posibles respuestas de la plataforma Openbravo son las que se pueden encontrar en la siguiente tabla, aunque de todas, solo nos interesen las relacionadas con el formato de la información enviada al proceso EDL. Es decir, los errores F_001 (Required parameters are missing. (Attribute "errorField" indicates the missing input parameter).) y F_004 (Malformed request (eg. parameters containing invalid characters).)

Code Erreur	HTTP	Description
A_001	503	Service is under maintenance, please try again later.
A_002	503	Cannot connect to account database
A_003	400	API call (input parameter “request”) not specified, or unknown API call.
A_004	400	Unknown output format requested
F_001	400	Required parameters are missing. (Attribute “errorField” indicates the missing input parameter.)
F_002	400	Incorrect data type or format. (Attribute “errorField” indicates the invalid input parameter.)
F_003	400	Invalid value. (Attribute “errorField” indicates the field that contains an invalid value.)
F_004	400	Malformed request (eg. parameters containing invalid characters).
F_005	400	A parameter must have a unique value. (Attribute “errorField” indicates the invalid input parameter.)
F_006	400	The Parameter does not exists in our environment. (Attribute “errorField” indicates the invalid input parameter.)
F_007	400	Product Link Error (Product don’t exists) : ID_Product [ ]

En primer lugar trataremos los flujos de producto. Para producto, como para las demás entidades, se comprobará la estructura de la información. Es decir, el ERP deberá responder con el mensaje de error correspondiente cuando no respetamos la especificación. En los siguientes casos realizaremos envíos de productos a falta de algún campo. Después, analizaremos los casos que son más propios de la lógica del negocio. Veamos que casos existen para *Product*:

REQUEST	RESPONSE
Properly formed Product	{"status":"200","message":"Ok"}
Product without Product_ID attribute	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid characters).","errorField":"Product_ID"}
Product without Name attribute	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid characters).","errorField":"Name"}
Product without Description attribute	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid

	characters).","errorField":" Description "}
Product without TaxeCode attribute	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid characters).","errorField":" TaxeCode "}
Product without TaxeCode attribute	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid characters).","errorField":" TaxeCode "}
Product without TaxeName attribute	{"status":"200","message":"Ok"}
Product without TaxeRate attribute	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid characters).","errorField":" TaxeRate "}
Product without DEA_Code attribute	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid characters).","errorField":"DEA_Code"}
Product without DEA_Amount attribute	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid characters).","errorField":"DEA_Amount"}
Product without DEE_Amount attribute	{"status":"200","message":"Ok"}
Product without Code_Famille attribute	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid characters).","errorField":"Code_Famille"}
Product without Code_Sous_Famille attribute	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid characters).","errorField":"Code_Sous_Famille"}
Product without Code_EAN attribute	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid characters).","errorField":"Code_EAN"}
Product without Brand attribute	{"status":"200","message":"Ok"}
Product without Product_Type attribute	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid characters).","errorField":"Product_Type"}
Product without Grouped_Product attribute	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid characters).","errorField":"Grouped_Product"}
Product without Show_stock_screen attribute	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid characters).","errorField":"Show_stock_screen"}
Product without Editable_Price attribute	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid characters).","errorField":"Editable_Price"}
Product without Show_Characteristics_Description attribute	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid characters).","errorField":" Show_Characteristics_Description "}
Product without ProductComplementary attribute	{"status":"400","code":"F_001","message":"Required parameters are missing. (Attribute ?errorField? indicates the missing input parameter.) Les paramétres obligatoires sont absents","errorField":"ProductComplementary"}

Product without ProductCharacteristics attribute	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid characters).","errorField":"ProductCharacteristics"}
Product without InAssortment attribute	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid characters).","errorField":"InAssortment"}
Product without IsLocal attribute	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid characters).","errorField":"IsLocal"}
Product without Product Assortment attribute	{"status":"400","code":"F_001","message":"Required parameters are missing. (Attribute ?errorField? indicates the missing input parameter.) Les paramétres obligatoires sont absents","errorField":"Product Assortment"}
Product without Store_ID attribute (IsLocal=Y)	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid characters).","errorField":"Store_ID"}
Product without Store_ID attribute (IsLocal=N)	{"status":"200","message":"Ok"}
Product without Store_Name attribute (IsLocal=Y)	{"status":"200","message":"Ok"}
Product without Store_Name (IsLocal=N)	{"status":"200","message":"Ok"}
Product without BestSeller attribute (IsLocal=Y)	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid characters).","errorField":"BestSeller"}
Product without BestSeller attribute (IsLocal=N)	{"status":"200","message":"Ok"}

Como se ve en la tabla, los atributos generan distintas respuestas dependiendo del carácter de este. En los casos en los que recibimos el mensaje “OK” se debe al carácter no obligatorio del atributo. Las respuestas de tipo *Required parameters are missing* son generadas por atributos que deben estar presentes pero el producto se llega a insertar, aunque se deberá corregir el problema para su uso. Mientras que las respuestas de tipo *Malformed request* son generadas por atributos que obligatoriamente deben estar presentes aunque en algún caso pueda tener el valor null. En este caso el producto no llega ha insertarse.

Para la entidad *Services* tendremos los siguientes casos:

REQUEST	RESPONSE
Properly formed Service	{"status":"200","message":"Ok"}
Service without Product_ID attribute	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid characters).","errorField":"Product_ID"}
Service without Name attribute	{"status":"400","code":"F_004","message":"Malformed

	request (eg. parameters containing invalid characters).","errorField":"Name"}
Service without TaxeCode attribute	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid characters).","errorField":"TaxeCode"}
Service without TaxeName attribute	{"status":"200","message":"Ok"}
Service without TaxeRate attribute	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid characters).","errorField":"TaxeRate"}
Service without StartDate attribute	{"status":"200","message":"Ok"}
Service without ProductCategory/Type_ID attribute	{"status":"200","message":"Ok"}
Service without ProductCategory/Name attribute	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid characters).","errorField":"TaxeRate"}
Service without Code_EAN attribute	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid characters).","errorField":" Code_EAN"}
Service without IsLocal attribute	{"status":"200","message":"Ok"}
Service without ProductAssortment/Store_ID (IsLocal=N) attribute	{"status":"200","message":"Ok"}
Service without ProductAssortment/Store_ID (IsLocal=Y) attribute	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid characters).","errorField":"Store_ID"}
Service without ProductAssortment/Store_Name (IsLocal=N) attribute	{"status":"200","message":"Ok"}
Service without ProductAssortment/Store_Name (IsLocal=Y) attribute	{"status":"200","message":"Ok"}
Service without ProductAssortment/BestSeller (IsLocal=N) attribute	{"status":"200","message":"Ok"}
Service without ProductAssortment/BestSeller (IsLocal=Y) attribute	{"status":"200","message":"Ok"}

Para la entidad *Prices* tendremos los siguientes casos:

REQUEST	RESPONSE
Properly formed Price	{"status":"200","message":"Ok"}
Price without Product_ID attribute	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid

	characters).","errorField":"Product_ID”}
Price without IsLocal attribute	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid characters).","errorField":”IsLocal”}
Price without Currency attribute	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid characters).","errorField":”Currency”}
Price without PVE_Price attribute	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid characters).","errorField":”PVE_Price”}
Price without PriceList attribute	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid characters).","errorField":”PriceList”}
Price without Price_includes_Tax attribute	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid characters).","errorField":” Price_includes_Tax”}
Price without PriceList/Store_Name attribute	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid characters).","errorField":”Store_Name”}
Price without PriceList/Store_ID attribute	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid characters).","errorField":”Store_ID”}
Price without PriceList/Store_Price attribute	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid characters).","errorField":”Store_Price”}

Para la entidad *Warehouses* tendremos los siguientes casos:

REQUEST	RESPONSE
Properly formed Warehouse	{"status":"200","message":"Ok"}
Warehouse without NosWareHouseID attribute	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid characters).","errorField":”NosWareHouseID”}
Warehouse without Name attribute	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid characters).","errorField":”Name”}
Warehouse without IsStoreDefaultWarehouse attribute	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid characters).","errorField":”IsStoreDefaultWarehouse”}
Warehouse without IsBookable attribute	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid characters).","errorField":”IsBookable”}
Warehouse without IsSellable attribute	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid characters).","errorField":”IsSellable”}
Warehouse without IsReturnable	{"status":"400","code":"F_004","message":"Malformed

attribute	request (eg. parameters containing invalid characters).","errorField":”IsReturnable”}
Warehouse without IsActif attribute	{"status":"200","message":"Ok"}
Warehouse without NosStoreID attribute	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid characters).","errorField":”NosStoreID”}
Warehouse without NosStoreName attribute	{"status":"200","message":"Ok"}
Warehouse without WareHouseType/ID attribute	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid characters).","errorField":”ID”}
Warehouse without WareHouseType/Label attribute	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid characters).","errorField":”Label”}
Warehouse without WareHouseAddress/Adr1 attribute	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid characters).","errorField":”Adr1”}
Warehouse without WareHouseAddress/Adr2 attribute	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid characters).","errorField":”Adr2”}
Warehouse without WareHouseAddress/PostalCode attribute	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid characters).","errorField":”PostalCode”}
Warehouse without WareHouseAddress/City attribute	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid characters).","errorField":”City”}
Warehouse without WareHouseAddress/Country attribute	{"status":"400","code":"F_004","message":"Malformed request (eg. parameters containing invalid characters).","errorField":”Country”}

4. Implementación

En los siguientes apartado se nombrarán las tecnologías utilizadas para el desarrollo del *Fake Environment* además de los detalles de la implementación realizada. También se explicara el desarrollo de unos scripts para la instalación y configuración del entorno *Fake* en un entorno virtual Amazon EC2.

El inicio del proyecto comenzó con la implementación del caso *StockMD*. Para este se utilizo el *framework* NodeJS, que se trata de una librería y entorno de ejecución basado en Javascript. La librería nos proporciona una manera fácil para construir programas orientados a la arquitectura Cliente-Servidor. NodeJS dispone de muchos módulos, pero en este caso se han utilizado los siguientes:

1. Módulo Express: librería oriente al desarrollo de aplicaciones web. Además de ser de muy fácil manejo, es bastante potente.

2. Módulo HTTP: este es el módulo que nos permitirá trabajar con el protocolo HTTP. Nos servirá para crear un servidor HTTP que acepte solicitudes desde un cliente web.
3. Módulo URL: este módulo cuenta con funcionalidades para la resolución de URLs y parseo.

Utilizando el *framework* junto a estos tres módulos, la implementación de un servidor que atiende a una única URL y sin apenas lógica más que la de diferenciar entre dos tipos de peticiones y devolver una respuesta u otra se convierte en una tarea sencilla.

La implementación del servidor y el cliente se ha realizado en JAVA como se ha mencionado anteriormente debido a que los demás desarrollos también se han hecho en este lenguaje y por comodidad de uso del mismo. Como JDK (Java Development Kit) se ha utilizado OpenJDK 7, por seguir la política de uso de las herramientas de código libre que sigue la empresa. Se han utilizado las siguientes librerías:

1. Sun Net HTTPServer : para el uso de manejadores, cabeceras e intercambio de información mediante el protocolo HTTP.
2. Apache Commos Codec: para realizar la codificación en base 64 de la clave de autorización.
3. Apache Commos IO: para realizar el manejo de los *streams* de entrada y salida.
4. Codehaus Jettison Json: para el manejo de objetos con el formato JSON.

Se ha usado el entorno de desarrollo Eclipse debido al conjunto de herramientas que posee para la creación, edición y depuración del código en tiempo real.

Como se pretende desplegar el entorno en un entorno virtual Amazon EC2 para que los desarrolladores no tuvieran la necesidad de tener cada uno una instancia en local, se optó por crear dos scripts que permitieran, por un lado el arranque del servidor de manera automática y por otro, simplificar el arranque del cliente.

Para crear los scripts se ha utilizado la consola de comandos Bash de los sistemas Linux. Bash es un interprete que permite la ejecución de los scripts.

Para la ejecución automática de los scripts se ha utilizado el administrador de procesos Cron. Cron se utiliza para regular los procesos que se ejecutan en segundo plano en intervalos regulares. Se puede configurar en el fichero Crontab el proceso a ejecutar, así como la hora en la que debe hacerlo.

Finalmente, la herramienta Mercurial ha sido utilizado para el manejo del código y sus versiones mediante el repositorio Bitbucket. De esta forma, se ha podido desarrollar en local y actualizar la versión del entorno Amazon EC2 de una forma simple.

5. Instalación y configuración del entorno

En los siguientes apartados se van a detallar las principales características de la instalación y configuración de ambos entornos en maquinas virtuales de tipo AMAZON EC2. El escenario completo ha instalar y configurar se compone de dos partes.

La parte correspondiente a la plataforma Openbravo se utilizará para la realización de las pruebas de performance que comprueben el rendimiento de los desarrollos de integración realizados para BUT. En cuanto al *Fake Environment*, se utilizará como servidor que emule la respuesta del sistema de BUT. De esta forma el sistema ERP podrá ejecutar los procesos de integración y los desarrolladores podrán comprobar sus desarrollos.

A continuación en el diagrama, veremos la presencia de una tercera entidad que representara el cliente que se ejecuta en modo local para las pruebas de performance. Estas pruebas se realizarán usando el cliente que nos proporciona JMeter aunque hablaremos de él más adelante.

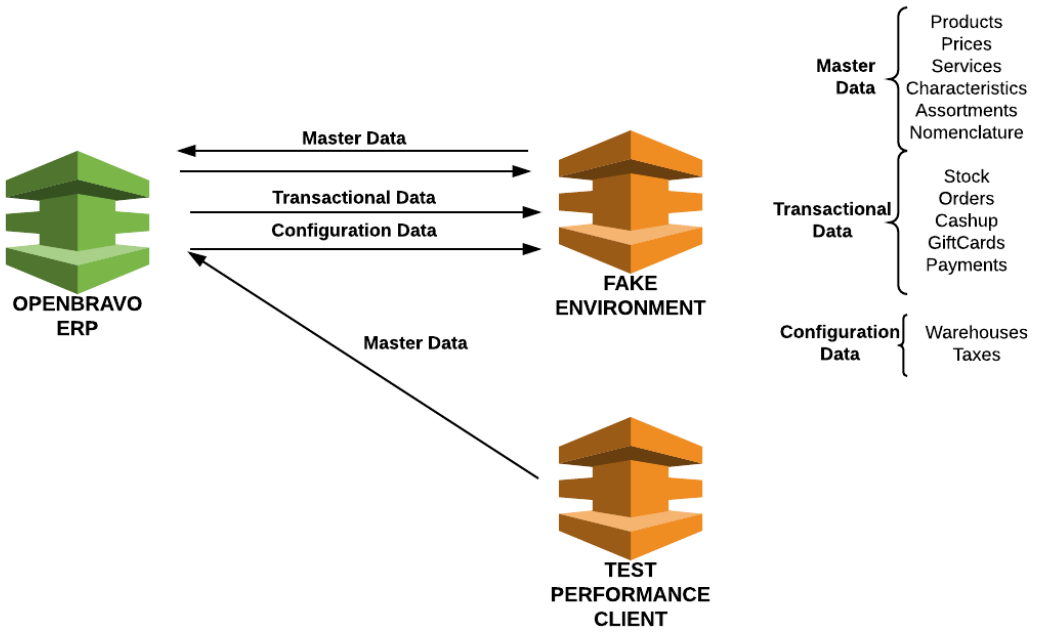

Figura 15: Diagrama de despliegue

INSTALACIÓN Y CONFIGURACIÓN: OPENBRAVO ERP

En una maquina virtual instalaremos una distribución de la plataforma Openbravo. Amazon EC2 (Elastic Compute Cloud) es un servicio ofrecido por la empresa Amazon.com que permite a los usuarios alquilar computadoras virtuales para la ejecución de aplicaciones

propias. Openbravo dispone de un contrato con Amazon.com para proveer de estos servicios a sus desarrolladores.

Lo primero para obtener una instancia EC2 es disponer de una cuenta en AWS (Amazon Web Services). Una vez dentro, seleccionamos la opción EC2 y en el panel de control presionamos lanzar nueva instancia de tipo Ubuntu Server 14.04 LTS. Al realizar el proceso se habrá creado una clave privada que podremos utilizar para acceder a la maquina virtual.

Una vez creada nuestra instancia, deberemos configurar el firewall para permitir la entrada y salida de los servicios webs que requiera la instancia de Openbravo. Pasemos a la instalación y configuración del ERP.

En primer lugar, nos copiaremos el código base del ERP y los módulos *custom* relacionados con la integración de BUT desde un repositorio a nuestra maquina virtual. Mediante la utilización de la herramienta Apache Ant, ejecutaremos la tarea *setup* para configurar la base de datos y a continuación la tarea install.source para la instalación del entorno.

Una vez finalizada la instalación, deberemos realizar la siguiente configuración para el correcto funcionamiento del proceso EDL.

1. Acceder al *backoffice* y buscar la venta *EDL Configuration*. Activar el proceso para las entidades *Product*, *Prices*, *Services* y *Warehouses* como se muestra en la siguiente imagen.

Figura 16: Ventana EDL Configuration

2. En la ventana *Preference* crear la preferencia *CreateIntWhenStoreNotExists* para que en caso de no existir la tienda insertada crear una nueva.

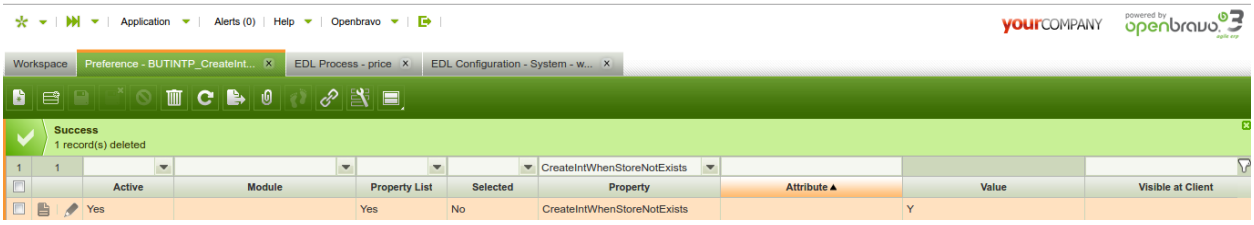

Figura 17: Ventana Preference

3. En la ventana *Role* deberemos de activar la característica *Is Web Service Enabled* al rol que queramos dar permisos para acceder al proceso EDL.

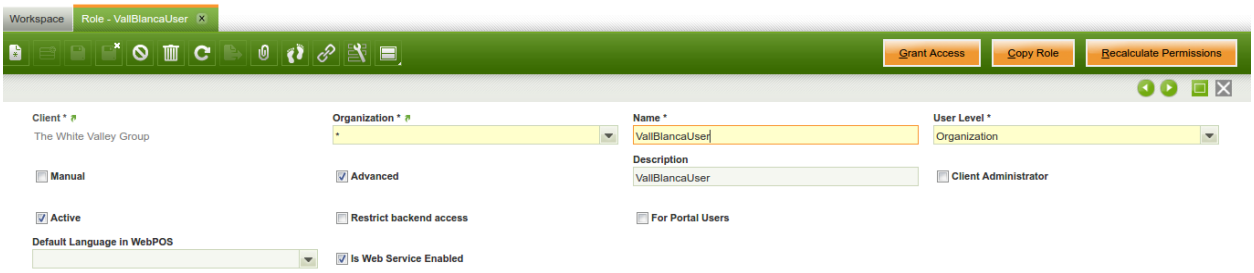

Figura 18: Ventana Role

4. En la ventana *Process request*, en caso de que no exista, deberemos configurar el proceso *Rebuild Product Category Tree* como se muestra en la siguiente imagen y lanzar el proceso mediante el botón *Schedule Process* una vez que este el *Fake Environment* desplegado, para que genere el árbol de categorías.

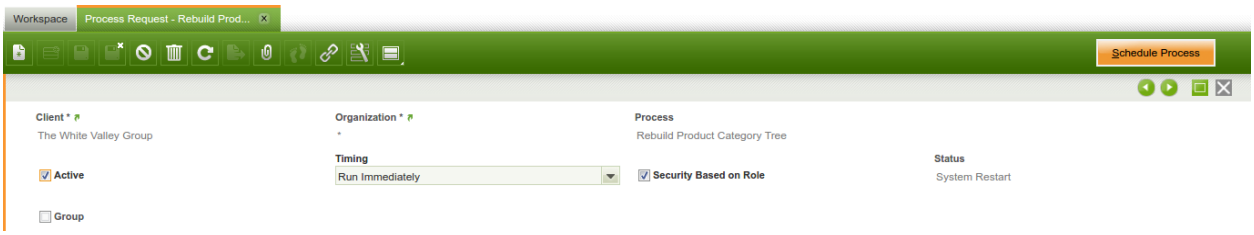

Figura 19: Ventana Process Request

Ya tenemos el entorno configurado para que el proceso EDL pueda estar a la escucha de peticiones de productos, servicios, precios e información de almacenes. Ahora nos falta desplegar la parte *Fake* para que el proceso *Rebuild Product Category Tree* pueda llevarse a cabo. Si este proceso no se llega a ejecutar antes de intentar importar cualquier producto, no se insertarán correctamente debido a la falta del árbol de la categoría de productos.

INSTALACIÓN Y CONFIGURACIÓN: FAKE ENVIRONMENT

Para el despliegue del *Fake Environment* se han seguido unos pasos iniciales muy similares. Necesitaremos crear una nueva instancia Amazon EC2 con el mismo sistema operativo para instalar el entorno. Una vez creada la instancia, deberemos clonar el paquete desde el repositorio de Bitbucket. En el paquete se incluye la siguiente estructura:

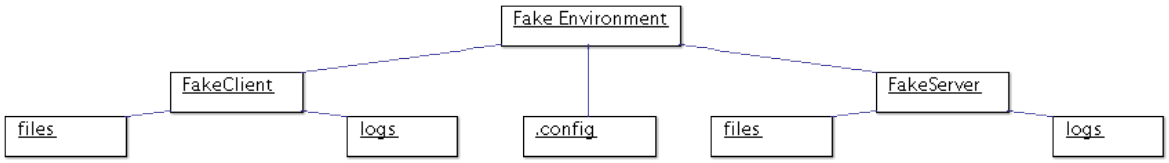

Figura 20: Estructura de carpetas

En la carpeta .config encontraremos los ficheros referentes a la configuración tanto del cliente como del server. En los directorios logs, encontraremos los ficheros relacionados con el registro que va creando la aplicación. En la carpeta *FakeClient* esta el ejecutable en formato JAR del cliente, así como en la carpeta *FakeServer* el del servidor. De esta forma la aplicación no fallará en caso de no tener alguna librería presente ya que el ejecutable evita este tipo de problemas.

Para el despliegue es necesario mantener esta estructura debido a la lectura de ficheros que realiza la aplicación. Los scripts de arranque están construidos para crear la estructura de carpetas requerida por la aplicación, por lo que si ejecutamos por primera vez la aplicación directamente sin script, no funcionará por falta de ficheros. Mientras que el script de finalización del servidor, solo finaliza con la ejecución de este.

Finalmente, configuraremos el demonio cron del entorno virtual mediante el fichero Crontab para que ejecute todos los días a las 8 de la mañana el script de arranque del servidor y a las 8 de la tarde en el parada (el fichero crontab se puede observar en el Anexo I). De esta forma, los desarrolladores dispondrán de un servidor que emule la respuesta de los sistemas de BUT sin tener que hacer nada más que configurar la plataforma Openbravo para que apunte al entorno de Amazon.

6. Test de performance del Fake Environment

En este apartado, se detallara el plan seguido para la realización de los test de performance contra el proceso EDL. El test de performance tiene como objetivo verificar que el proceso EDL funciona dentro de unos limites de tiempo de respuesta y procesamiento a la hora de realizar la integración de entidades como *Product*, *Prices*, *Services* y *Warehouses*, además de obtener resultados para entender en que parte del proceso se consume el tiempo. Al cliente BUT, le resulta interesante el rendimiento del proceso debido a la alta densidad de trafico que puede llegar a tener en un futuro, ya que realizarán cargas de grandes cantidades de datos en algunas ocasiones.

Para la realización de las pruebas se ha utilizado el software Apache JMeter. JMeter es una aplicación que nos permite realizar peticiones HTTP así como esnifar y reproducirlas, crear hilos de ejecución paralelos para testear la concurrencia, crear variables que nos permiten

variar la petición en base a la iteración, etc.. Para el caso nos interesa generar entidades nuevas con IDs o demás datos variables para así evitar que el ERP actualice entidades existentes en vez de añadir nuevas, ya que si intentamos añadir un producto existente el proceso EDL lo buscará en la DB y lo actualizará en vez de crear una nueva inserción, que es lo que nos interesa. En el caso de los precios, si realizamos la integración de uno cuyo producto no ha sido insertado anteriormente, el proceso lanzará un error debido a la falta de esta referencia.

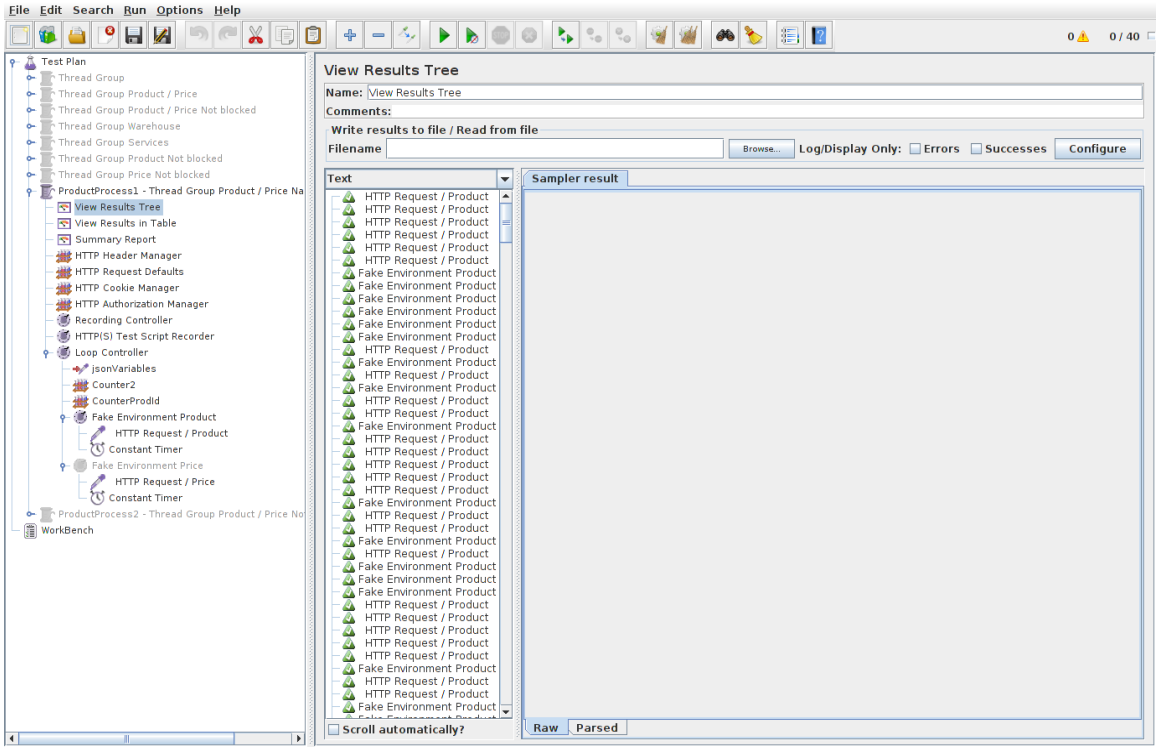

Figura 21: Captura de pantalla de JMeter

Además del uso de JMeter, se ha recurrido al registro de Tomcat y al seguimiento de la base de datos para obtener los resultados de la prueba. La plataforma ERP incluye la librería log4j de Apache que permite el uso de un *logger* para la depuración del código o para mostrar información relevante en el registro. El *logger* requiere de un fichero de configuración para su uso. El proceso EDL esta implementado en el módulo “com.openbravo.but.integration.master”. Este módulo contiene las clases que depuraremos, por lo que en el fichero de configuración de log4j debemos añadir los nombre de las clases que queramos debugear. Para obtener resultados, no se recurrió al log de Tomcat, solo se observaron los resultados de JMeter y de la base de datos. Las siguientes consultas nos permiten obtener el numero de inserciones por minuto de los últimos 3 minutos.

Product (tabla m_product):

```
select round(count(1)*1.00/180,2) as Ticket_per_second from m_product where
updated>=(now()- interval '3 min') and  updated<now() and producttype <> 'S';
```

Service (tabla m_product):

```
select round(count(1)*1.00/180,2) as Ticket_per_second from m_product where
updated>=(now()- interval '3 min') and  updated<now() and producttype = 'S';
```

Price (tabla m_pricelist):

```
select round(count(1)*1.00/180,2) as Ticket_per_second from m_pricelist where
updated>=(now()- interval '3 min');
```

Warehouse (misma tabla m_product):

```
select round(count(1)*1.00/180,2) as Ticket_per_second from m_warehouse where
updated>=(now()- interval '3 min');
```

Mediante estas pruebas básicas se observó que el rendimiento de los procesos *ImportPrice*, *ImportService* e *ImportWarehouse* era bueno, pero el relacionado con producto presentaba algún problema. El flujo de pruebas relacionado con el producto se compone de tres posibilidades. El primer proceso realiza inserciones de productos que presentan características nuevas. El segundo realiza inserciones de productos cuyas características estén presentes en el ERP. Finalmente el tercero realiza inserciones de manera similar al segundo a excepción de que existe una nueva característica con el valor del EAN en el producto. Los resultados son los siguientes:

c4.xlarge 300 IOPS with 1 thread executing a loop of each entity each time

Entity	Nº per second	Unitary execution	Insert Frequency
Warehouses	6.5	153.85 ms	153.85 ms
Services	6.6	151.52 ms	151.52 ms
Prices	5.7	175.44 ms	175.44 ms
Products process 1	0.8	1.25 s	1250 ms
Products process 2	4	250 ms	250 ms
Products process 2 with EAN as Characteristic	2.2	454.55 ms	454.55 ms

Para observar en más detalle que ocurría con el proceso *ImportProduct* se opto por la depuración de las clases *ImportProduct* y *CreateCharacteristics* mediante el *logger* log4j (véase Anexo II). Mediante el *logger*, se pretenden añadir instrucciones de depuración al código para mostrar los tiempos de ejecución de cada parte y así poder acotar el origen del problema. El método *importData*, es el método principal del proceso por lo que fue el primero en acotar (véase Anexo III).

De esta forma se concluyo que la parte que más tiempo consumía era la relacionada con las características, en concreto el método *getCharacteristicValue* de la clase *CreateCharacteristics*. Este método se llama desde un bucle del método *initCharacteristics*, por lo que se midió el tiempo requerido por cada iteración además del coste de cada parte de *CreateCharacteristics*. Los métodos quedaron de la siguiente forma:

```
public static void initCharacteristics(List<ProductCharacteristicsBUT> productCharacteristicsBut) {
   long t0,t1;
   log.debug("~~~~~~~~~~~~~~~ INIT CHARACTERISTICS ~~~~~~~~~~~~~~~~");
   for (ProductCharacteristicsBUT productCharacteristicBut : productCharacteristicsBut) {
         t0 = System.currentTimeMillis();
    getCharacteristicValue(productCharacteristicBut.getCharacteristic(),
         productCharacteristicBut.getValue(),t0);
    t1 = System.currentTimeMillis();
    log.debug(" ## Product Characteristic {}, Value: {} ",productCharacteristicBut.getCharacteristic(),
productCharacteristicBut.getValue());
    log.debug("Iteration time: {} ms", t1-t0);
   }
 }


private static CharacteristicValue getCharacteristicValue(String characteristicName, String value, long
t00) {
   log.debug("~~~~~~~~~~ Get Characterictic Value ~~~~~~~~~~~~");
   long t0 = System.currentTimeMillis();
   log.debug("## t0 difference: {}",t0-t00);
   Characteristic characteristic = searchCharacteristic(characteristicName);
   long t1 = System.currentTimeMillis();
   if (characteristic == null) {
     characteristic = createCharacteristic(characteristicName);
     return createCharacteristicValue(value, characteristic);
   }
   long t2 = System.currentTimeMillis();

   CharacteristicValue charValue = searchCharacteristicValue(value, characteristic);
   long t3 = System.currentTimeMillis();
   if (charValue == null) {
     charValue = createCharacteristicValue(value, characteristic);
   }
   long t4 = System.currentTimeMillis();
   log.debug(" getCharacteristicValue (" + characteristicName + ") --- search: " + (t1 - t0) + " ms,
create: " + (t2 - t1) +
   " ms, search_ch_val: " + (t3 - t2) + " ms, create ch val: " + (t4 - t3) + " ms");
   return charValue;
 }
```

Los resultados obtenidos, que se muestran a continuación, fueron bastante sorprendentes. Para sorpresa, el coste de ejecución total del método *getCharacteristicValue* no coincide con la suma de cada parte.

```
20:07:47,123 Creating characteristic: 530710 name
20:07:47,203  ## Product Characteristic 530710, Value: 530711
20:07:47,203 Iteration time: 1218 ms
20:07:53,230 ~~~~~~~~~~ Get Characterictic Value ~~~~~~~~~~~~
20:07:53,230 ## t0 difference: 0
20:07:53,232  getCharacteristicValue (530710) - search: 1 ms, create: 0 ms, search_ch_val: 1 ms,
create ch val: 0 ms
20:07:53,234 ~~~~~~~~~~ Get Characterictic Value ~~~~~~~~~~~~
```

En el caso de esta característica, podemos observar que el coste total es de 1218ms mientras que la suma de las partes da como resultado 1ms+0ms+1ms+0ms=2ms, una diferencia de alrededor de 1 segundo. En primer lugar, se comprobó el estado de la memoria, así como el de las *CPU*-s y base de datos y no se observó ninguna evidencia del problema. La memoria no llegaba a llenarse, ni los procesadores a saturarse y la base de datos no está formando ningún tipo de cuello de botella. También se analizo el comportamiento del recolector de basura y tampoco se observo nada extraño.

Todas estas observaciones se realizaron mediante el software Munin que permite una monitorización extensa de la aplicación. Los gráficos que se muestran a continuación reflejan lo comentado.

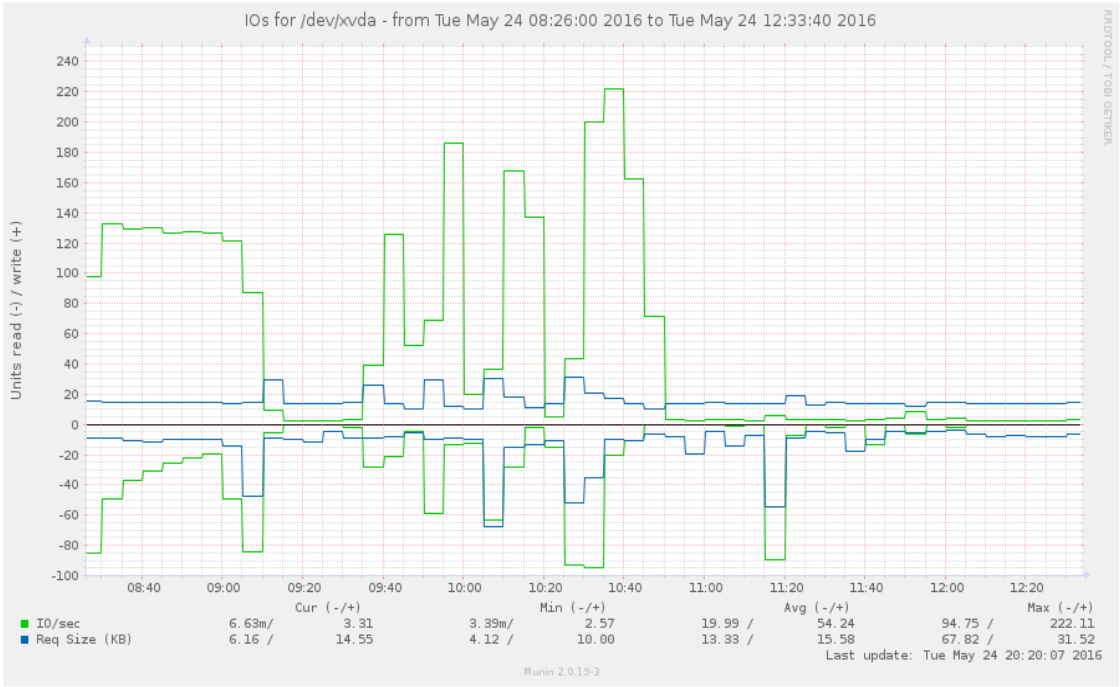

Figura 22: Monitorización del uso de memoria

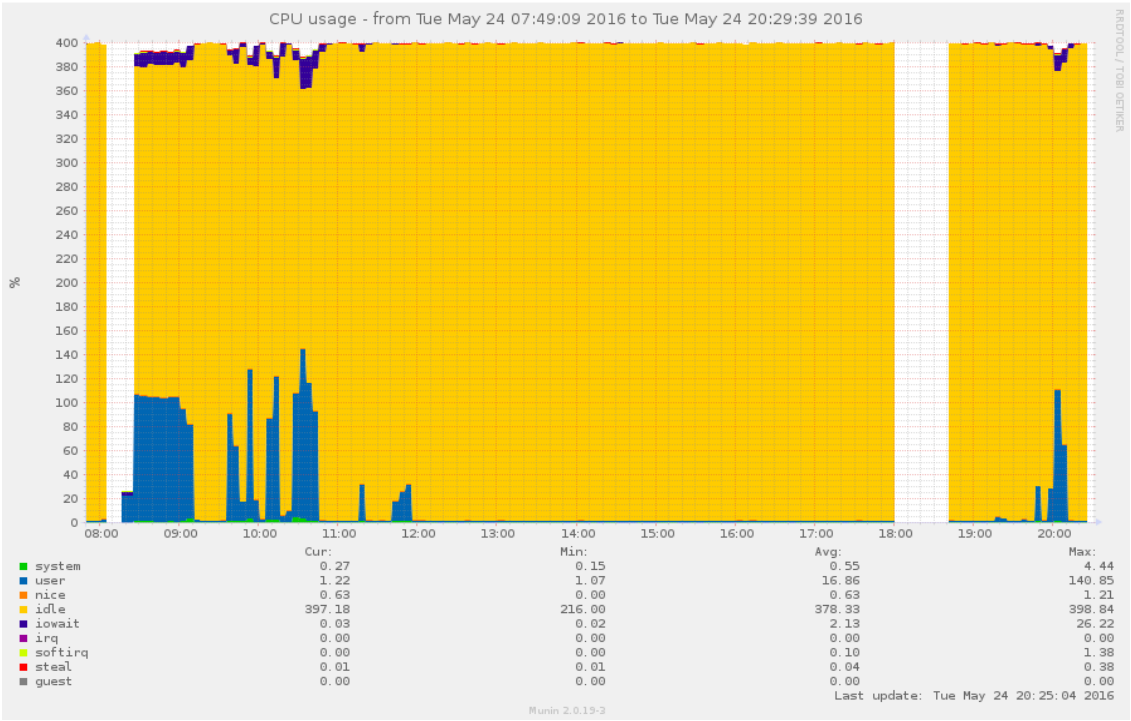

Figura 23: Monitorización del uso de cpu

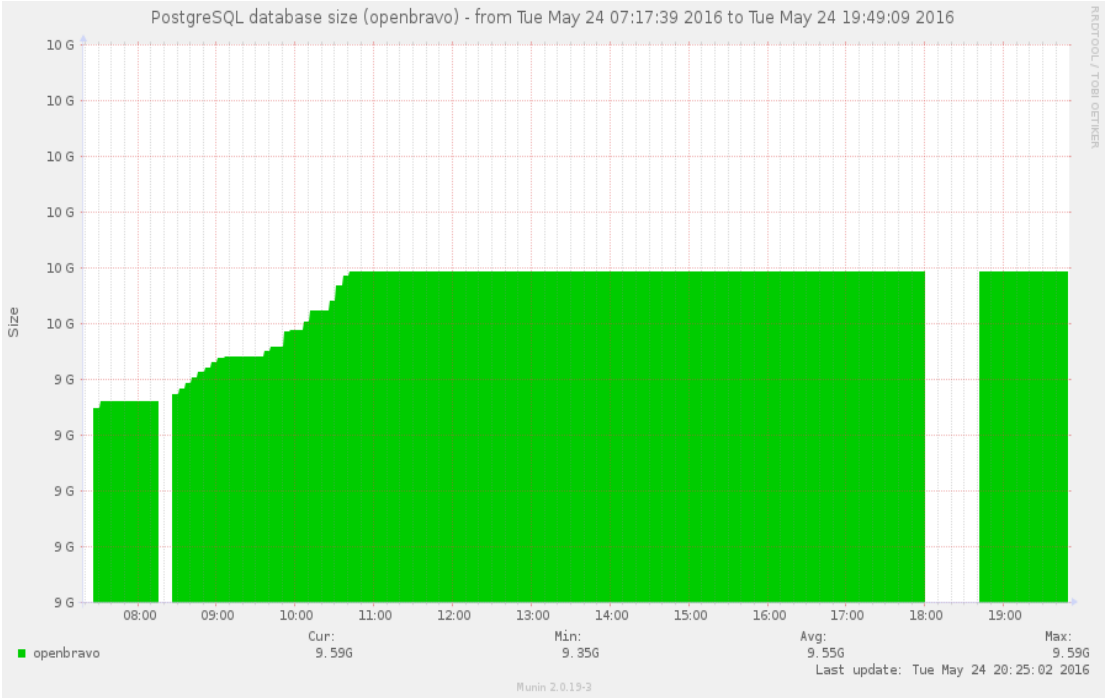

Figura 24: Monitorización del uso de base de datos

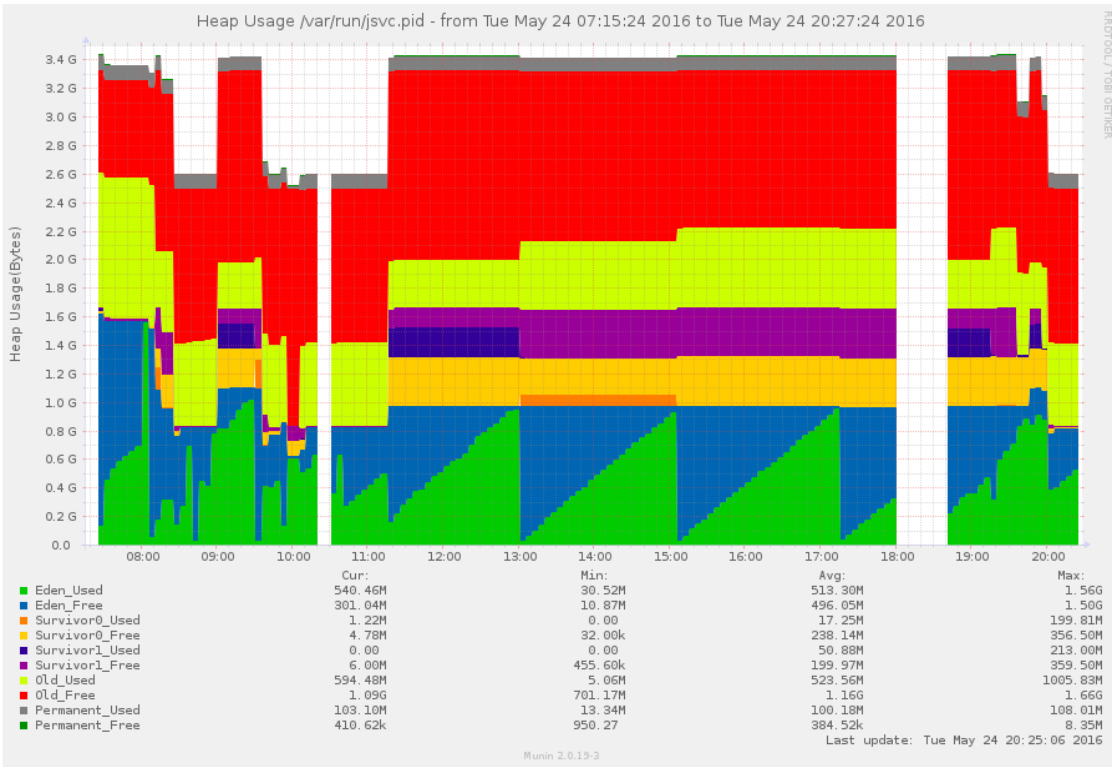

Figura 25: Monitorización del garbage colector de JAVA

Como se observa en las pruebas, no hay ningún pico que nos de a entender que la maquina tenga un problema de recursos. Por fin, se ha conseguido determinar la causa o procedencia de la perdida del tiempo de ejecución en el método *CreateCharacteristics*. Por lo visto, el problema se generaba por de culpa de un cuello en los IOPS (Input output per second) del disco que ofrecen las maquinas virtuales de Amazon. El problema se soluciona contratando un mayor disco, por que Amazon limita los IOPS en base al tamaño del disco.

7. Conclusiones

En primer lugar, considero que la relación directa del trabajo fin de grado con el mundo laboral hace que éste se plantee como un producto de mercado por lo que adquiere mayor seriedad. Creo que al desarrollarlo en un entorno profesional se ven reflejadas las buenas practicas que se siguen en la empresa.

En mi opinión, la realización de las prácticas curriculares junto al trabajo de fin de grado permite al estudiante ser guiado en mayor medida que si lo tuviera que realizar de manera individual. El entorno de empresa presiona en cuanto a cumplir fechas y no entorpecer el trabajo a los demás. ese aspecto actúa de forma favorable para ayudar a mantener una constancia con el proyecto.

En concreto, el *Fake environment* formará parte de la ejecución de los test automáticos de Openbravo. No se pueden realizar test que dependan de terceros sistemas, ya que en caso de no tenerlos disponibles, el test no funcionaría. Por ello, el entorno *Fake* evitará el problema y permitirá el testeo de los servicios web.

El trabajo en equipo es un aspecto muy importante. Motiva mucho al personal ha realizar tareas, ya que se consigue una buena organización y división de las tareas. Trabajar de esta forma es muy cómodo, solo te tienes que preocupar de la tarea encomendada y no de tareas pequeñas y alguna importante como suele ocurrir de manera individual.

También comentar que como desventaja del mundo laboral está el cambio de requisitos continuo. Debido a que el TFG forma parte de un proyecto de poca madurez y todavía no se han determinado del todo muchos aspectos de las integraciones, los cambios ocurren constantemente. Esto no es muy favorable para realizar la documentación del proyecto ya que está en constante cambio.

En último lugar, me gustaría agradecer tanto a la UPNA como a Openbravo la ocasión de poder realizar el TFG, como las prácticas curriculares. Muchas gracias también a todos los profesores, tutores o compañeros que me han ayudado en todo este proceso de aprendizaje y en especial a mi tutor de TFG, Fitxi (Federico Fariña Figueredo), por su apoyo constante en la carrera y por demostrar que existen profesores de verdad y no señores/señoras con muchos conocimientos incapaces de guiar al alumnado.

8. Anexos

1. Fichero de configuración del demonio Cron, crontab

```
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
15 08 * * * /opt/FakeEnvironment/launchServer.sh
58 19 * * * /opt/FakeEnvironment/killServer.sh
```

2. Fichero de configuración de log4j, log4j.lcf

```
# ***********************************************************************
# * The contents of this file are subject to the Openbravo  Public  License
# * Version  1.1  (the  "License"),  being   the  Mozilla   Public  License
# * Version 1.1  with a permitted attribution clause; you may not  use this
# * file except in compliance with the License. You  may  obtain  a copy of
# * the License at http://www.openbravo.com/legal/license.html
# * Software distributed under the License  is  distributed  on  an "AS IS"
# * basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. See the
# * License for the specific  language  governing  rights  and  limitations
# * under the License.
# * The Original Code is Openbravo ERP.
# * The Initial Developer of the Original Code is Openbravo SLU
# * All portions are Copyright (C) 2007-2013 Openbravo SLU
# * All Rights Reserved.
# * Contributor(s):  _____.
# ***********************************************************************

# to enable logging in i.e. eclipse console add ", C" to the line below
log4j.rootCategory=INFO, R

# Set our global levels
log4j.category.org=WARN
log4j.category.org.openbravo=INFO

log4j.appender.R=org.apache.log4j.RollingFileAppender
```

```
log4j.appender.R.File=${catalina.base}/logs/openbravo.log
log4j.appender.R.MaxFileSize=10000KB

# R uses PatternLayout.
log4j.appender.R.layout=org.apache.log4j.PatternLayout
log4j.appender.R.layout.ConversionPattern=%d{ISO8601} [%t] %-5p %c - %m%n

# Keep one backup file
log4j.appender.R.MaxBackupIndex=1

# C is an optional ConsoleAppender for usage in i.e. Eclipse
log4j.appender.C=org.apache.log4j.ConsoleAppender
log4j.appender.C.layout=org.apache.log4j.PatternLayout
log4j.appender.C.layout.ConversionPattern=%-4r [%t] %-5p %c - %m%n

log4j.category.reloadXml=ERROR

# To debug an specific class
#log4j.category.org.openbravo.erpCommon.ad_process=DEBUG
log4j.category.com.openbravo.but.integration.master.process.ImportProduct=DEBUG
log4j.category.com.openbravo.but.integration.master.process.CreateCharacteristics=DEBUG

#*************************************************
# Hibernate
#*************************************************
log4j.appender.HB=org.apache.log4j.RollingFileAppender
log4j.appender.HB.File=${catalina.base}/logs/openbravo_hibernate.log
log4j.appender.HB.MaxFileSize=10000KB

# R uses PatternLayout.
log4j.appender.HB.layout=org.apache.log4j.PatternLayout
log4j.appender.HB.layout.ConversionPattern=%d{ISO8601} [%t] %-5p %c - %m%n

# Keep one backup file
log4j.appender.HB.MaxBackupIndex=1

log4j.logger.org.hibernate=error, HB

### log HQL query parser activity
#log4j.logger.org.hibernate.hql.ast.AST=error, HB
```

3. Método importData de la clase Java ImportProduct.java

```
public static List<String> importData(String json) throws JSONException {

   try {
     OBContext.setAdminMode(true);
     ProductBUT productBUT = new ProductBUT(json);
     //log.debug("~~~~~~~~~~ PRODUCT IMPORT DATA ~~~~~~~~~~~~~~");
     //log.debug("{} Start product process: {} ID", new Date(), productBUT.getProductId());
     boolean createPref = IntegrationUtils.getPreferenceNosicaStore();
     boolean disable = false;
     long t0 = System.currentTimeMillis();
     Organization store = null;
     if (productBUT.isLocal() && productBUT.getOptProductAssortment() != null) {
       store = MasterUtils.getStore(productBUT.getOptProductAssortment().getId());
     }
     long t01 = System.currentTimeMillis();
     //log.debug(" --- needed masterdata checked: --- store {} ms", t01 - t0);

     // Touch system client id to avoid issue in OBInterceptor.
     // TODO: Remove when issue 32858 is resolved and included.
```

```
      OBDal.getInstance().get(Organization.class, "0").getClient().getId();

      /**
       * The product is not created and the OK message is returned.
       *
       * The product has to be local, store not found and the property has to be set to false.
       */
      if (productBUT.isLocal()) {
        if (store == null && !createPref) {
          return Collections.emptyList();
        }
        // Local products have to be created in their store organization. If this store does not
        // exists it has to be updated to isActive false
        if (store == null) {
          store = IntegrationUtils.getDefaultOrgProxy();
          disable = true;
        }

        // If assortment is not created do it
        if (productBUT.getInAssortment() && productBUT.getOptProductAssortment() != null) {
          String nosicaStoreID = productBUT.getOptProductAssortment().getId();
          CreateAssortment.initializeAssortment(store, nosicaStoreID, !disable);
        }
      }
      long t02 = System.currentTimeMillis();
      //log.debug(" --- needed masterdata checked: --- product {} ms", t02 - t01);
      // Check and create needed master data to properly insert the product. In case it is needed
to
      // create any it is done in synchronized methods to avoid concurrency issues adding the same
      // entity.
      checkProductCategoryRebuild(productBUT.getProductCategory());
      if (store != null) {
        store = OBDal.getInstance().get(Organization.class, store.getId());
      }
      //log.debug(" --- product category checked {} ID", productBUT.getProductId());
      long t03 = System.currentTimeMillis();
      //log.debug(" --- needed masterdata checked: --- productCategoryRebuild {} ms", t03 - t02);

      TaxCategory taxCategory = CreateTaxCategory.getTaxCategory(productBUT.getTaxCategory());
      //log.debug(" --- taxcategory checked {} ID", productBUT.getProductId());
      long t031 = System.currentTimeMillis();
      //log.debug(" --- needed masterdata checked: --- taxCategory {} ms", t031 - t03);
      CreateCharacteristics.initCharacteristics(productBUT.getProductCharacteristicsList());
      //log.debug(" --- characteristics checked {}", productBUT.getProductId());
      long t04 = System.currentTimeMillis();
      //log.debug(" --- needed masterdata checked: --- characteristics {} ms", t04 - t031);

      phieco_taxcategory dea = null;
      if (productBUT.getOptTaxDea() != null) {
        dea = getDEA(productBUT.getOptTaxDea());
        if (dea == null) {
          dea = createDEA(productBUT.getOptTaxDea());
        }
      }
```

9. Bibliografia

- *Información sobre la empresa Openbravo y su producto.*
  http://www.openbravo.com/

- Información sobre la empresa BUT.
  https://fr.wikipedia.org/wiki/But_%28entreprise%29

- *Wiki Openbravo. Información general sobre la aplicación, su utilización y funcionamiento.*
  https://es.wikipedia.org/wiki/Openbravo

- *Información sobre los requisitos del sistema Openbravo ERP.*
  http://wiki.openbravo.com/wiki/System_Requirements

- *Información sobre la distribución 3.0PR16Q2*
  http://wiki.openbravo.com/wiki/Release_Notes/3.0PR16Q2

- *Documentación del framework NodeJS.*
  https://nodejs.org/en/

- *Documentación del software Apache JMeter.*
  http://jmeter.apache.org/

- *Documentación de la aplicación de monitorización de performance Munin.*
  http://munin-monitoring.org/wiki/Documentation

- *API de JAVA*
  https://docs.oracle.com/javase/7/docs/api/

- *Documentación sobre el OpenJDK, JDK (Java Development Kit) de código abierto.*
  http://www.openjdk.java.net

- *Manual del gestor de procesos de segundo plano Cront.*
  http://linux.die.net/man/1/crontab

- *Documentación sobre los entornos virtuales Amazon EC2.* https://aws.amazon.com/es/ec2/

- *Documentación de la herramienta cliente para el control de versiones Mercurial.* https://www.mercurial-scm.org/

- *Documentación del entorno de desarrollo Eclipse IDE.* http://www.eclipse.org/documentation/?sess=32267642b71e61a5209ee01bb85d8fdb