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

Automatización y validación de la actualización periódica de los datos de muestra de Openbravo ERP

Grado en Ingeniería
en Tecnologías de Telecomunicación

Trabajo Fin de Grado

Carlos Sousa Navó

Jose Javier Astrain Escola

Pamplona

Indice

Resumen y palabras clave

Resumen

Este trabajo fin de grado se centra principalmente en el ERP de Openbravo, más concretamente, trata de añadir una funcionalidad nueva a esta herramienta para poder automatizar una tarea que resulta muy necesaria para la empresa.

El objetivo de esta nueva función es actualizar los datos de Openbravo ERP cada trimestre y validar los nuevos flujos funcionales que se han creado con la inserción de esta información. La realización de este trabajo le ahorrará tiempo a la empresa y le ayudará a tener una herramienta nueva y una mejor infraestructura para poder mejorar la aplicación web.

Para poder realizar esta tarea he tenido que instalar un entorno de Openbravo ERP para poder trabajar con la aplicación web y así aprender cómo funciona. Además, para poder programar de una forma correcta tuve que ver programas escritos por los trabajadores y analizar su funcionamiento.

Gracias a la infraestructura de Selenium que tiene Openbravo y a otras herramientas que se ilustraran en este trabajo, he podido automatizar una serie de flujos funcionales para que se ejecuten cada trimestre actualizando y validando los datos del ERP.

Palabras Clave

- Openbravo

- ERP

- Automatización

- Selenium

- Java

- Aplicación Web

-Mercurial

1.Introducción

En este apartado se describirá brevemente a la empresa, la motivación que me impulsó a realizar este proyecto, los antecedentes del trabajo, así como la propuesta y los objetivos de este.

1.1 Origen y evolución de la empresa

"Tecnicia" fue el primer nombre que tuvo la empresa al ser fundada en 2001 por Nicolás Serrano, Ismael Ciordia y Moncho Aguinaga. Los tres trabajaron en la Universidad de Navarra.

Más tarde, en 2006, Manel Sarasa y Josep Mitjá se unieron a la empresa como *Chief Executive Officer* (CEO) y *Chief Operating Officer* (COO), respectivamente. En el mismo año la compañía cambió su nombre por el de Openbravo. Su producto estrella fue lanzado y publicado con el nombre de OpenbravoERP. Como Openbravo quería que su aplicación fuera de código libre utilizó Compire, una empresa con un ERP de código libre y licencias públicas, como base para desarrollar su propio ERP. Así pues, durante el año 2006 publicaron el código de la Aplicación.

En el año 2007 comenzaron a trabajar en un nuevo producto para dar solución a los pequeños comercios, denominado Openbravo *point of sale* (POS).

En 2008 Openbravo se convirtió en una de las empresas líderes de código libre gracias a la colaboración y financiación de tres grandes inversores, Amadeus, GIMV y Adara que donaron la cantidad de 12.5 millones de dólares.

En julio de 2012 se lanza Openbravo Web POS para minoristas, esta solución supuso un cambio en el anterior POS ya que este se trataba de un producto Web mucho más adaptativo y flexible que Openbravo Java POS.

En 2015 tanto la Plataforma de Comercio y la Plataforma ERP son renombradas a la Suite de Comercio y la Suite de Negocio. En este momento Openbravo cambió el objetivo y se centró en impulsar su producto para minoristas.

Openbravo es una empresa muy grande con oficinas en Barcelona (España), Pamplona (España), París (Francia), Querétaro (México) y Kolkata (India), aunque también tiene empleados trabajando por otros lugares del mundo como EEUU y Cuba.

La oficina de Pamplona es la sede principal, en ella trabajan muchos empleados para el desarrollo y mantenimiento del software. Openbravo, al tener un producto tan grande, necesita agrupar a los trabajadores en equipos para que cada uno de estos equipos se encargue de una tarea distinta y así abarcar mejor los proyectos.

Dentro de la oficina se encuentran el equipo de plataforma, que se encarga de mantener la herramienta de Openbravo ERP abstrayéndose de la funcionalidad, el equipo de retail, que se ocupa de Openbravo web POS; el equipo funcional, cuya tarea es la de aportar la funcionalidad del ERP de Openbravo, es decir, utilizar la herramienta supervisada por plataforma para generar código, QA, que una vez que funcional genera el código, estos, se encargan de confirmar que es estable para meterlo en el repositorio principal. Además, también se ocupa de la realización y mantenimiento de los tests automáticos, cloud, que se encarga de mantener los servidores de Amazon en perfectas condiciones y además optimizar los problemas de performance. Por último, el equipo de BUT, este último es un equipo especial formado por integrantes de los anteriores grupos centrados en el desarrollo de la aplicación para el cliente BUT.

1.2 Motivación

Este trabajo me llamó mucho la atención ya que me pareció una buena idea aportar algo de mí a un producto tan grande como el ERP de Openbravo. Ellos necesitaban añadir una funcionalidad al producto que hasta ahora no tenían y me pareció una gran

oportunidad para adentrarme un poco más en el mundo laboral y aprender cómo funciona una empresa.

Openbravo tiene muchos premios, como el premio Info World` BOSSIE (*Best of Open Source Software awards*) por ser una de las empresas más importantes en la utilización de código libre comercial en el mundo. Este fue uno de los aspectos más importantes que me impulsó a unirme a esta entidad. Sabía que iba a poder estar con profesionales en este ámbito e iba a poder aprender de ellos cómo desarrollar una aplicación de forma correcta. Era una ocasión que no podía dejar pasar.

1.3. Antecedentes

Openbravo ERP es un software de gestión integrado y consiste en una serie de módulos conectados a una base de datos para administrar los procesos operativos de la empresa, agrupando las distintas funciones de gestión en un único sistema. De esta forma permite a los clientes de la empresa dar tiempos rápidos de respuesta a los problemas, así como un manejo de información muy eficiente para minimizar costes.

Para que un consumidor utilice esta herramienta y pueda realizar esta serie de acciones es necesario que se introduzca la información en la aplicación web.

Esta información se guarda en una base de datos a la que la aplicación le hará peticiones para recuperarla. Los datos que se introducen sirven para recrear flujos funcionales dentro de OpenbravoERP como un movimiento de productos o una factura de compra.

Estos flujos funcionales recrean las acciones que realizaría un usuario de la aplicación para gestionar su empresa.

El equipo funcional de Openbravo se encarga de mantener actualizada la aplicación web corrigiendo algún error que hayan localizado, añadiendo nuevas herramientas y renovando los datos que se introducen en la aplicación para poder

realizar flujos funcionales, como puede ser el movimiento de un producto desde su salida del almacén hasta su entrega en el punto de recogida.

Pues bien, estos datos son sumamente importantes y necesitan ser actualizados cada trimestre para que cuando un cliente o un trabajador de la empresa lo vaya a utilizar pueda tener una experiencia adecuada con la herramienta.

De esta tarea se suele encargar una persona del equipo funcional en la que invierte alrededor de una semana. El problema es que se trata de un proceso que se debe de repetir cada tres meses y supone una pérdida de tiempo.

Para no preocuparse de realizar la actualización de datos cada tres meses se planeó automatizar esta tarea.

En Openbravo hay dos formas de automatizar procesos, la primera es utilizando JUnit que resumidamente, es un Framework que se encarga de ejecutar clases de Java de manera controlada para evaluar si realmente los métodos de esas clases se están comportando como debieran. La segunda forma con la que Openbravo realiza test automáticos es utilizando Selenium.

Gracias a Selenium se consigue dar órdenes al navegador para que realice una serie de acciones, lo que resulta perfecto porque de esta forma no solo se introducen los datos dentro de la aplicación, sino que además se simula que un usuario lo está haciendo manualmente y así podemos comprobar que el flujo funcional actúa correctamente y no hay ningún *bug*.

Openbravo tiene una gran cantidad de tests automáticos para probar que los errores que se han producido en la aplicación han sido resueltos. Estos tests se lanzan todos los días de forma automática y validan los nuevos parches que se introducen en la herramienta.

Antes de sacar una versión de la aplicación primero un miembro del equipo funcional verifica que funcionalmente el resultado es el esperado, a

continuación, se ejecutan esta serie de tests para validar el producto final. Dentro de este proceso de integración y validación se utiliza Jenkins para tener un control mucho más exhaustivo a la hora de detectar en qué lugar se localiza el error. Jenkins es un servidor de integración continua y de código libre, esta herramienta sirve, junto a la herramienta de control de versiones Mercurial, para que cuando un miembro del equipo suba algo al repositorio central, se lancen estos tests de forma automática y así poder tener siempre la aplicación vigilada.

De esta forma se puede garantizar que el producto final es estable y cumple todas las especificaciones descritas por los desarrolladores.

1.4 Propuesta

Desde un principio, la propuesta de proyecto que ofertaba la empresa era automatizar el proceso de inserción periódica de datos del ERP de Openbravo.

Como se ha visto en el apartado anterior, Openbravo, actualiza la información manualmente y después verifica la funcionalidad del producto y lo válida para poder cerrar el producto final.

Este proceso, al ser tan costoso como imprescindible y al realizarse en cada nueva actualización, es conveniente automatizarlo para no tener que repetirlo en cada nueva actualización del ERP.

En primer lugar, para poder realizar esta tarea de automatización periódica de los datos del ERP he tenido que aprender de forma autónoma una serie de conceptos sobre cómo funciona el ERP de Openbravo con la ayuda de una serie de cursos de formación que ofrece la empresa. En estos cursos he aprendido cómo generar flujos funcionales de inicio a final. Para poder automatizar estos flujos primero es necesario saber cómo reproducirlos a mano ya que es necesario programarlos después con Selenium.

Una vez concluidos los cursos formativos, fue importante aprender cómo trabajaban en la empresa con Mercurial, cómo se organizaban a la hora de resolver problemas y la hora asignar las tareas.

En Openbravo utilizan Mercurial como herramienta para el control de versiones. Ellos tienen un repositorio central que se llama PI. En PI se encuentra la versión final de OpenbravoERP. Sin embargo, el equipo funcional desarrolla primero sobre otro repositorio, que es una copia actualizada de PI, este repositorio se llama PI-Funcional, de esta forma se aseguran de que no se introduce ningún error en PI.

Por último, tuve que aprender a manejar Selenium. Para ello durante una semana estuve viendo cómo funcionaban los tests automáticos que ya tenía Openbravo para otros flujos funicionales y también automatizando algún test para validar parches en issues.

Utilizando estas herramientas se trata de dar una solución a la propuesta creada por la empresa. Para eso después de haber analizado la situación y tras aprender cómo utilizar esta serie de tecnologías, me propongo cumplir con las necesidades de la empresa y añadir esta nueva funcionalidad e integrarla dentro de OpenbravoERP.

De esta forma, mi aportación en la empresa y al producto es de vital importancia ya que les ayudo a realizar una tarea que tarde o temprano tendrían que desarrollar, como también a ampliar y completar la infraestructura de Selenium para poder utilizar estas funciones en un futuro. No solo esto, sino que también provoca un ahorro de tiempo muy valioso a la empresa que puede traducirse en dinero.

1.5 Objetivos del Proyecto

Uno de los objetivos principales de este proyecto es cubrir las necesidades de la empresa acerca de la automatización del proceso de inserción de datos en

Openbravo ERP, como a su vez la validación del producto final para que la aplicación web pueda ser actualizada cada trimestre y así los trabajadores de Openbravo puedan dedicarse a solventar otros problemas.

Además de esto, Openbravo va a poder contar con la actualización y ampliación de su framework de Selenium, ya que para el desarrollo de este trabajo se han tenido que mejorar y aportar nuevas herramientas de trabajo que podrían utilizar si lo necesitaran.

He podido aprender acerca de temas que no se enseñan en la carrera y que en mi opinión complementan muy bien el conocimiento adquirido en el grado de telecomunicaciones. Este trabajo, al estar más enfocado en la carrera de informática, me ha ayudado a conocer en mayor profundidad temas como las relaciones en bases de datos, la programación orientada objetos, el modelo vista controlador, la utilización de un control de versiones para el seguimiento del proyecto...

El aprendizaje de estos conocimientos me parece un objetivo primordial de este trabajo, ya que en la realización de un proyecto fin de grado no solo se tiene que poner en práctica lo aprendido en años anteriores, sino que el alumno debe de aportar un esfuerzo adicional.

2. ERP Openbravo

En este capítulo se explicará el concepto de un ERP, se profundizará sobre cómo es la metodología de trabajo en Openbravo, describiré el funcionamiento del ERP de Openbravo e indicaré cuales son los pasos para crear un entorno de Openbravo y que herramientas hay que utilizar.

2.1 ¿Qué es un ERP?

Un ERP es una aplicación que automatiza las actividades organizativas y la gestión de la cadena de suministros, mediante la utilización de una base de datos y la incorporación de las mejores prácticas, de forma que facilite una toma de decisiones más rápida, una reducción de los costes y un mayor control.

Dependiendo de lo que entendamos por sistemas ERP, se puede distinguir entre:

1. Genéricos. El sistema ERP es empleado por empresas de diferentes sectores y características. Esta acepción es la más adecuada con el concepto de ERP.

2. Pre-parametrizados. El ERP antes de implantarse se adapta al sector en el que trabaja la empresa según las necesidades que tenga.

3. Individualizados o a medida. El sistema ERP se personaliza según las necesidades de una compañía en concreto.

Uno de los factores más destacables de un ERP es que centraliza y ordena la información de la empresa en un solo lugar, así cualquier suceso queda a la vista de forma inmediata y se puede agilizar el trabajo. En el momento en el que se decide implantar un ERP se eliminan las barreras entre el resto de

departamentos de la empresa, de esta toda la información es accesible para todo el mundo.

Los sistemas ERP son avances que vinieron a sustituir a los anteriores sistemas MRP, los cuales solo se centraban en la planificación de materiales y capacidades productivas. Los ERP tiene herramientas para realizar la planificación de los trabajos en planta. Esta planificación se realiza enfrentando los requerimientos de materiales y capacidad de los productos a fabricar contra las existencias y capacidades sin asignar. Los ERP más completos ofrecen módulos para planificar a capacidad finita.

Entre las características de los ERP cabe destacar:

1. Tecnología Cliente-Servidor. Existe un sistema centralizado, normalmente un servidor, que almacena toda la información generada en las áreas organizativas y procesa la información para suministrarla a los clientes. Esta característica es la que diferencia a un ERP nuevo de los primeros modelos que salieron al mercado.
2. Al ser una tecnología abierta pueden utilizar distintas plataformas, sistemas operativos o bases de datos puesto que no necesitan un hardware específico.

3. Estandarización. Los ERP se pueden implementar en distintas organizaciones.

4. Modularidad. Se suele tener un núcleo al que se le añaden módulos que contienen distintas áreas de la empresa.

5. Capacidad de adaptación. Los ERP se pueden adaptar muy fácilmente a las necesidades de los distintos clientes gracias a su fácil abstracción.

6. Flexibilidad. Un ERP puede ser implantado modificando los procesos de trabajo existentes o llevando a cabo una reestructuración

para mejorar los procesos de negocio o eliminar los que ya no sean importantes.

7. Integración. Como he mencionado anteriormente la información que se recoge es centralizada en una base de datos que reduce la repetición de documentos y los tiempos de respuesta.

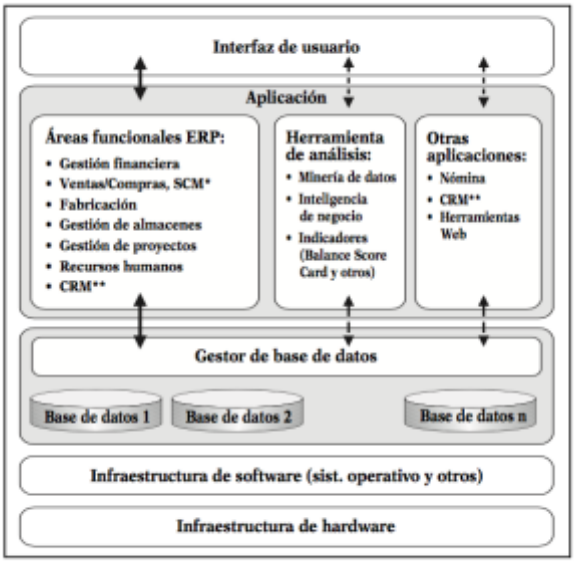
Esta serie de características hacen que los ERP aporten una serie de ventajas para las grandes multinacionales ya que suponen una forma de homogeneizar los procedimientos de trabajo.

La gran cantidad de ventajas de los ERP provienen de la integración de la información en una única base de datos común. De esta forma todo el mundo puede tener la información de forma inmediata.

La integración puede incluir en la cadena de valores a los proveedores y clientes, haciendo que las compañías mejoren su posición respecto a sus competidores. Esta mejora posicional normalmente suele ser una mejora de los resultados en temas de ventas, beneficios, reducción de los niveles de inventario.

Por terminar este apartado, se podría decir que un ERP se puede dividir en distintas capas:

FIGURA 2
REPRESENTACIÓN DE UN ERP



* SCM: Supply Chain Management
** CRM: Customer Relationship Management

FUENTE: Microsoft-Tomás Navarro 2005.

[1] (Imagen 2.1)

Siguiendo esta estructura se pueden destacar los siguientes componentes:

1. Infraestructura de hardware y comunicaciones. Este aspecto viene determinado por las características y requisitos del fabricante del ERP y también por cómo quiere el usuario de esta herramienta proteger los datos y procesos. Estos procesos de negocio dependen fuertemente de la forma en la que funcione la aplicación por lo que la información se tiene que proteger cautamente.

2. Infraestructura de software. Tras haber elegido la infraestructura de hardware y comunicación es necesario estudiar qué software se utilizará. Hoy en día los sistemas operativos más importantes son, Windows, Unix y Linux. Con ello, se llega a lo que propiamente es el ERP, dependiendo de la arquitectura que haya elegido el fabricante, la aplicación y el interfaz pueden ir en un solo bloque o separados. La forma más utilizada últimamente por los fabricantes es la de mantenerlos separados, ya que permite un mayor aprovechamiento de los recursos, de esta forma se puede desplazar la ejecución de la aplicación a otra máquina de usuario.

3. Gestión de base de datos. Aparecen muchos tipos debido a la gran cantidad de información que se puede almacenar y también por la forma en la que se puede gestionar y proteger dicha información.

4. Aplicación. Esta es la parte más importante del sistema ya que va a aportar la funcionalidad que el usuario desea reflejando los procesos internos de la empresa. Por esta razón se suele personalizar el diseño estandarizado por el fabricante para poder adecuarse de mejor manera a las especificaciones exigidas por el cliente. Los ERP constan de módulos para dar distintas funcionalidades como: gestión financiera, Ventas/Compras/SCM, fabricación, gestión de almacenes, logística, gestión de proyectos, CRM (*Customer Relationship Management)* y recursos humanos.

   Generalmente no suelen implementarse todos los módulos ya que no tienen por qué utilizar esos procesos dentro de la empresa.

5. Interfaz de usuario. Todo lo anterior se completa gracias a la interfaz con la que el usuario podrá trabajar con la aplicación.

2.2 Metodología trabajo Openbravo

Openbravo es una empresa muy flexible, los empleados pueden elegir si trabajar en sus casas o en la oficina, al tener que tratar con una aplicación web permite este tipo de cosas. Sin embargo, no todas las empresas que desarrollan aplicaciones tienen esta suerte. En Openbravo se logran comunicar muy fácilmente gracias a herramientas como Skype o Gmail, entre otras, para poder gestionar las tareas.

Todos los días, en cada equipo se realiza una reunión por Google Hangouts para fijar los nuevos objetivos y para repasar los antiguos. De esta forma todo el equipo está al tanto de cuál es el estado del proyecto y si alguien necesita ayuda poder echarle una mano. Con estas reuniones se consigue tener un mayor control sobre el desarrollo del objetivo y así poder llegar al resultado final.

Además de esta herramienta de Google, se utiliza Skype para comunicarse con los integrantes del resto de equipos, con compañeros que trabajan en otros países, con los proveedores, con los clientes y con otras empresas para establecer negociaciones.

Estas dos herramientas son las más utilizadas dentro de la empresa y con ellas se consigue una gran sincronización entre los equipos.

No obstante, estas herramientas sirven para la comunicación verbal dentro de la empresa, pero existen otras herramientas de comunicación que son vitales para que los proyectos se desarrollen satisfactoriamente.

Una de estas herramientas es Mercurial. Aunque será explicada en los próximos capítulos es necesario introducirla para entender cómo trabajan dentro de Openbravo.

Mercurial es un sistema de control de versiones multiplataforma para el desarrollo de aplicaciones generalmente extensas. En Openbravo existe un

repositorio principal que es PI, en el que está el producto final de Openbravo ERP, para poder introducir algo dentro de este repositorio se somete a muchas pruebas de validación con tests automáticos y revisiones de empleados encargados solo en esa labor.

Todos los equipos tienen un repositorio propio, que es una copia de la última versión de PI, que recibe el nombre de PI acompañado del nombre del equipo. De esta forma, los equipos pueden trabajar sin ningún miedo de producir algún error en el producto final. El equipo de QA es el encargado de introducir los nuevos desarrollos dentro de PI. Para esto tiene que tener mucho cuidado y revisar varias veces los nuevos cambios.

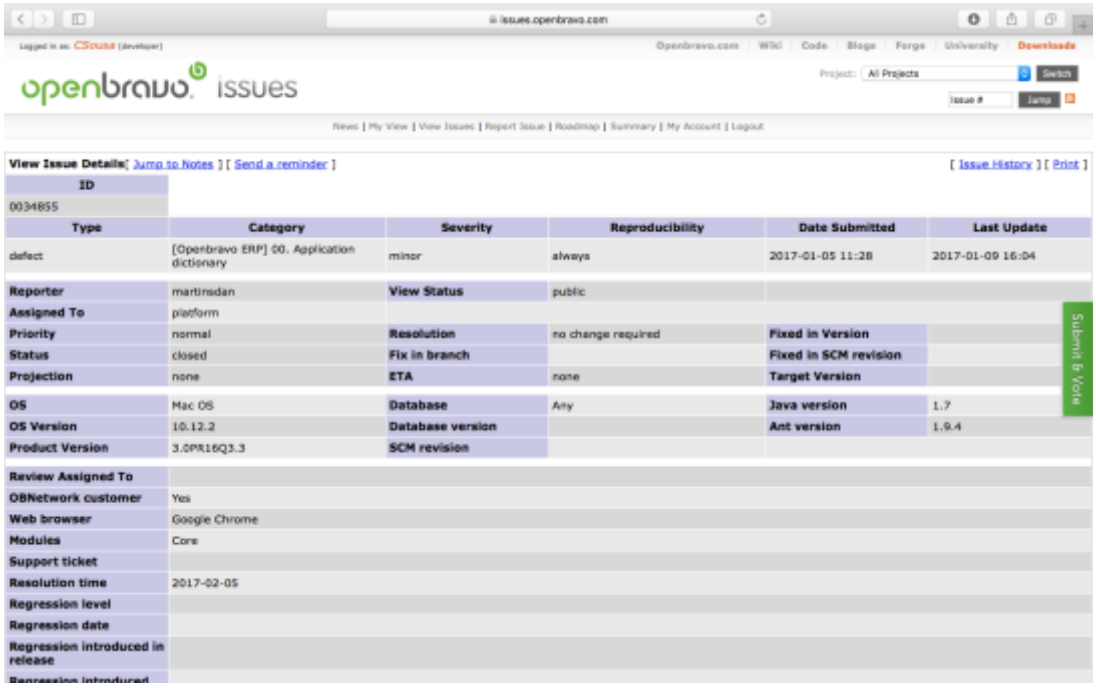
Además de Mercurial, también utilizan Jenkins, un software de integración continua que se aloja en un servidor para complementar el uso de Mercurial. Con este software se pueden lanzar tareas para poder controlar los cambios con pruebas de calidad y así tener un seguimiento más exhaustivo del proyecto.

Aun teniendo esta serie de herramientas para facilitar la sincronización entre los trabajadores de Openbravo, también utilizan otros métodos de trabajo para organizarse cuando tienen que corregir errores en la aplicación.

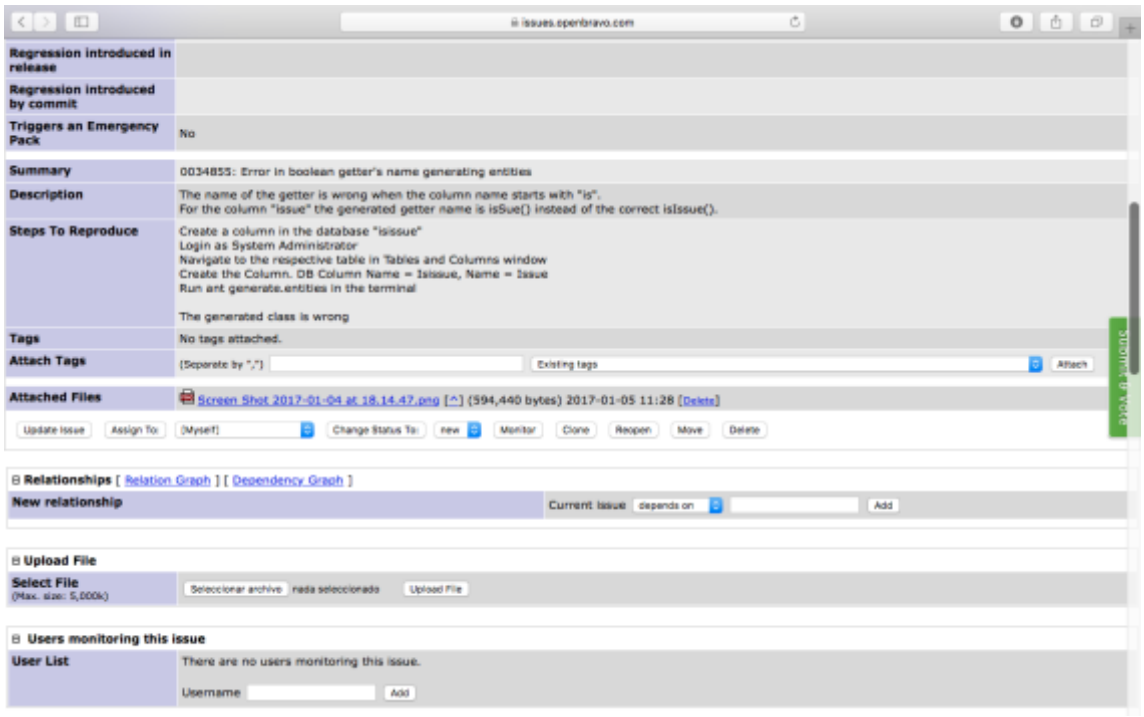
Utilizan una hoja de Excel para distribuirse la corrección de los issues de la aplicación web. En este documento aparece una serie de eventos que se le asignan a una persona, que cuando termina una acción tiene que dejarlo reflejado en la hoja de Excel para así tener un control sobre cuál es el estado de los issues.

Este documento de Excel se complementa con la página web que utiliza Openbravo para registrar los issues nuevos que aparecen en OpenbravoERP. En esta página web se indica cuál es el problema en concreto, cuando se produjo, en que versión del ERP apareció, a quién se le ha asignado la tarea de resolverlo, los pasos para poder reproducir el error, de vez en cuando hay una propuesta de la solución del issue y por último se deja un espacio de notas para que la persona que lo resuelva deje indicaciones de cómo lo ha solucionado y cuáles son los pasos a seguir para demostrar que el error ya no aparece.

En estas imágenes de debajo se puede ver un ejemplo de cómo es la página web que utiliza la empresa para poder almacenar los issues y así poder saber si se han cerrado o si todavía siguen abiertos…



(Imagen 2.2)



[2] (Imagen 2.3)

Para corregir estos issues hay una metodología establecida dentro de la empresa para que no falte ningún paso ya que si faltara podría ser crítico para el sistema.

Los pasos que se deben seguir son los siguientes:

1. Asignación del issue. Cuando alguien tiene que resolver un issue lo primero que tiene que hacer es asignárselo en la hoja de Excel que existe para planificar las tareas.

2. Registrar el issue en la página web. Es necesario cambiar el estado en la página web para que el resto de trabajadores sepan que ese issue está ya asignado a una persona para que lo corrija.

3. En el caso de que sea una regresión es necesario especificarlo en la página web y añadir la causa.

4. Actualizar a la última versión de PI. Antes de empezar a modificar algo en el código es muy importante traer a local los nuevos cambios que existan en el repositorio por si otra persona ha corregido algo importante y no estaba en lo último que se tenía en local.

5. Si es para un módulo habrá que clonar el módulo y las dependencias asociadas a este. También habrá que utilizar la rama adecuada y actualizarla al último cambio.

6. Hay que mirar si se está utilizando la versión adecuada de eclipse y también si se han importado las preferencias que utiliza Openbravo para formatear el texto (utilizan unas preferencias de formato para que el texto siga una serie de reglas http://wiki.openbravo.com/wiki/How_to_setup_Eclipse_IDE#Import_Preferences).

7. Realizar los cambios oportunos para corregir el error y actualizar el copyright del código.

8. Comprobar la corrección en postgres y en Oracle.

9. Si es una regresión, hay que corregirla y testearlo en cada *backport*.

10. Escribir en la página web los pasos a seguir para probar que el error ya no aparece.

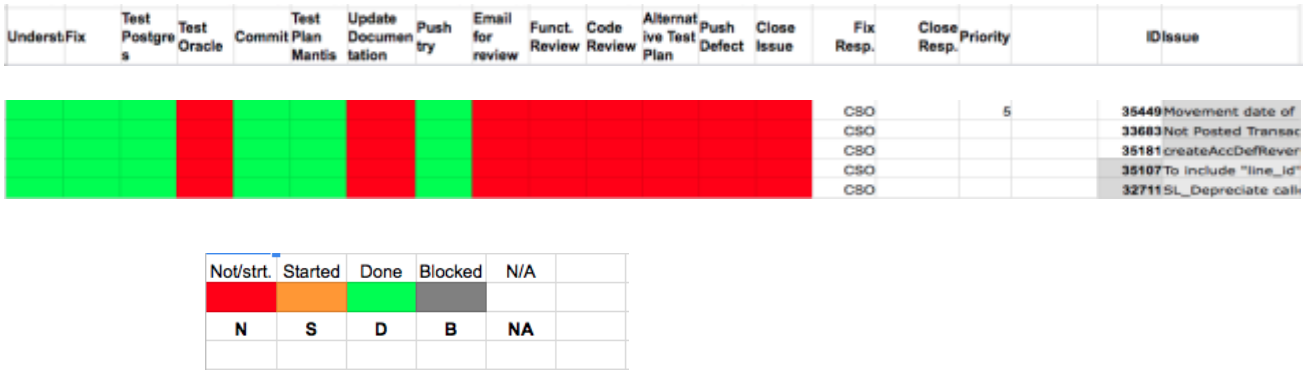
11. Preparar un commit de la corrección usando *hg commit* y escribiendo una cabecera del mensaje menor 80 caracteres de la forma: "Fixes issue <nº del issue>: un resumen del issue" y el cuerpo del mensaje: "Descripción de la corrección".

12. Si se trata de un issue en el núcleo hay que *pushear* el commit a *try*.

13. Habrá que exportar el commit a un fichero llamado <nº issue>. diff usando *hg export.*

14. Adjuntar el archivo <nº issue>. diff en un mail para enviárselo al equipo para que revise la corrección y le dé el visto bueno.

15. Actualizar la hoja de Excel con cada paso realizado.



(Imagen 2.4)

Una vez que el responsable de revisar las correcciones les da el visto bueno se manda al equipo de QA para que lo metan dentro de PI y cierren el issue.

2.3 Descripción ERP Openbravo

Openbravo ERP es el producto más importante que desarrolla la empresa, junto al Openbravo Web POS, que desarrolla la empresa. Para poder trabajar con esta herramienta se necesita cohesionar una serie de elementos para formar lo que se llama entorno. Este entorno está formado por las siguientes herramientas:

1. Postgres. Es una base de datos en la que se almacena en tablas la información que se va a utilizar después en la aplicación web.

2. Apache Tomcat. Se trata de un contenedor web de servlets escrito en Java en el que se encuentran los archivos que se ejecutan para dar la lógica a Openbravo ERP.

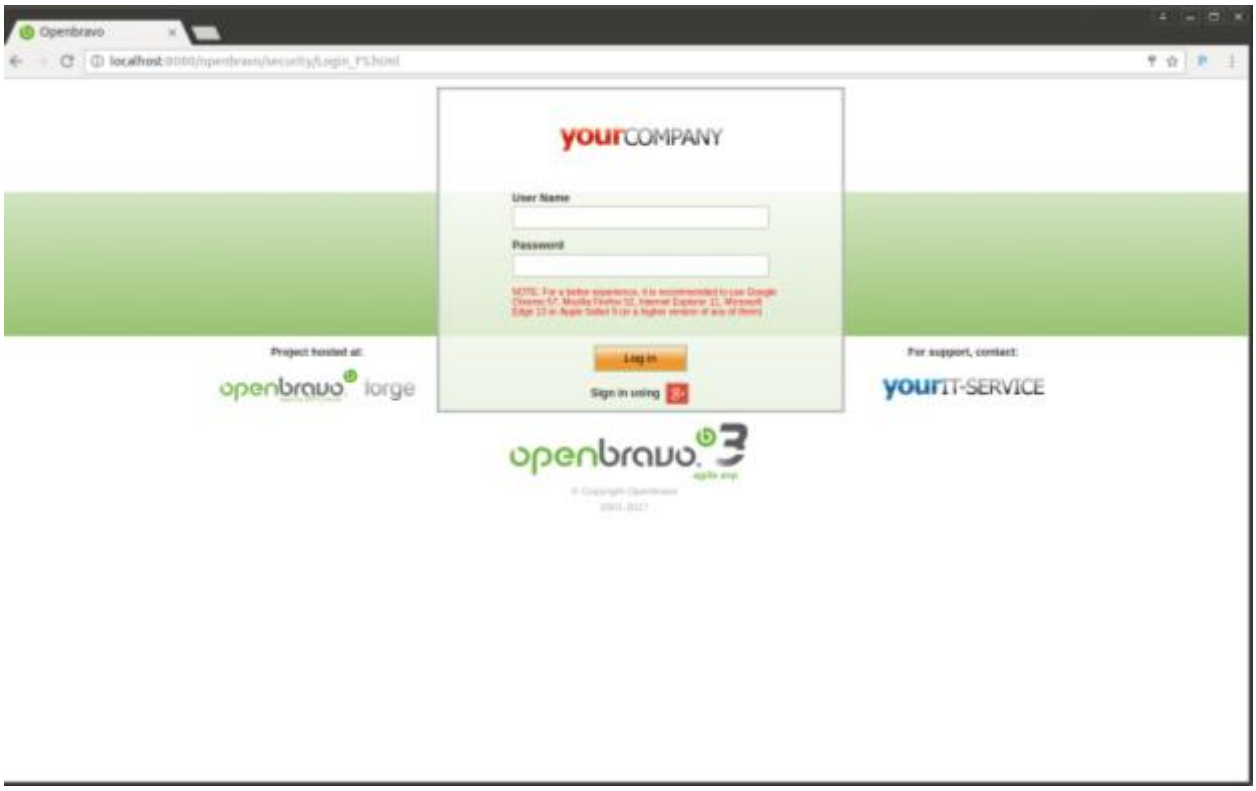
3. Eclipse. Esta herramienta se utiliza para poder desarrollar de una forma más rápida y más cómoda que con un editor de texto, provee muchas herramientas que facilitan la programación.

4. Ant. Una herramienta utilizada para lanzar tareas en el proceso de compilación.

Con estas herramientas se consigue crear un Entorno del ERP de Openbravo, aunque esto será mejor explicado en el siguiente apartado. Todos los archivos necesarios para poder levantar el entorno se tienen que obtener del repositorio principal de la empresa en la que está el ERP (https://code.openbravo.com/erp/devel/pi).

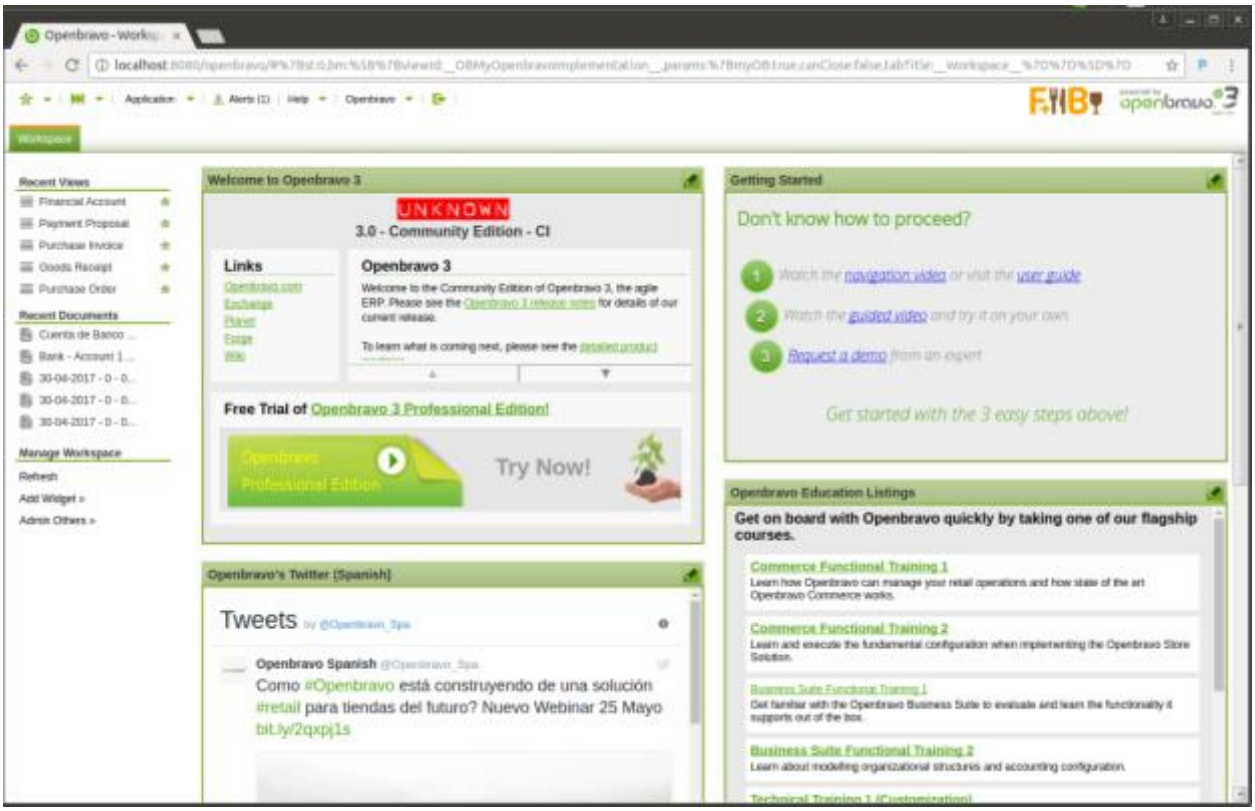
Una vez instaladas las herramientas y con el clon del repositorio de pi se puede levantar tomcat en eclipse para podernos conectar con el navegador a la

dirección localhost:8080/Openbravo y así utilizar OpenbravoERP. Lo primero que veríamos al conectarnos con el navegador es la página principal para iniciar sesión.



(Imagen 2.5)

Una vez iniciada la sesión se abrirá la página principal del ERP y tendrá este aspecto:

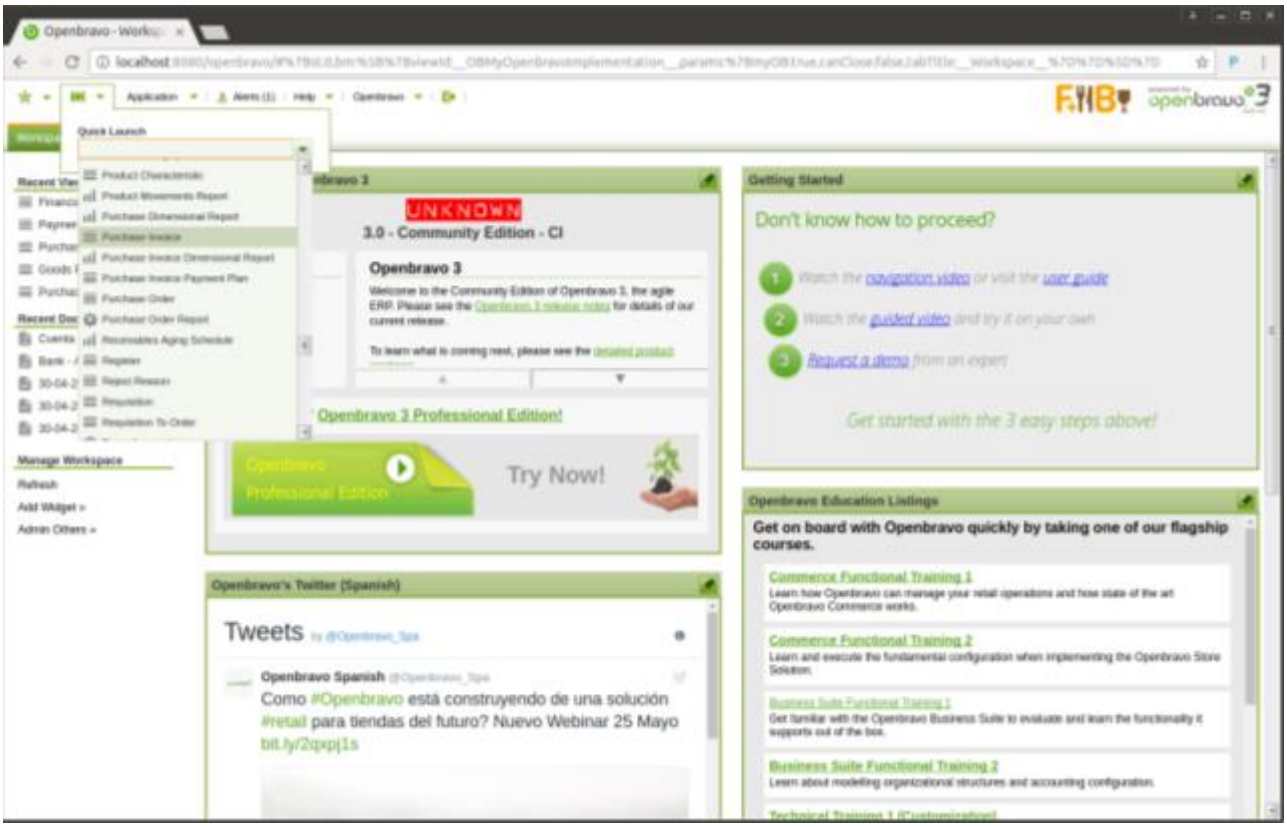


(Imagen 2.6)

En la imagen 2.6 se puede ver que este ERP tiene una barra con distintas pestañas.

(Imagen 2.7)

Las dos primeras pestañas sirven para poder navegar por la aplicación de forma rápida, al pulsar sobre ellas aparece un desplegable en el que se puede buscar la ventana deseada.



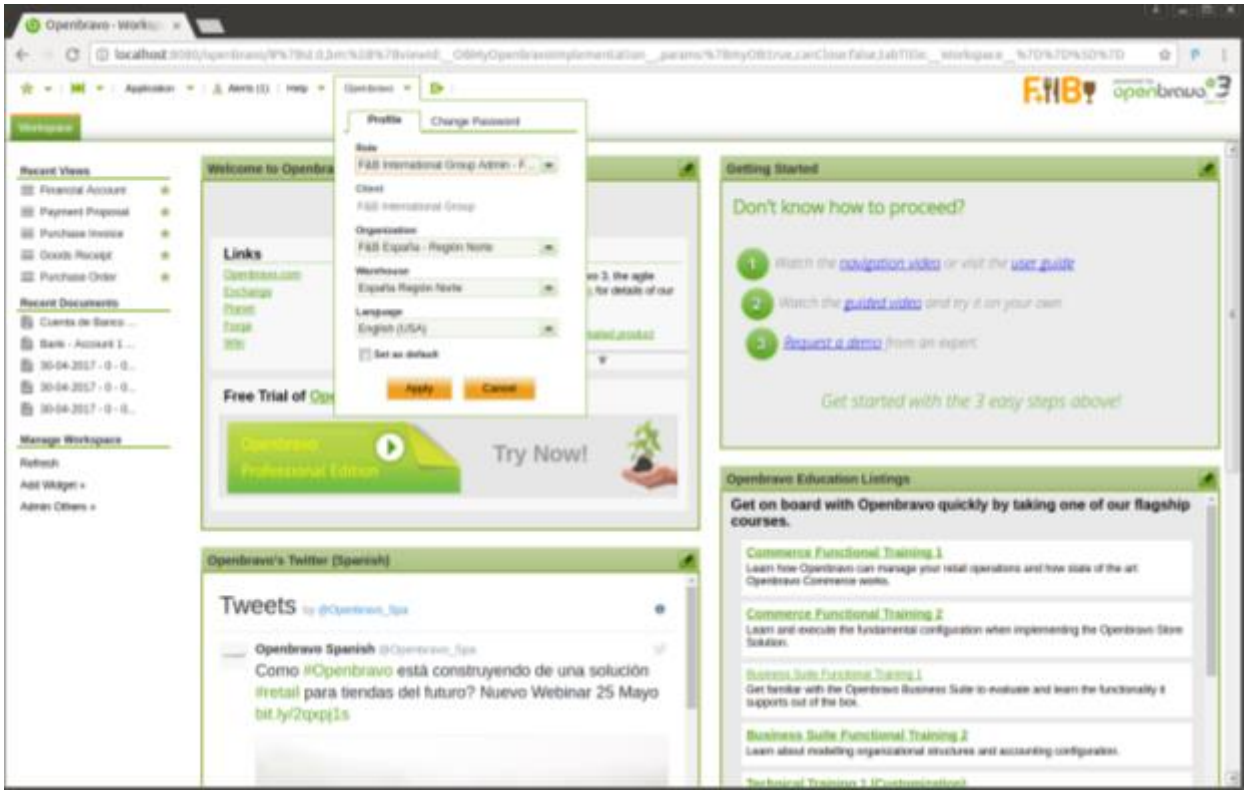
(Imagen 2.8)

- La pestaña ‘‘aplicación’’ también sirve para navegar por la aplicación, pero teniendo que buscar dentro del directorio de carpetas.

- La pestaña de ‘‘alertas’’ redirige a una ventana en la que aparecerán notificaciones que nos informarán acerca de que problemas hay en la aplicación.

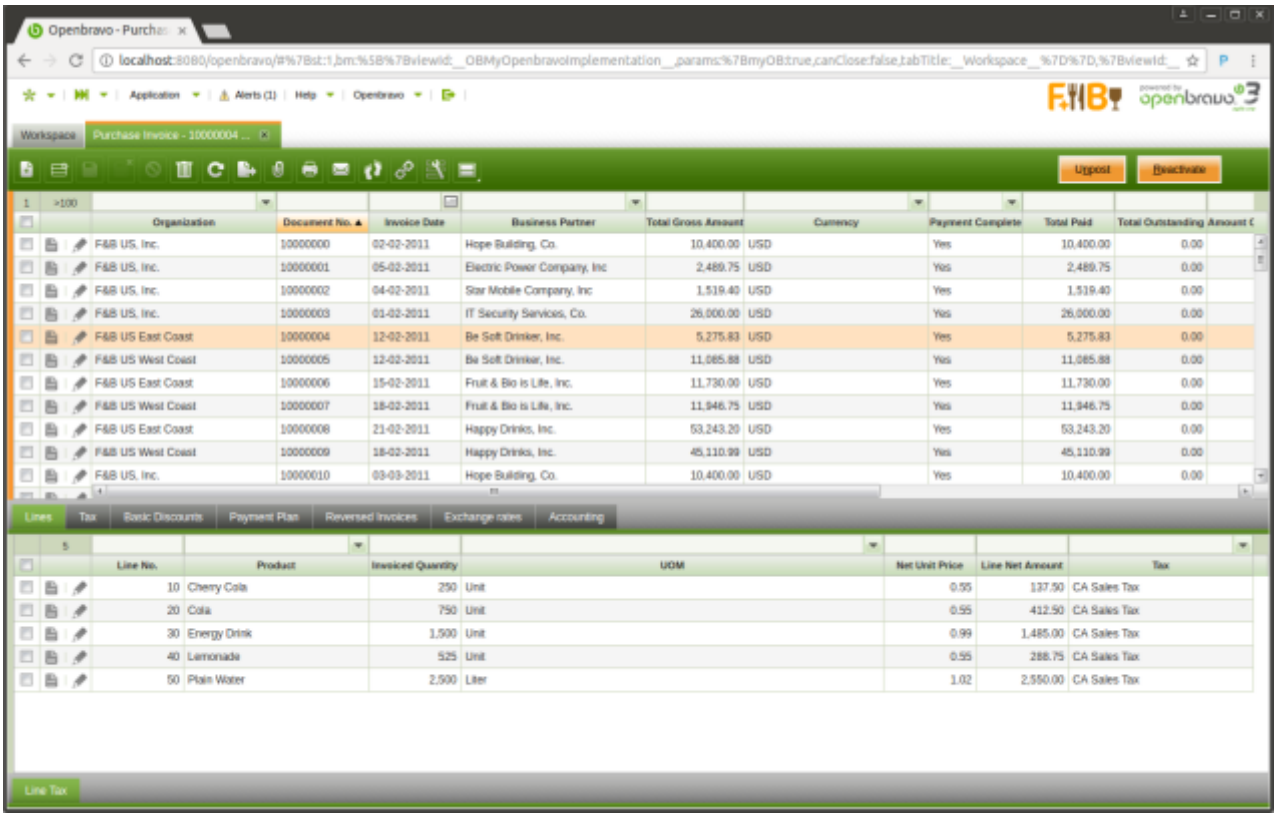
- La pestaña de ‘‘Help’’ ofrece una serie de ayudas básicas sobre el funcionamiento del ERP.

Al hacer clic sobre la pestaña Openbravo se abre un desplegable para poder cambiar la organización, el rol de la organización, el lenguaje y el almacén.



(Imagen 2.9)

Para crear una transacción o un movimiento de un producto o alguna gestión que se quisiera realizar lo único que hay que hacer es ir a la ventana adecuada. Por ejemplo, si quisiéramos crear una factura de compra habría que ir a la pestaña ⏭ e ir a la ventana "Purchase Invoice" de esta forma llegaríamos a la siguiente ventana.



(Imagen 2.10)

En esta ventana se pude apreciar que hay dos bloques. El bloque de arriba es la cabecera y en ella aparecen registros en los que se encuentra el nombre de la organización al que corresponde la factura, el número de documento de esa factura, el proveedor, la cantidad total, el tipo de moneda utilizada, el estado de la factura y otros campos más que están ocultos.

En el bloque de debajo se encuentra la pestaña de líneas, cada registro del bloque de arriba tiene un bloque de líneas en el que se muestran los productos que ha pedido dicha organización. Se puede apreciar que en este bloque también hay registros con los campos de producto, cantidad, precio unitario del producto, precio total y tipo de impuesto.

Encima de la cabecera se encuentra una barra verde en la que están los botones para crear un nuevo registro, para guardarlo, para eliminarlo y dependiendo del tipo de archivo que se esté generando también se pueden procesar, completar o crear las líneas automáticamente desde otro documento.

Con esta herramienta es muy fácil y sencillo recrear los procesos funcionales que sigue el interior de una empresa.

2.4 Creación Entorno Openbravo

En este apartado explicare cómo montar un entorno de Openbravo. En primer lugar, para un sistema operativo Unix (Linux) se necesita instalar y configurar los distintos componentes del entorno de Openbravo ERP (Tomcat, ant, postgres, java, eclipse…) y utilizar Ubuntu como sistema operativo en este caso.

Una vez se ha realizado la instalación de estas tareas hay que asegurar que:

- El usuario del sistema operativo tiene acceso de lectura y escritura al directorio de instalación de tomcat (CATALINA_BASE).

- Las variables de entorno (ej. ANT_OPTS, CATALINA_OPTS, JAVA_HOME, ANT_OPTS, etc.) han sido declaradas en las categorías específicas.

- Se tiene descargada la última versión de Eclipse.

Para poder seguir, se necesita descargar el código fuente del repositorio principal utilizando mercurial. No hay ningún tipo de restricción de lectura por lo que no se necesita ningún tipo de acreditación para acceder al repositorio.

Para tener la última versión del código, en un directorio en el que se quiera tener el clon, hay que hacer:

- *hg clone https://code.openbravo.com/erp/devel/main openbravo*
- *cd openbravo*
- *hg up 3.0PR17Q1.1*

Los desarrolladores de openbravo trabajan con el repositorio de *PI:*

- *hg clone https://code.openbravo.com/erp/devel/pi*
- *cd pi*

La instalación de Openbravo se realiza en varios pasos.

Configurar las propiedades.

En primer lugar, hay que configurar Openbravo ERP especificando algunas propiedades generales, como por ejemplo el directorio de instalación de tomcat, la conexión con la base de datos, etc. Esto debe de hacerse mediante línea de comandos ya que tiene que compilarse con ant.

Para compilar hay que ir al directorio en el que se tiene el clon del repositorio y realizar:

- *Ant setup*

Utilizando este comando compilaremos y ejecutaremos automáticamente una aplicación de consola. Utilizando el manual que aparece en consola llegamos hasta el final y seleccionamos *Accept* para aplicar los cambios y cerrar la aplicación.

Como resultado de la ejecución de este programa se habrá generado un nuevo archivo llamado *Openbravo.properties* en el subdirectorio *config*. Este archivo contiene las configuraciones que se han establecido con el programa.

Instalar las fuentes

Para instalar Openbravo ERP hay que realizar tres procedimientos principales:

- Se tienen que introducir unos datos iniciales dentro de la base de datos de Openbravo ERP.

- Las fuentes tienen que generarse.

- Todas las fuentes se tienen que compilar en binarios que después serán ejecutados en el servidor web.

Todo esto se realiza ejecutando *ant install.source* desde la raiz del directorio en el que se encuentra el clon del repositorio.

Este proceso puede llevar unos 25 minutos, dependiendo de la configuración del hardware. Para tener un control sobre este proceso se suele redirigir la salida del log a un fichero para que después se pueda analizar o enviar al equipo de soporte en caso de que exista algún problema.

Una vez que el proceso acaba debe aparecer el mensaje de *BUILD SUCCESFUL* al final del archivo.

Una vez este paso se ha realizado correctamente ya se puede importar a Eclipse.

- Importar Eclipse

Primero se debe lanzar Eclipse, una vez que ya se ha abierto hay que ir al menú *Project* y deshabilitar *Build Automatically*.

Segundo hay que importar cuatro proyectos. Para eso hay que dirigirse al menú *File* > *Import* y después *General* > *Existing Projects into Workspace*. Los cuatros proyectos a importar son los siguientes:

openbravo *XXX\opensource\openbravo\erp\devel\main*

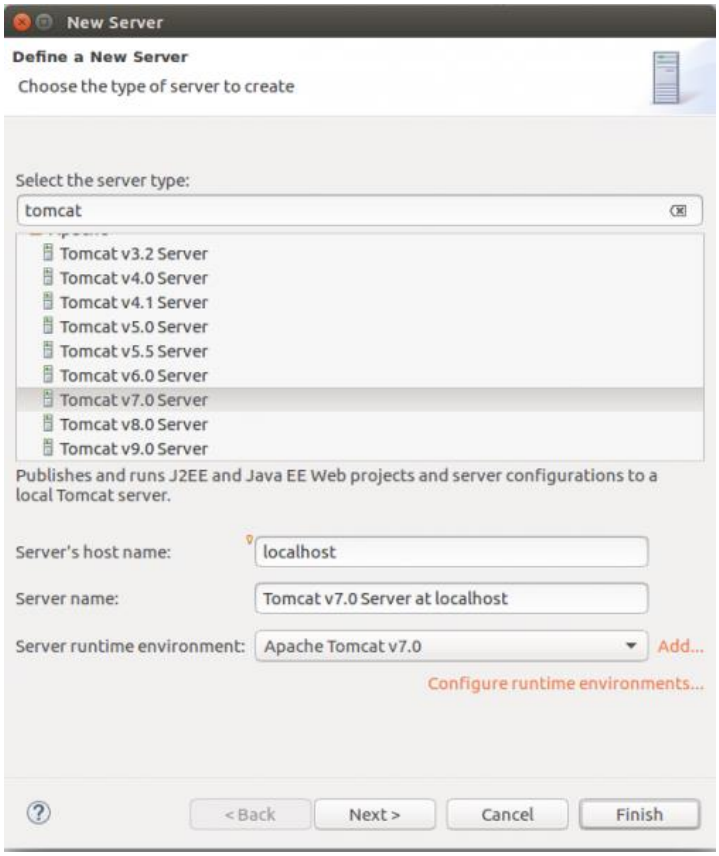
OpenbravoCore *XXX\opensource\openbravo\erp\devel\main\src-core*

OpenbravoTrl *XXX\opensource\openbravo\erp\devel\main\src-trl*

OpenbravoWAD *XXX\opensource\openbravo\erp\devel\main\src-wad*

- Crear servidor Tomcat

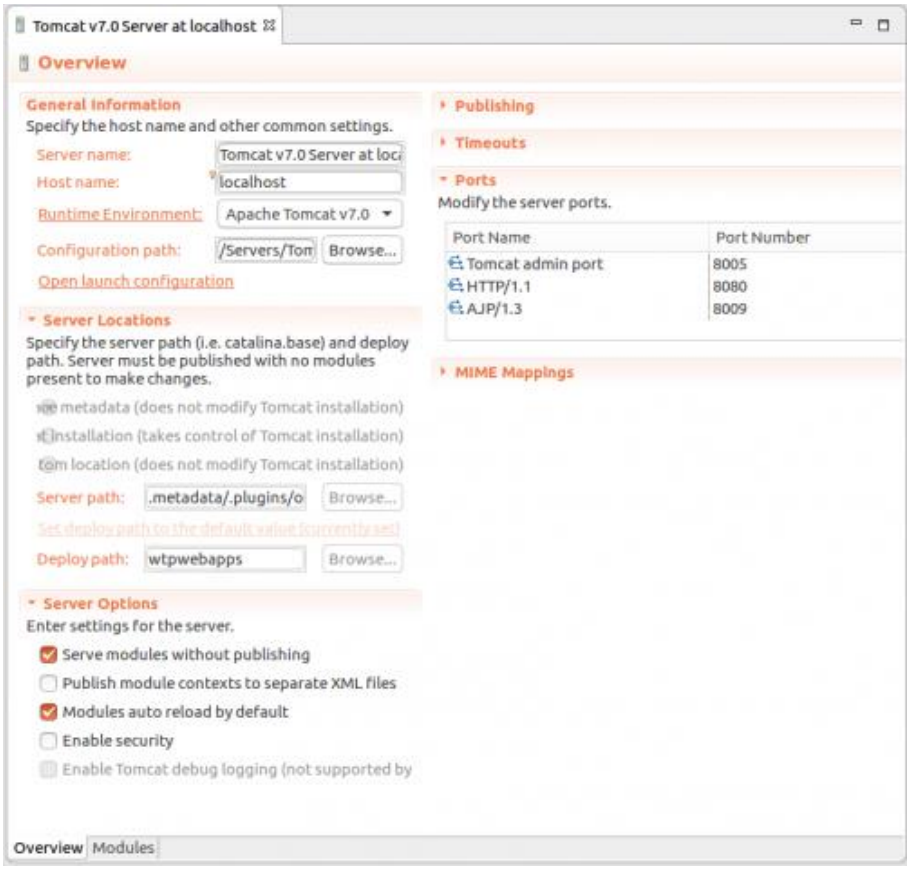
Primero hay que abrir la vista *Servers* y crear una nueva instancia del servidor tomcat:



(Imagen 2.11)

Dentro del asistente hay que seleccionar *openbravo* y añadirlo a *configured resource.* Después de esto seleccionar *Finish.* Una vez se hayan finalizado estos pasos la instancia del servidor aparecerá en la vista de *Servers*.

El siguiente paso es hacer doble clic en el servidor para realizar algunos cambios de configuración.



(Imagen 2.12)

En esta ventana hay que seleccionar el flag *Serve modules without publishing* además en la pestaña *modules* hay que editar el módulo openbravo y deseleccionar el flag *Auto reloading enabled.*

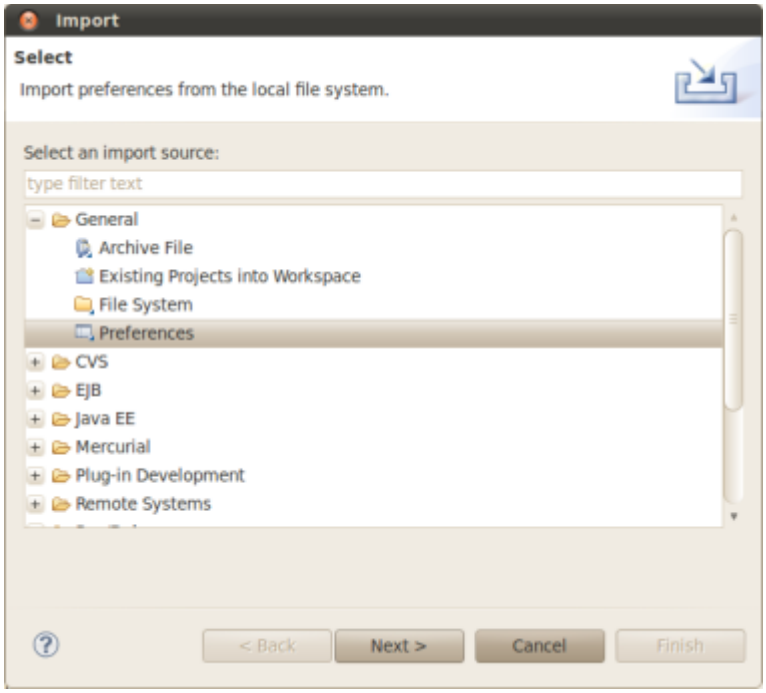
Al terminar esto, hay que ir a *General Information* y hacer clic en *Open launch configuration link* después hay que cambiar a la pestaña *Arguments* y añadir la línea que viene a continuación al final del input *VM arguments* input:

-server –Djava.awt.headless=true –Xms384M –Xmx1536M –XX:MaxPermSize=256M

Después en *Server options* hay que seleccionar la casilla *Serve modules without publishing* y en *Timeouts* hay que cambiar los valores de *Start* y de *Stop*

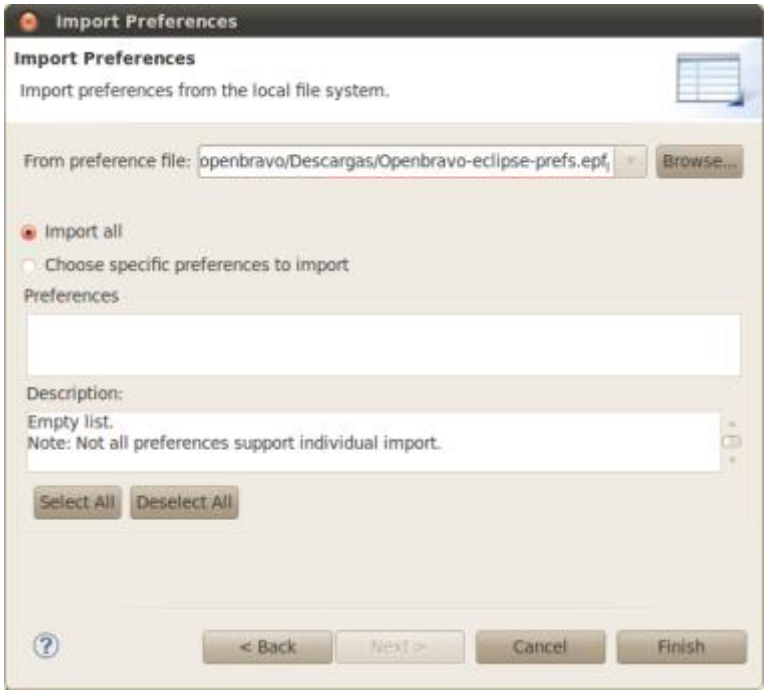
a 120 segundos. Una vez finalizada la configuración del servidor hay que guardar.

Lo siguiente que hay que hacer es cambiar las preferencias y poner las mismas que se utilizan en Openbravo para que formatee el texto adecuadamente. El fichero de preferencias se encuentra en la carpeta *openbravo/config/eclipse* en el proyecto *development.* Para importar el archivo hay que ir a la pestaña File > *Import* después seleccionar *General* > *Preferences* y clicar en el botón *Next button.*



(Imagen 2.13)

Lo siguiente que hay que hacer es navegar a *openbravo/config/eclipse/* y buscar el archivo *openbravo-eclipse-prefs.epf*, después hay que seleccionar el botón *Import all* y por ultimo darle a *Finish.*



(Imagen 2.14)

Una vez se haya finalizado habrá que seleccionar todos los proyectos, habrá que refrescarlos y hacer el rebuild, esto solo debería de dar algún warning pero no debería de dar errores. [3]

3. Análisis y diseño

En este trabajo se han ido completando una serie de objetivos para poder llegar al resultado final.

Para empezar, he tenido que aprender cómo funciona Openbravo ERP a nivel funcional. Para esto he realizado una serie de cursos formativos que suministra la empresa para que los trabajadores puedan coger soltura con el ERP.

En estos cursos se enseña a navegar por el ERP y a realizar flujos funcionales para recrear acciones que un cliente haría.

Antes de empezar con el desarrollo del proyecto, tuve que poner en práctica estos conocimientos adquiridos realizando labores para la empresa. Era necesario familiarizarme con las herramientas de trabajo que utilizaban, ya que iban a ser muy importantes a la hora de desarrollar el trabajo.

Una vez finalizada la fase de aprendizaje, mantuve varias reuniones con mi tutor en la empresa para fijar las condiciones del trabajo. En estas conversaciones se dejó claro que Openbravo ya tenía una serie de herramientas fijas que utilizaban para montar el entorno del ERP y que iban a ser necesarias para realizar el TFG. Estas herramientas son: Ant, Tomcat, Postgress, Mercurial, Jenkins, Eclipse y Java.

A la hora de decidir la forma de automatizar los flujos funcionales, se eligió Selenium porque, además de que Openbravo ya tenía una gran API desarrollada para realizar test automáticos, era necesario simular la interacción que tendría un usuario cualquiera con la interfaz de la aplicación web.

Selenium permite automatizar los flujos funcionales que la empresa solicitaba y además validarlos cada vez que se quiera ejecutar el test automático.

Cuando ya se fijaron las herramientas que se necesitaban para automatizar los flujos, el siguiente paso fue montarme un entorno de Selenium dentro del

entorno de Openbravo que ya tenía de los cursos formativos. Openbravo tiene un repositorio en el que se encuentra el entorno de Selenium con la API necesaria para realizar test automáticos. Este repositorio se llama PI-SMOKE y es mantenida por el equipo de QA.

Para montar el entorno tuve que utilizar las versiones de firefox 45.7.0.esr, java 8 y selenium-server-standalone-3.0.1.

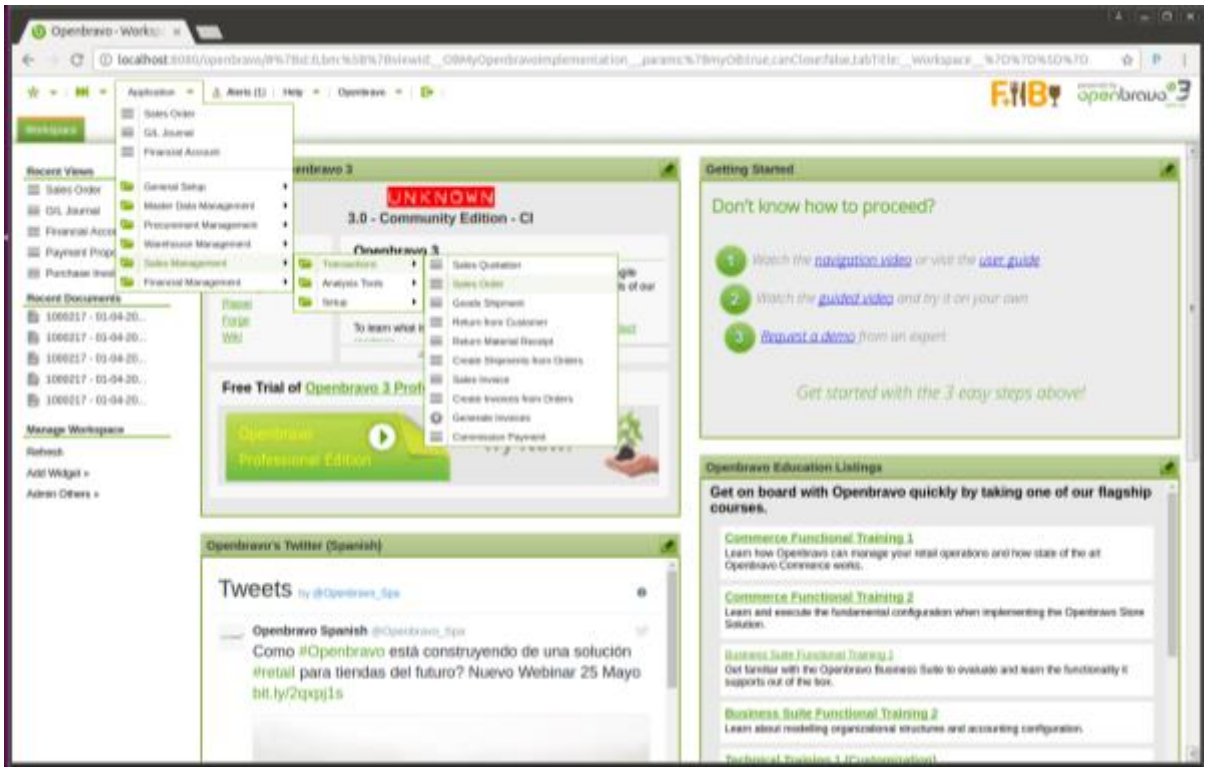
Una vez incorporado el entorno de Selenium tuve que aprender a utilizar la API de Selenium que tenía Openbravo.

Muchas de las ventanas del ERP de openbravo ya tienen su infraestructura creada en la API. De esta forma, se puede interactuar con las ventanas creando un objeto con los métodos necesarios.

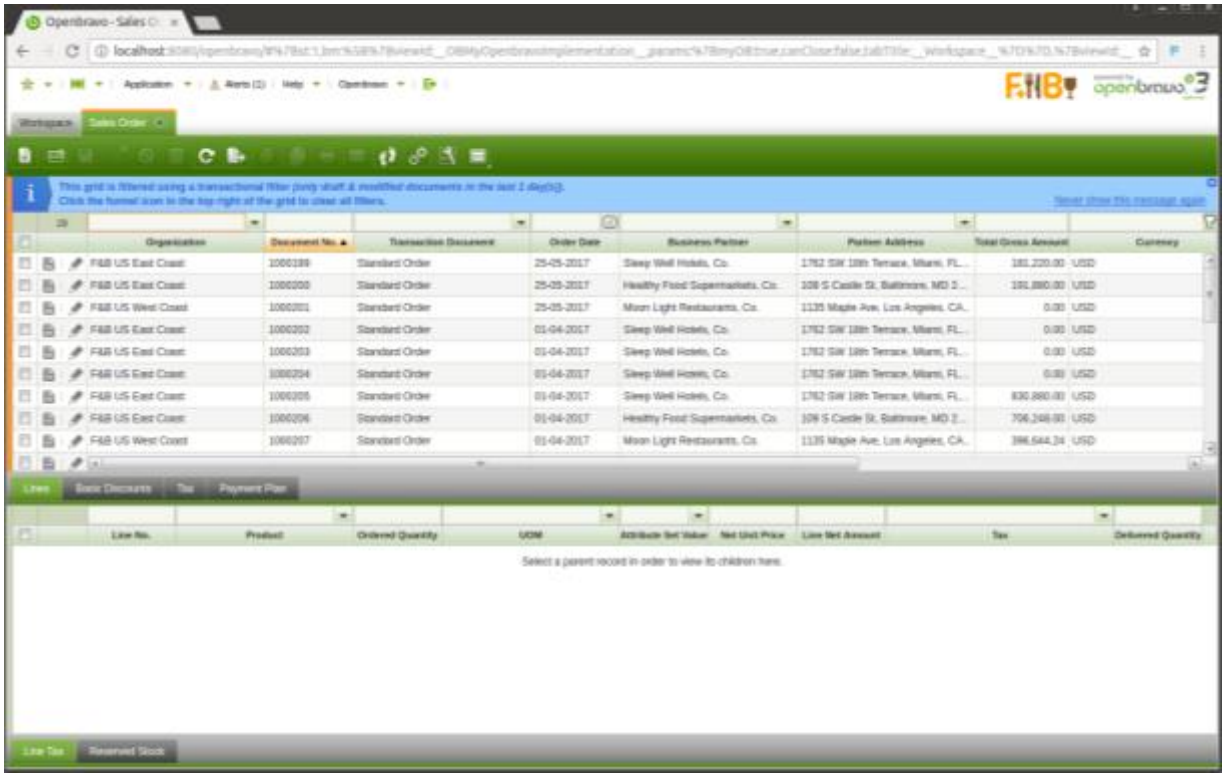
Como he mencionado, antes de empezar con el trabajo tuve que analizar cómo funcionaban los test automáticos para poder automatizar por mi cuenta. Por eso en este capítulo explicaré cómo crear un test automático y cómo funciona.

Primero enseñaré un flujo de ejemplo para qué se entienda mejor que es lo que hace cada línea en el código.

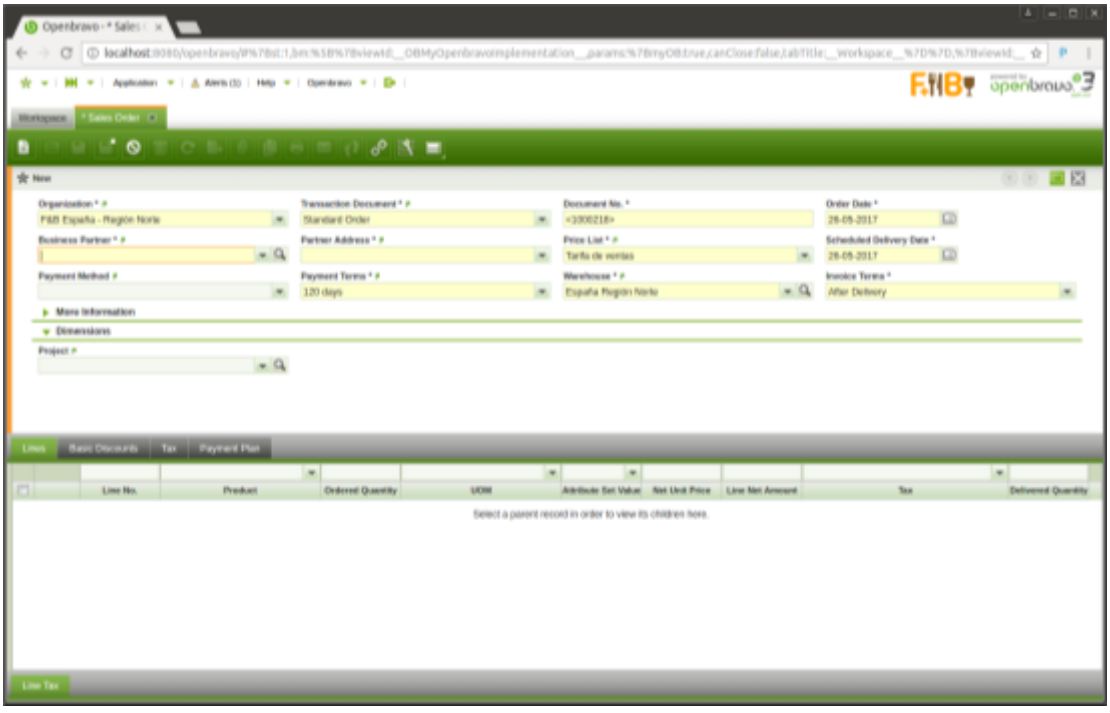
(Imagen 3.1)



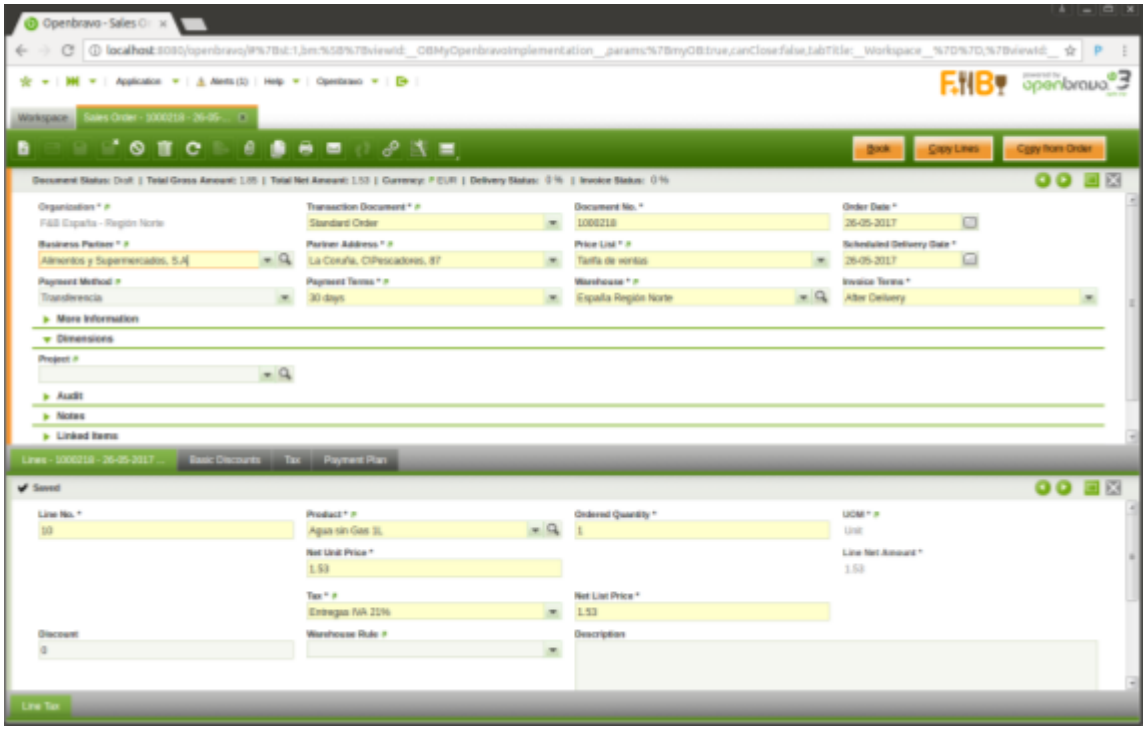
(Imagen 3.2)



(Imagen 3.3)



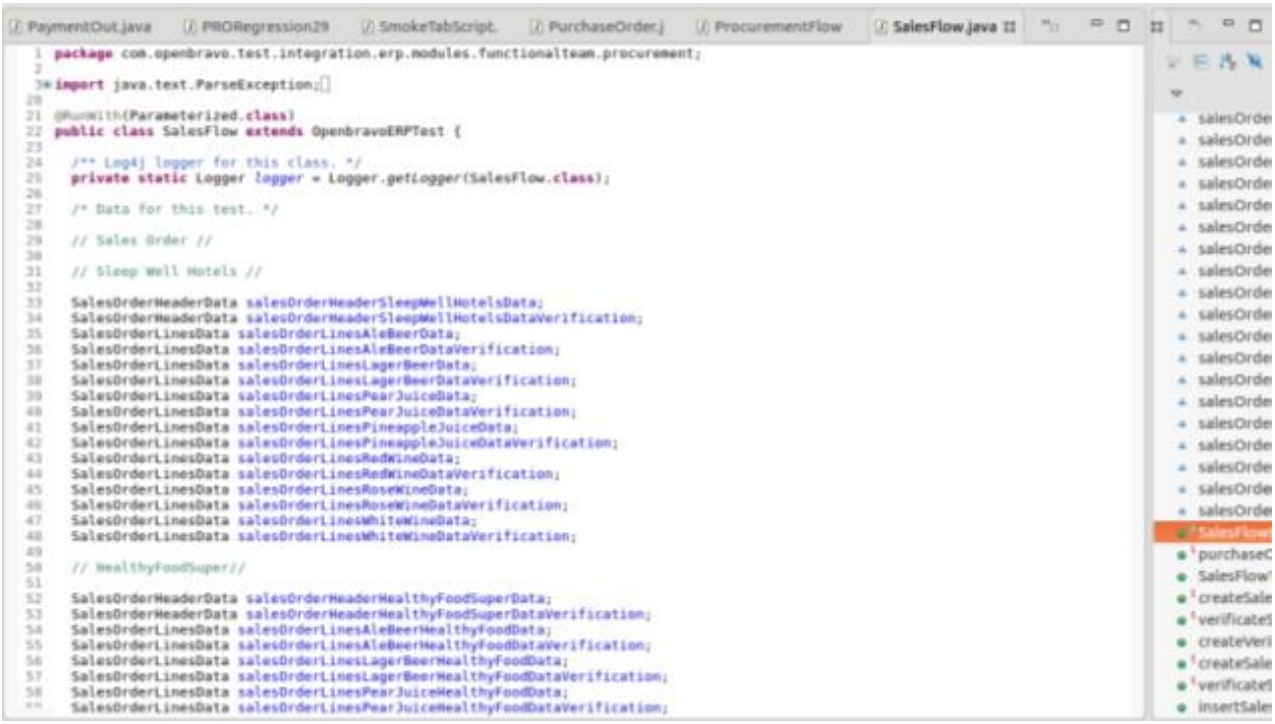
(Imagen 3.4)



(Imagen 3.5)

En este flujo funcional de ejemplo se crea un pedido. Para ello en la aplicación web es necesario hacer log, navegar hasta la ventana de *Sales Order*, crear un nuevo registro dándole al botón de nuevo registro, insertar los datos deseados en los campos de la cabecera, añadir los productos en la pestaña de líneas y completar el pedido haciendo clic en el botón de *Book*.

Ahora mostraré y explicaré la clase en la que se programa este flujo funcional:

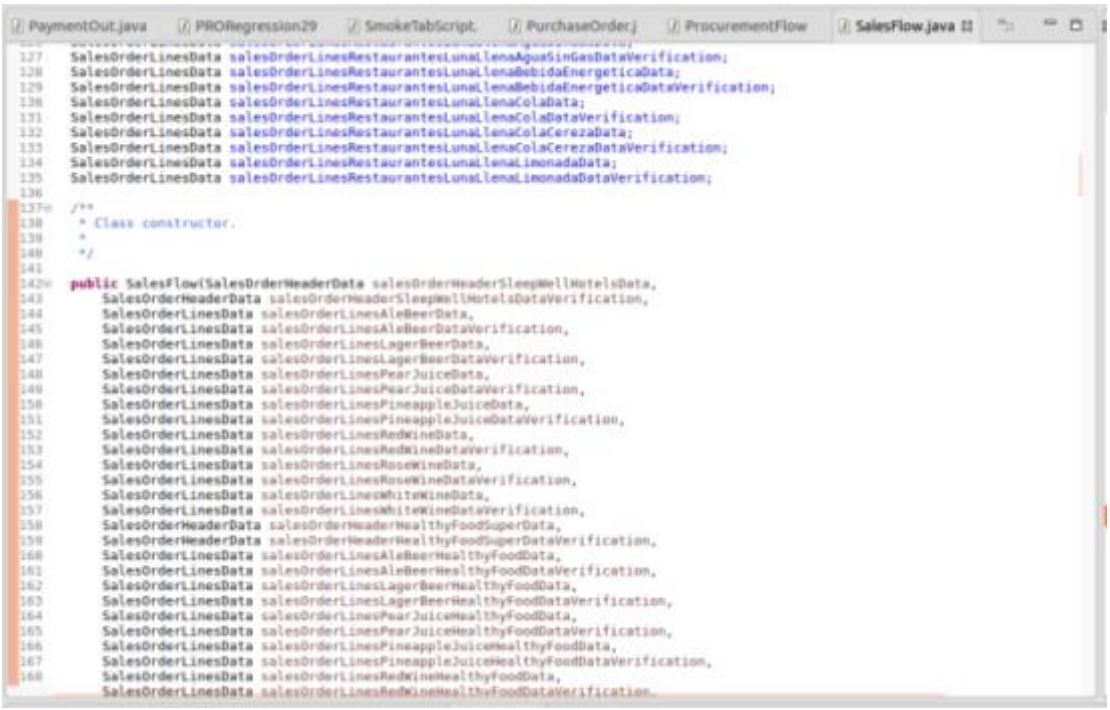


(Imagen 3.6)

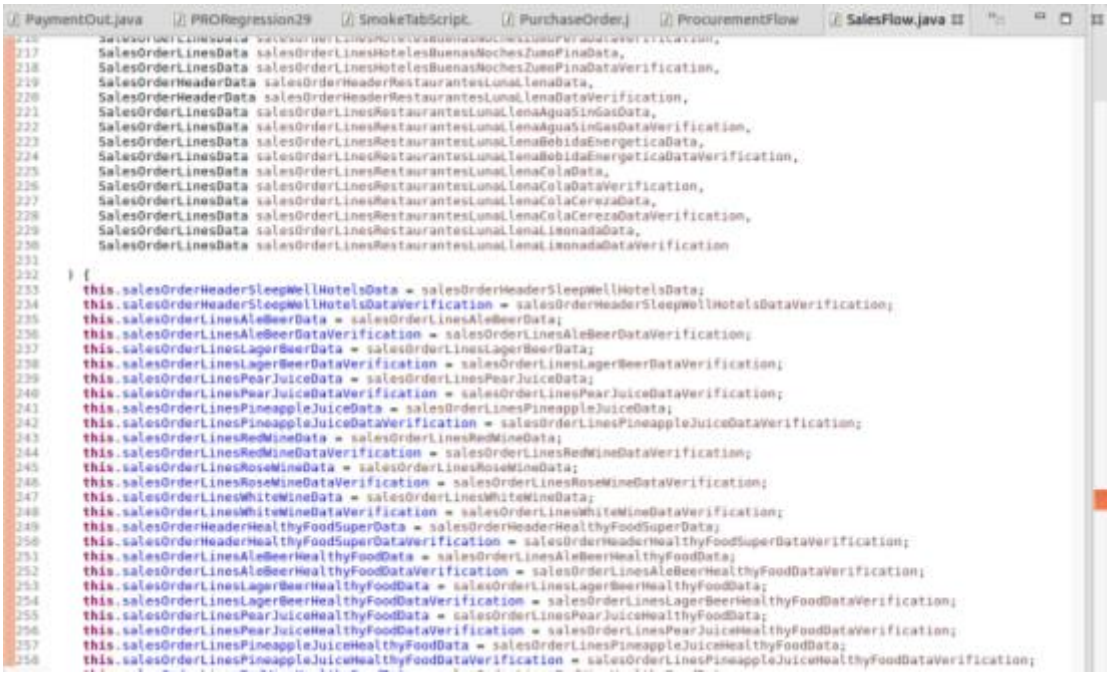
En la imagen 3.6 se puede ver que se declaran las variables que vamos a utilizar en este test. Es importante notar que estas variables son de dos tipos: *SalesOrderHeaderData* y *SalesOrderLinesData*. Estos dos tipos de objetos son en los que guardaremos los datos de los campos que vamos a querer rellenar dentro de las pestañas de la ventana *Sales Order*. Los objetos *SalesOrderHeaderData* llevarán los datos de la cabecera mientras que *SalesOrderLinesData* llevarán los datos de la pestaña *Lines*.

Además, podemos ver que los nombres de las variables representan de forma genérica la información que habrá dentro de cada objeto como, por ejemplo, el socio comercial o el nombre del producto.

Por último, cabe destacar que las variables que acaban con la coletilla *verification* son variables que sirven para verificar que el resto de campos que no han sido rellenados por el test tienen valores estables y esperados. Esto es una forma de validar que la información es correcta.



(Imagen 3.7)



(Imagen 3.8)

En las imágenes 3.7 y 3.8 se puede ver el constructor de la clase *SalesFlow*. En el constructor se inicializan los campos y es muy importante fijarse que tienen que seguir el mismo orden a la hora de declararse y a la hora de instanciarse ya

que después se rellenarán en las variables con los valores que se les introduzca en el método *@Parameters*.

(Imagen 3.9)

Al principio de la imagen 3.9, en la línea 322 se crea un objeto *LogInData* que es el encargado de hacer log en la pantalla inicial de la aplicación web con el username y la contraseña como aparece en la imagen 3.1.

Estos objetos se tienen que rellenar en el mismo orden en el que fueron declaradas las variables.

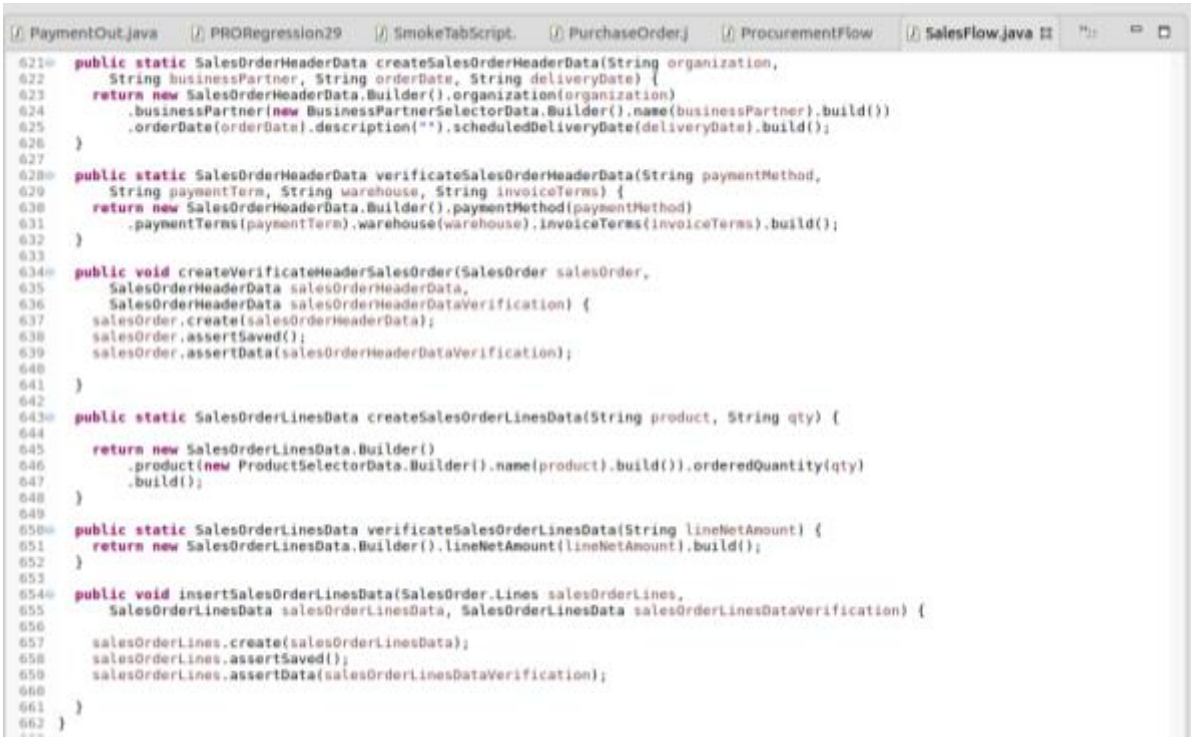
Debajo de esta línea se encuentra el método *@Parameters* en el que se crean los objetos *Data* que son los que tienen la información para los campos de las ventanas. Se puede apreciar que lo coloreado en azul son los datos que van a rellenar los campos de las ventanas en los lugares que corresponde.

Uno de los aspectos más importantes a la hora de realizar un test es el de validar una serie de campos, que se rellenan automáticamente, en función del resto de campos. Por ejemplo, en la línea 342 se está insertando el nombre del producto “Ale Beer” y la cantidad “36,000.00”, mientras que en la línea 343 se está creando otro objeto del tipo *SalesOrderLinesData* con el precio total del

pedido. Esto verifica que, al multiplicar la cantidad del producto insertado en la aplicación por el precio unitario (que se inserta automáticamente) es igual a 73,440.00 $.

De esta forma tenemos una herramienta muy útil para detectar errores en la aplicación. No solo para cuando lo ejecutemos manualmente, sino para cuando el test este dentro del proceso de integración continua.

Dentro de este método *@Parameters* hay varios métodos propios de la clase. Esos métodos son los siguientes:

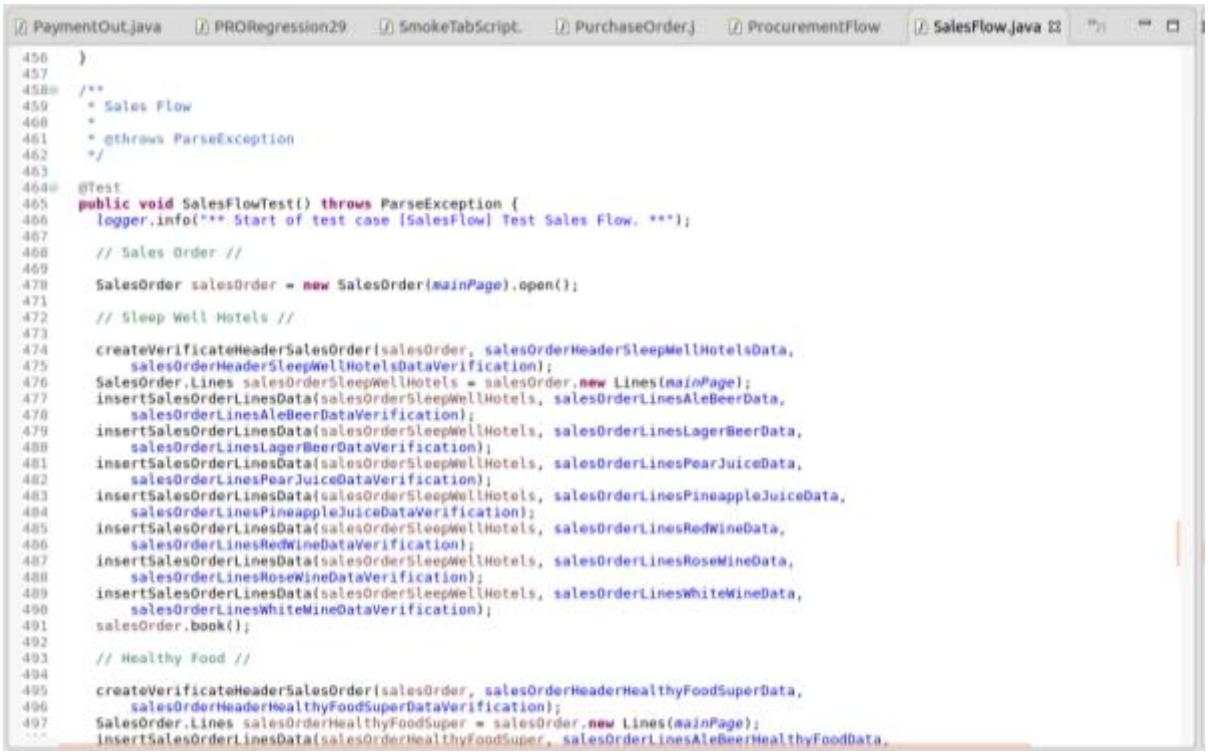



```
public static SalesOrderHeaderData createSalesOrderHeaderData(String organization,
    String businessPartner, String orderDate, String deliveryDate) {
  return new SalesOrderHeaderData.Builder().organization(organization)
      .businessPartner(new BusinessPartnerSelectorData.Builder().name(businessPartner).build())
      .orderDate(orderDate).description("").scheduledDeliveryDate(deliveryDate).build();
}

public static SalesOrderHeaderData verificateSalesOrderHeaderData(String paymentMethod,
    String paymentTerm, String warehouse, String invoiceTerms) {
  return new SalesOrderHeaderData.Builder().paymentMethod(paymentMethod)
      .paymentTerms(paymentTerm).warehouse(warehouse).invoiceTerms(invoiceTerms).build();
}

public void createVerificateHeaderSalesOrder(SalesOrder salesOrder,
    SalesOrderHeaderData salesOrderHeaderData,
    SalesOrderHeaderData salesOrderHeaderDataVerification) {
  salesOrder.create(salesOrderHeaderData);
  salesOrder.assertSaved();
  salesOrder.assertData(salesOrderHeaderDataVerification);

}

public static SalesOrderLinesData createSalesOrderLinesData(String product, String qty) {

  return new SalesOrderLinesData.Builder()
      .product(new ProductSelectorData.Builder().name(product).build()).orderedQuantity(qty)
      .build();
}

public static SalesOrderLinesData verificateSalesOrderLinesData(String lineNetAmount) {
  return new SalesOrderLinesData.Builder().lineNetAmount(lineNetAmount).build();
}

public void insertSalesOrderLinesData(SalesOrder.Lines salesOrderLines,
    SalesOrderLinesData salesOrderLinesData, SalesOrderLinesData salesOrderLinesDataVerification) {

  salesOrderLines.create(salesOrderLinesData);
  salesOrderLines.assertSaved();
  salesOrderLines.assertData(salesOrderLinesDataVerification);

}
}
```

(Imagen 3.10)

En esta imagen 3.10 se pueden ver los métodos internos de la clase. Estos métodos sirven para introducir dentro de los objetos *Data* la información que se quiere rellenar y la información que se quiere verificar.

El Objeto *Data* tiene una referencia al campo de la ventana que se quiere rellenar. Gracias a los métodos que tiene este objeto *Data,* es posible abstraerse de la lógica que hay por debajo para hacer esto.



(Imagen 3.11)

En la imagen 3.11 se encuentra el método *@Test*. En este método se describe cuáles son los pasos que tiene que seguir el test. En la línea 470 se puede ver que lo primero que se hace es instanciar la ventana *Sales Order* y abrirla.

Con esta línea, se realizaría lo que reflejan las imágenes 3.2 y 3.3. En las líneas 474 y 475 se llama a uno de los métodos internos que he ilustrado antes para abrir la pestaña de la cabecera, insertar, verificar y guardar los datos dentro de los campos. Esto se ve en la imagen 3.4 y 3.5.

En la línea 476 se instancia la pestaña *Lines* que es en la que se introducirán los productos del pedido. En la línea 477 al igual que en la línea 474 y 475 se llama a un método interno para repetir el proceso anterior. Una vez que ya se han insertado los valores en los campos de ambas pestañas se llega al resultado de la imagen 3.5.

Para finalizar solo quedaría comentar que en la línea 491 se llama al método de la ventana *Sales Order* para completar el documento.

Si existiera un error en la aplicación aparecería en la consola la línea exacta en la que se detiene la ejecución. En caso contrario el test acabaría con el registro creado con los datos insertados.

Después de aprender a realizar test automáticos hablé de nuevo con mi tutor para concretar cuales iban a ser los flujos automáticos que necesitaba la empresa que desarrollara.

Antes de empezar a automatizar estos flujos tuve que reproducirlos manualmente para ver cuáles eran los campos obligatorios que exigía la aplicación, los campos que se rellenaban automáticamente y por último mirar cuales eran los pasos exactos que había que automatizar.

A continuación, describiré cuáles son los flujos que han sido automatizados en este trabajo:

Flujo de Compra

Para generar este flujo se van a realizar transacciones en la ventana de *Purchase Order*, *Goods Receipt* y en la de *Purchase Invoice*.

En la primera ventana, para los proveedores de las organizaciones F&B US y F&B España, se va a realizar una orden de compra con día de creación el día en el que se ejecute el test y con fecha final una semana posterior.

Los proveedores de estas organizaciones son:

F&B US:

- Fruit & Bio is Life, Inc.
- Happy Drinks, Inc.
- Be Soft Drinker, Inc.

F&B España:

- La fruta es la Vida, S.L
- Bebidas Alegres, S.L
- Refrescos Naturales, S.A

Para cada uno de estos proveedores se tiene que crear un nuevo registro en la pestaña de la cabecera y a su vez crear una nueva línea para cada producto del que se va a hacer la orden de compra.

La segunda ventana, se trata de una recepción de la mercancía en la que también habrá que crear un registro en la cabecera para cada uno de los proveedores listados anteriormente y a su vez las líneas necesarias para registrar los productos que se han van a recibir.

En la tercera ventana se va a crear la factura de la compra. En esta ventana se creará también un registro para la pestaña de la cabecera y las líneas correspondientes a los productos. En este registro se va a poner como día de cobro de la factura el último día del mes en el que se registró el pedido de compra.

En cada una de las ventanas se tiene que cerrar todos los procesos para confirmar que se quiere realizar esa transacción.

Por último, hay que registrar una factura de compra para unos proveedores adicionales. Estos proveedores adicionales son:

Para F&B US:

- Electric Power Company, Inc.
- Hope Building, Co.
- Star Mobile Company, Inc.
- IT Security Services, Co.

Para F&B España:

- Telefonía SGT, SA
- SI Rapid, SA
- Servicios Inmobiliarios Los mejores, SA
- Generación Eléctrica, SA

Para estos proveedores solo hay que realizar la factura de compra. En esta ocasión la fecha de cobro de la factura será distinta a la de los anteriores proveedores, poniendo como fecha de pago de la factura el último día del mes, pero con fecha final de la factura el día 3 del siguiente mes para hacer que la factura esté retrasada en cuanto al pago.

Flujo de Pago

En este flujo tres proveedores tienen como plazo de pago "inmediato", esto significa que el pago se tendrá que realizar en la misma fecha que la fecha de pago de la factura de compra. El resto de proveedores tienen como plazo de pago "30 días", "60 días" y "90 días".

Esto significa que, si se está realizando la transacción en abril y tiene como plazo de pago "30 días", el pago se debería de realizar en mayo.

Esta comprobación se tiene que hacer para cada proveedor. Para hacer esto hay que ir a la ventana *Payment Proposal* y añadir un registro en la cabecera indicando el proveedor del que se quiere realizar el pago y la fecha en la que se tiene que realizar. Para finalizar, solo hay que darle al botón *Select Expected Payments* para que se efectúen los pagos de las facturas que estén pendientes de pago para ese mes.

Además, hay que registrar una entrada de salarios en la pestaña *transactions* de la ventana *financial account*.

Para terminar, hay que asignar el coste de cada uno de los servicios de los proveedores y el salario a cada una de las organizaciones de F&B US y F&B España. Esto se hace mediante la ventana *G/L Journal*. En esta ventana hay que crear una cabecera indicando la organización y la fecha y un registro en la pestaña *Lines* para cada servicio, indicando a qué organización pertenece separando la cantidad en crédito o en débito.

Flujo de venta

En este flujo se debe realizar una orden de venta, una salida de mercancía y una factura de venta para los siguientes proveedores:

F&B USA:

- Sleep Well Hotels
- Healthy Food Supermarket
- Moon Light Restaurants

F&B España:

- Alimentos y supermercados, SA
- Hoteles Buenas Noches, SA
- Restaurantes Luna Llena, SA.

Este flujo seguiría la misma estructura que el flujo de compra, habría que crear un registro en la cabecera de cada ventana y rellenar las líneas con los productos deseados. Al terminar cada registro, habría que completar cada proceso. El día de pago tiene que ser igual que el día de la factura de venta.

Además, en este flujo se va realizar también la creación de los cobros. Una vez que se ha efectuado un pedido de venta o una transacción de producto, hace falta crear los cobros para esas facturas.

Amortización

Después, hay que procesar la amortización del año. Este paso solo se tiene que hacer si es el último mes del año. Las empresas suelen tener objetos que cuestan dinero como un coche de empresa o un ordenador, pero a lo largo del tiempo el valor que tenía dicho objeto va disminuyendo y eso hay que registrarlo en este flujo.

Reconciliación

Todas las transacciones que se han creado en los pasos anteriores necesitan ser verificados con las cuentas que tiene el banco. Para eso, se utiliza este flujo, al final el precio total de la transacción tiene que ser igual que el precio total que marca el banco. Si esto es así, se creará la reconciliación y todo estará en orden.

Mantenimiento Inventario

Cuando se crean todos estos flujos, la cantidad de algunos productos va disminuyendo por lo que es necesario actualizar el inventario añadiendo más productos para que la cantidad de estos sea estable y se puedan seguir creando transacciones.

Una vez que ya se ha reproducido a mano esta serie de flujos funcionales y ya se conoce cuáles son los campos que se tienen que rellenar y los parámetros necesarios, se puede empezar a automatizar el proyecto.

Como he comentado antes, Openbravo tiene una API muy extensa para realizar test automáticos y hay mucha infraestructura ya creada. Aun así, para este trabajo, he tenido que crear nueva infraestructura que no tenían para poder realizar los test automáticos.

Por ejemplo, en Openbravo no había una infraestructura creada para la ventana *G/L Journal* y era necesario crearla. Para ello, se tiene que analizar el comportamiento que tiene la ventana en la aplicación para poder programar las acciones que realiza.

Además de esto, para la última parte del proyecto, Openbravo no soportaba la ejecución periódica de los tests parametrizados, por lo que se tuvo que crear una nueva infraestructura para que el flujo pudiera repetirse tres veces cambiando los valores de las fechas con un parámetro.

Una vez que el flujo ya está automatizado y se ejecuta correctamente dando un buen resultado, solo faltaría hacer *push* a PI. De esta forma, se introduce en el proceso de integración continua en donde se le somete a una serie de tests para validarlo y verificar que pasa las pruebas de calidad exigidas por la empresa.

Dentro de la integración continua hay una tarea que solo se activa en cada nueva versión de la aplicación y es ahí donde se encuentra el nuevo test que he creado. Esta tarea se encarga de promocionar el ERP de Openbravo crear el producto final.

4. Desarrollo del proyecto

En este capítulo se van a explicar las herramientas que se han utilizado para la realización de este proyecto como también la forma adecuada para crear el entorno de Selenium, la automatización y verificación del flujo funcional.

4.1 Herramientas Tecnológicas

Como he dicho antes, en este apartado se van a explicar las distintas herramientas tecnológicas empleadas.

4.1.1 Ant

Apache Ant es una herramienta de línea de comandos usada para la realización de tareas mecánicas y de repetición utilizadas sobre todo a la hora de compila. Se puede decir que sirve para automatizar los procesos de compilación desarrollados en Java. Además, una de las grandes ventajas de esta herramienta es que se trata de una herramienta multi-plataforma que no depende del sistema operativo ya que se basa en los archivos XML para la realización de las tareas.

En este trabajo, ant, se utiliza para realizar una serie de tareas de compilación, en la mayoría de los casos solo es necesario utilizar 3 tareas que son muy importantes:

- *ant smartbuild*

- *ant install.source*

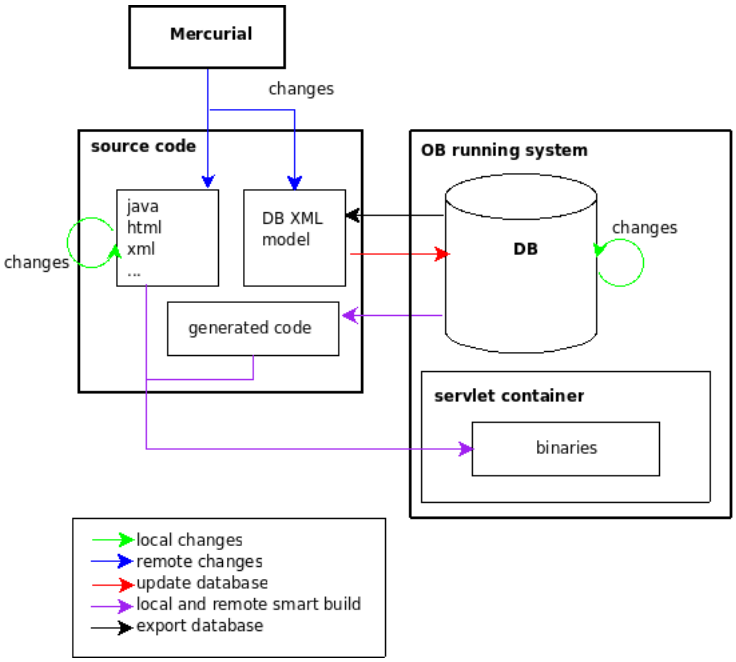
- *ant export.database*

La tarea principal para el proceso estándar es *ant smartbuild* realiza todos los procesos requeridos que se explican debajo. Esta tarea acepta dos propiedades opcionales:

- *local* para desarrollos locales o remotos, por defecto está en *yes*.

- *restart* indicando si después del proceso de *build*, Tomcat, debe reiniciarse con el valor por defecto puesto a no.

La diferencia entre local y remoto se puede ver en la imagen de debajo. Los desarrollos locales son los cambios que hace el desarrollador mientras que los desarrollos remotos son cambios hechos por otro desarrollador.



(Imagen 4.1)

Una de las tareas más utilizadas y más importantes es *ant install.source*. Este comando crea la base de datos, inserta los datos de muestra dentro de la aplicación y además compila y despliega la aplicación de acuerdo con el modo de despliegue elegido.

Una vez que los desarrollos están listos para ser testeados, hay que compilar la aplicación, pero no hace falta hacer una compilación completa, solo de los nuevos cambios que aparezcan. Esta tarea se consigue hacer con el comando *ant smartbuild.*

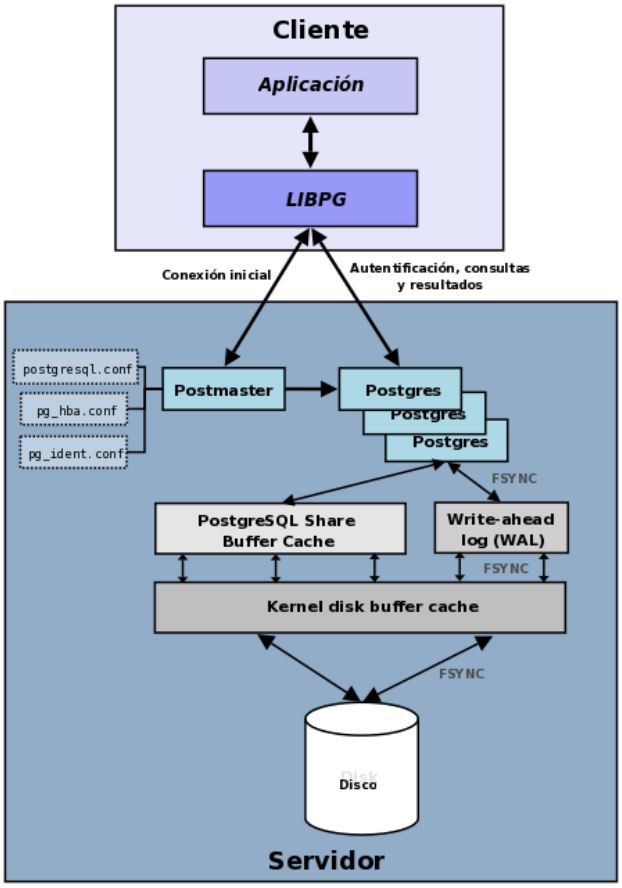
En muchas ocasiones cuando se desarrolla, se producen cambios en la base de datos. Para poder exportar estos cambios en el desarrollo hay que utilizar el comando *ant export.database.* [4]

4.1.2 PostgreSQL

PostgreSQL es un sistema de gestión de bases de datos objeto-relacional, distribuido bajo licencia BSD y con su código fuente disponible libremente.

PostgreSQL utiliza un modelo cliente/servidor y utiliza multiprocesos en vez de multihilos para garantizar la estabilidad del sistema. De esta forma un fallo en un proceso no afectará al resto y el sistema podrá seguir funcionando.

En la imagen siguiente se muestra una ilustración de los componentes más importantes de un sistema PostgreSQL.



(Imagen 4.2)

- Aplicación cliente: es la aplicación cliente que utiliza PostgreSQL para administrar la base datos, la comunicación que se establece se puede hacer por TCP/IP o por sockets locales.

  Postmaster: Este proceso es el encargado de escuchar las conexiones entrantes por medio de un puerto/socket. También se encarga de crear los procesos para autentificar las peticiones.

- Ficheros de configuración: los tres ficheros más importantes de configuración postgresql.conf, pg_hba.conf, pg_ident.conf.

- Procesos hijos: se encargan de autentificar a los clientes, de gestionar las consultas y mandar los resultados a las aplicaciones clientes.

- PostgreSQL share buffer cache: memoria donde se guardan los datos en cache.

- *Write-Ahead* Log: se encarga de asegurar la integridad de los datos.

- *Kernel disk buffer cache*: cacheé del disco del sistema operativo

- Disco: lugar donde se almacenan los datos y toda la información para que PostgreSQL funcione. [5]

4.1.3 Mercurial

Mercurial es un sistema de control de versiones distribuido libre, gratuito y similar a Git o Bazaar. Todos ellos rompen con el modelo tradicional de trabajo cliente/servidor de los repositorios como CVS o Subversión.

En los repositorios tradicionales existe una clara diferencia entre los conceptos servidor y cliente: el servidor es el que mantiene y controla el versionado de los ficheros, y el cliente se descarga copias del servidor para que sean modificadas y subidas posteriormente con el fin de que sean vistas por el resto de usuarios. En

cambio, en los repositorios distribuidos los clientes ya no se descargan una copia del repositorio, sino que lo clonan, comportándose como cliente y servidor al mismo tiempo. Es decir, que la copia podría comportarse a su vez como servidor ante otros clientes (construyendo así una estructura distribuida).

A pesar de que la estructura y gestión es algo más compleja que la de un SCM tradicional, existen situaciones en las que los repositorios distribuidos son muy útiles. Por ejemplo, en proyectos donde los desarrolladores se encuentren continuamente incomunicados y necesitan realizar acciones tan habituales como commit (locales), branch, etc …

Además de los comandos tradicionales que se pueden ejecutar sobre un sistema de control de versiones, los repositorios distribuidos añaden nuevos como: *push* y pull. Estos dos comandos permiten a un usuario subir los cambios locales a un servidor remoto y actualizarse con los cambios realizados por otros usuarios respectivamente. [6]

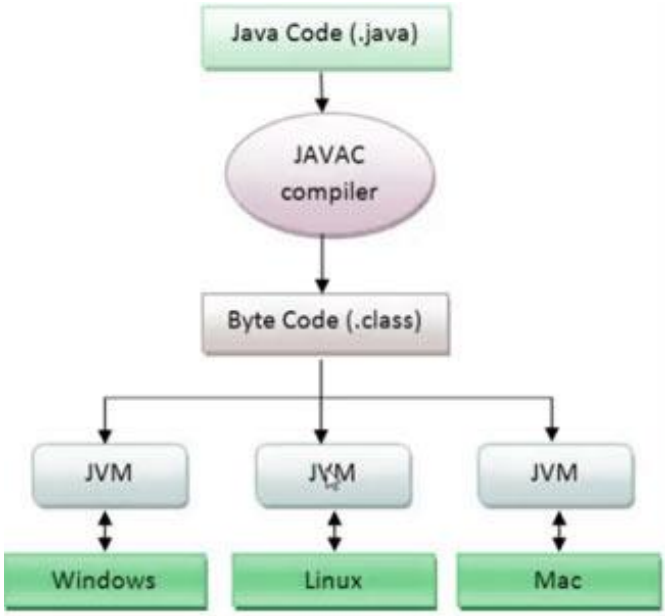
(Imagen 4.3)

4.1.4 Java

Java es un lenguaje de programación orientado a objetos que se ejecuta en una gran cantidad de plataformas con distintos sistemas operativos, como, por ejemplo, Mac OS, Linux, Windows… Para que esto sea posible, Java utiliza el JRE que se trata de un entorno de ejecución para los programas compilados. Gracias a este entorno un programa puede ejecutarse en sistemas operativos distintos.

Los programas Java se compilan en un lenguaje intermedio que se denomina Bytecode. El código se interpreta gracias al JRE para conseguir la portabilidad. El JRE es un paso intermedio entre el sistema operativo y el código Bytecode.

Esta imagen muestra un esquema de lo explicado:



[7] (Imagen 4.4)

Hay muchos entornos para desarrollar aplicaciones en Java. Estas herramientas ayudan a los programadores a desarrollar de una forma mucho más cómoda y más rápida. Estos productos se llaman IDE (*Integrated Development Enviroment*).

En Java se sigue el proceso: edición del código fuente, compilación y ejecución. Los programas Java se desarrollan y se compilan para obtener un código que será interpretado por el JRE.

La edición del programa fuente se realiza escribiendo el programa Java en un editor de texto o utilizando el entorno integrado de desarrollo. El código fuente se almacena en un fichero del tipo .java.

La fase de compilación se puede realizar con el IDE o con el compilador de Java. En este proceso se verifica que el código es correcto a nivel léxico, sintáctico y semántico en Java. Si no aparece ningún error se genera un fichero del tipo class que está en Bytecode. Cuando ya acaba la fase de compilación ya se puede ejecutar el programa.

4.1.5 Eclipse

Eclipse es una plataforma de software constituido por un conjunto de herramientas de código abierto multiplataforma. Esta herramienta se utiliza principalmente para el desarrollo de entornos de desarrollo integrados, por ejemplo, el IDE de java.

Entre sus principales características cabe destacar:

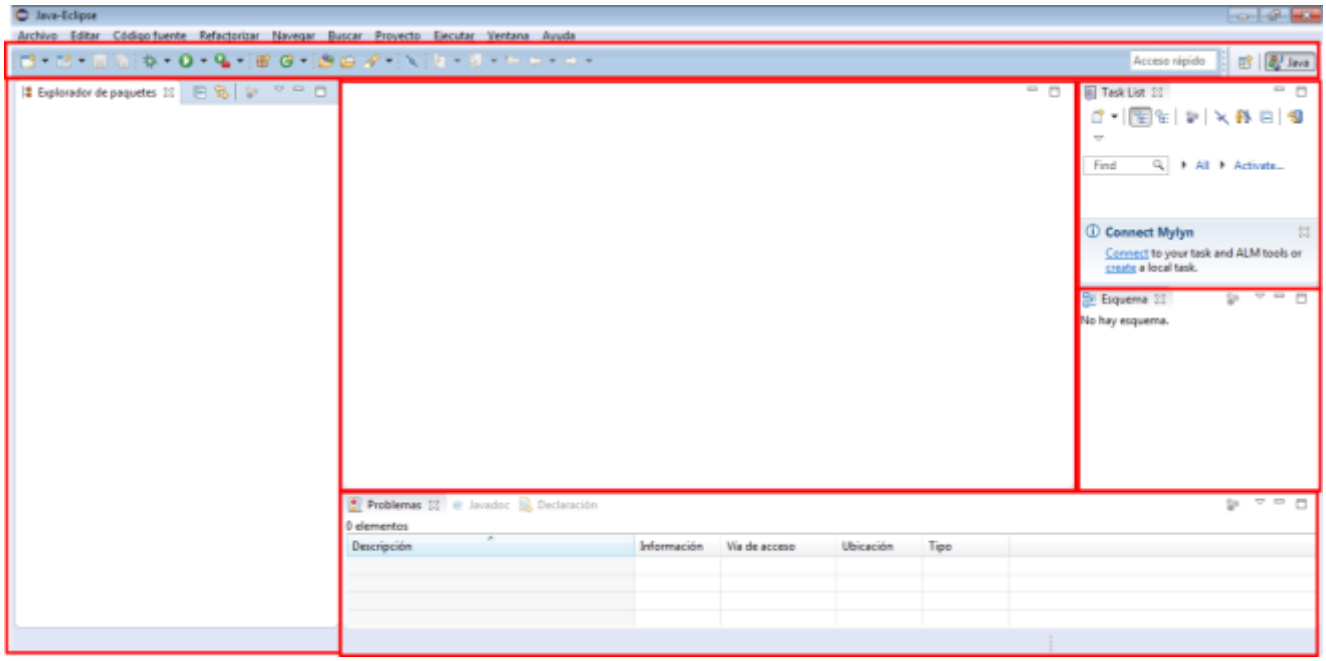
- Perspectivas, editores y vistas: En Eclipse todas las ventanas y perspectivas están relacionadas entre sí para poder trabajar de una forma mucho más rápida y eficaz

- Gestión de proyectos: En esta herramienta se encuentran los proyectos que son el conjunto de recursos que se relaciona entre sí.

  El IDE proporciona ayudas para la creación de proyectos. Cuando se crea uno se abre la perspectiva adecuada al tipo de proyecto con la colección de vistas, editores y ventanas configuradas por defecto.

- Depurador de código: se utiliza un depurador muy útil para mejorar el código. Para ello solo hay que ejecutar el programa con el modo depuración.

- Colección de *plug-ins*: Existe una gran cantidad de *plug-ins* desarrollados, unos publicados por Eclipse y otros desarrollados por terceros. Hay tanto gratuitos como de pago.



(Imagen 4.5)

4.1.6 Tomcat

Apache Tomcat (también llamado Jakarta Tomcat o simplemente Tomcat) funciona como un contenedor de servlets desarrollado bajo el proyecto Jakarta en la Apache Software Foundation. Tomcat implementa las especificaciones de los servlets y de JavaServer Pages (JSP) de Oracle Corporation (aunque creado por Sun Microsystems).

Tomcat es desarrollado y actualizado por miembros de la Apache Software Foundation y voluntarios independientes. Los usuarios disponen de libre acceso a su código fuente y a su forma binaria en los términos establecidos en la *Apache Software License*.

Las primeras distribuciones de Tomcat fueron las versiones 3.0.x. Las versiones más recientes son las 8.x, que implementan las especificaciones de Servlet 3.0 y de JSP 2.2. A partir de la versión 4.0, Jakarta Tomcat utiliza el contenedor de servlets Catalina.

Tomcat es un contenedor web con soporte de servlets y JSPs. Tomcat no es un servidor de aplicaciones, como JBoss o JOnAS. Incluye el compilador Jasper, que compila JSPs convirtiéndolas en servlets. El motor de servlets de Tomcat a menudo se presenta en combinación con el servidor web Apache.

Tomcat puede funcionar como servidor web por sí mismo. En sus inicios, existió la percepción de que el uso de Tomcat de forma autónoma era sólo recomendable para entornos de desarrollo y entornos con requisitos mínimos de velocidad y gestión de transacciones. Hoy en día ya no existe esa percepción y Tomcat es usado como servidor web autónomo en entornos con alto nivel de tráfico y alta disponibilidad.

Dado que Tomcat fue escrito en Java, funciona en cualquier sistema operativo que disponga de la máquina virtual Java.

[8] (Imagen 4.6)

4.1.7 Selenium

Selenium es un entorno de pruebas de software para aplicaciones basadas en la web. Selenium provee una herramienta de grabar/reproducir para crear pruebas sin usar un lenguaje de scripting para pruebas (Selenium IDE). Incluye también un lenguaje específico de dominio para pruebas (Selanese) para escribir pruebas en un amplio número de lenguajes de programación populares
Incluyendo Java, C#, Ruby, Groovy, Perl, Php y Python.

Las pruebas pueden ejecutarse entonces usando la mayoría de los navegadores web modernos en diferentes sistemas operativos como Windows, Linux y OSX. (Wikipedia. Selenium. https://es.wikipedia.org/wiki/Selenium)

Existen cuatro componentes principales en Selenium:

- Selenium IDE: se trata de una extensión de Firefox que permite grabar, editar y reproducir test automáticos.
- Selenium Remote Control: permite escribir tests en cualquier lenguaje de programación que pueda realizar peticiones HTTP.
- Selenium Core: es el framework de ejecución de javascript que finalmente interacciona con el navegador. Está incluido en Selenium Remote Control.
- Selenium Grid: permite la ejecución de paralela de tests de Selenium.

[9]

(Imagen 4.7)

4.2 Creación entorno Selenium

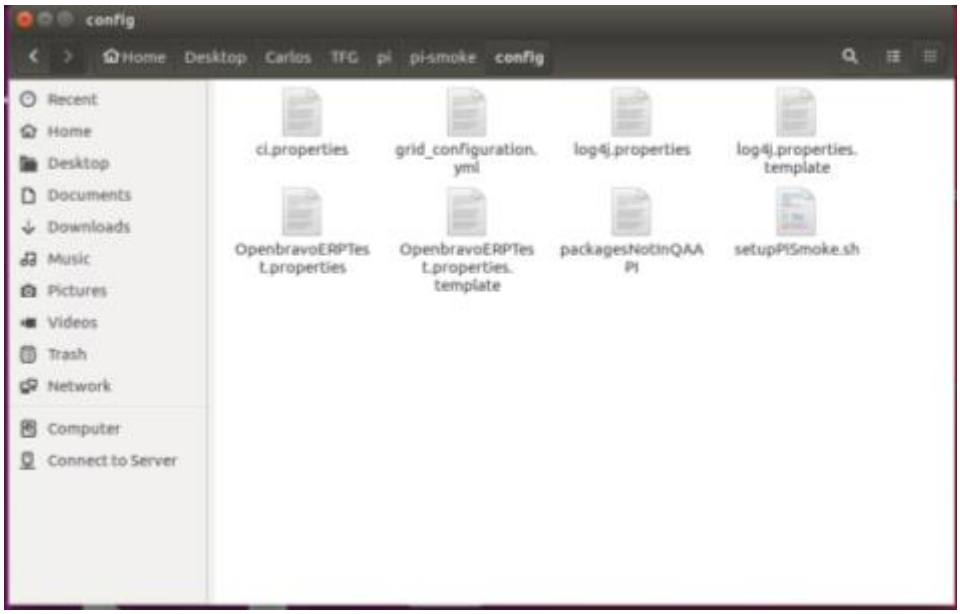
Lo primero para poder realizar la automatización de la inserción de los nuevos datos dentro del ERP de Openbravo es montar un entorno de Selenium. En este apartado voy a enseñar cómo crear un entorno nuevo.

La rama estable de test en la que se van incluyendo los desarrollos de nuevos tests es PI-SMOKE y la URL para poder hacer un clon y empezar a desarrollar es https://code.openbravo.com/tools/automation/pi-smoke/.

Para poder hacer un clon del repositorio, solo hay que utilizar el comando *hg clone*, además para poder hacerlo funcional tenemos que hacer una copia de los ficheros:

- pi-smoke/config/OpenbravoERPTest.properties.template
- pi-smoke/config/log4j.properties.template
- pi-smoke/.classpath.template

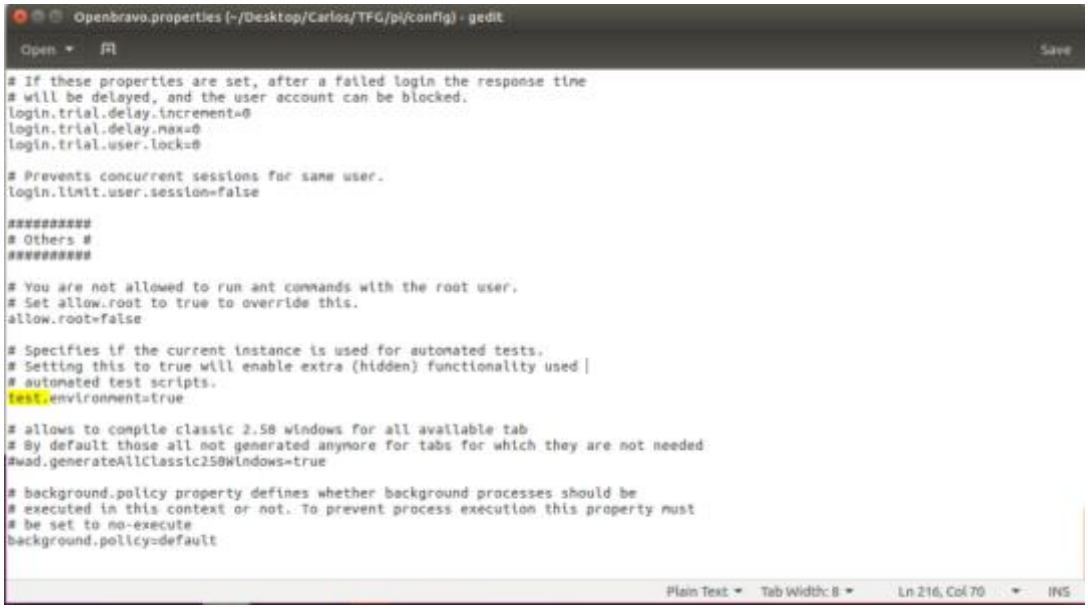
y eliminar la extensión .template.



(Imagen 4.8)

(Imagen 4.9)

Después de hacer la copia, se puede importar este proyecto junto al resto de proyectos en un entorno de Openbravo. Gracias al *.classpath*, lee automáticamente las librerías que tiene que importar. Es importante cambiar la variable *test.environment* a *true* en el archivo *Openbravo.properties* para poder ejecutar test automáticos. Además, en el fichero *OpenbravoERPTest.properties* hay que configurar la variable *test.videorecord* y ponerla con valor *"t"* para que cuando ejecutemos un test en local, si se produce un error, se pueda mirar dentro de la carpeta *last-utils* el vídeo que ha sido grabado durante la ejecución del test. De esta forma, se podría ver de una manera muy rápida dónde ha habido un error.

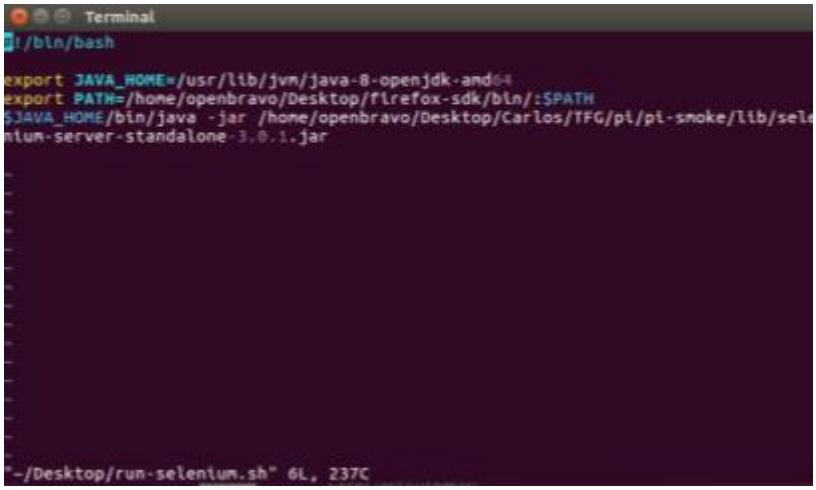


(Imagen 4.10)

(Imagen 4.11)

Una vez tenemos el entorno para automatizar un test, se necesitan una serie de requisitos. El entorno de Openbravo tiene que ser una instancia limpia sin ningún módulo extra. Además se necesitan las versiones de Firefox 45.7.0.esr, Java 8 y Selenium-server-standalone-3.0.1.

Antes de ejecutar un test hay que ejecutar la librería de Selenium e indicarle que utilice esa versión. Para eso hay que lanzar un pequeño script para decirle que exporte la variable de JAVA_HOME y para decirle que utilice java 8, exportar la variable PATH para que escoja la versión de Firefox 45.7.0.esr y ejecutar indicándole la ruta de nuestra librería de Selenium.



(Imagen 4.12)

A partir de aquí, ya se podría ejecutar un test. Para lanzar un test solo habría que ir a Eclipse seleccionar un test como por ejemplo QAAdminUser.java y hacer clic derecho sobre la clase e ir *run us* > *JUnit test.* La librería de Selenium va a crear una instancia, va a abrir Firefox y va a ejecutar el test.

4.3 Automatización del flujo

Para automatizar los flujos funcionales que se han explicado en el tercer apartado se ha utilizado las librerías de Selenium que tiene Openbravo. Una vez realizado el flujo funcional a mano, solo queda automatizarlo con esta herramienta. Para este trabajo se han creado una serie de clases nuevas para aportar una nueva funcionalidad al framework de Selenium que tiene openbravo.

Las clases que se han utilizado para este trabajo son:

- ProcurementFlow.java
- PaymentInFlow.java
- SalesFlow.java
- GLJournal.java
- GLJournalData.java
- GLJournalHeaderData.java
- GLJournalHeaderTab.java
- GLJournalLinesData.java
- GLJournalLinesTab.java
- GLJournalTab.java
- GLJournalWindow.java
- Amortization.java

- AmortizationHeaderTab.java
- AmortizationWindow.java
- AmortizationFlow.java
- FinancialAccountFlow.java
- PeriodsFlow.java
- PhysicalInventoryFlow.java
- TFGTestSuit.java

No existe una relación entre estas clases, por lo que se trata de una estructura básicamente horizontal, sin embargo, varias de ellas *extienden* de algunas más genéricas para poder invocar sus métodos. Todas las clases acabadas en *Flow extienden* de una clase llamada *OpenbravoERPTest.* Esta clase tiene todos los métodos para poder realizar un test con Selenium, por lo que todas las clases de flujos funcionales que tienen que ser ejecutadas *extienden* de esta.

A su vez, hay otras clases como las acabadas en *Data* que *extienden* de una clase más genérica, se trata de la clase *DataObject.* Esta clase sirve para poder generar objetos más customizados de este tipo, dependiendo de las necesidades que se tengan.

Las clases que acaban en *Tab extienden* de *GeneratedTab.* Todas las pestañas que se crean de Openbravo *extienden* de esta superior para poder beneficiarse de sus métodos y así poder actuar como lo que son, dependiendo de la funcionalidad que tengan, utilizaran más o menos métodos.

También existen las clases acabadas en *Window* que *extienden* de *StandardWindow* y cómo intuitivamente se puede averiguar de su nombre, tiene los métodos necesarios para poder crear las ventanas y describir cómo van a ser.

Por último, solo queda mencionar las clases que se nombran igual que las ventanas dentro del ERP, por ejemplo, *G/L Journal.* Estas clases *extienden* de *SmokeTabScript* e implementan *SmokeWindowScript*. A diferencia de la clase anterior, estas clases tienen los métodos para poder abrir una pestaña/ventana, para poderla cerrar, para seleccionarlas, en definitiva, para poder interactuar con ellas.

Especialmente, en este trabajo no se necesita profundizar mucho sobre estas clases tan genéricas, ya que gracias a la infraestructura que tiene Openbravo de Selenium puedes abstraerte de cómo están estas clases por debajo ya que los nombres de los métodos de las clases son muy intuitivos.

Una vez que ya se tiene una visión general de las clases que se han utilizado para realizar la automatización de los flujos voy a explicar cuál es la función de cada uno de las clases y sus métodos. Los datos nuevos que se van a insertar dentro del ERP de Openbravo son datos elegidos por la empresa, no son relevantes en lo que a nivel funcional se refiere, por lo que solo explicaré cómo funcionan las clases y el resultado que producen sin profundizar en los datos en cuestión.

ProcurementFlow.java

En esta clase se realiza la automatización del flujo de compra. Para eso se deben realizar tres pasos en el ERP de Openbravo.

El primero es crear un pedido de compra. Para eso, lo primero que hay que hacer es declarar las variables que vamos a utilizar en la ventana de *Purchase Order*.

Los objetos que se van a utilizar serán del tipo: *PurchaseOrderHeaderData*, *PurchaseOrderLinesData* y *PurchaseOrder*. Como he explicado en el capítulo tres, los atributos, los parámetros del constructor y la inicialización de los atributos tienen que estar en el mismo orden para que los datos puedan sustituirse en el lugar adecuado.

Para crear un pedido de compra hace falta tener un objeto *PurchaseOrder* para poder interaccionar con la ventana; después, hay que introducir los datos en cada uno de los campos. Para poder hacer esto, el objeto *PurchaseOrder* tiene un método llamado *create*() que, cogiendo un objeto *PurchaseOrderHeaderData* como parámetro, crea un nuevo registro e introduce en los campos los valores que tiene el objeto *PurchaseOrderHeaderData*.

Una vez que la cabecera ha sido creada con esos valores, se valida que no ha habido ningún problema a la hora de guardar el registro y que los valores que no han sido introducidos de esta forma son los esperados. Esto se consigue utilizando los métodos assertSaved() y assertData() que tiene el objeto *PurchaseOrder*. El método *assertData*() necesita como parámetro un objeto del tipo *PurchaseOrderHeaderData* en el que se encontrarán el valor de los campos que se van a validar. En el caso de que el valor de los campos no sea el deseado, dará un error en la línea del *assertData*() avisando que no tienen el valor esperado.

Una vez que la cabecera ha sido creada, se da paso a introducir los productos que se quieren comprar en la pestaña *Lines*. Esto sigue una estructura similar a la inserción de los datos en la cabecera. Primero, hay que crear un objeto para poder interactuar con esta pestaña. Para esto se instancia un objeto del tipo *Lines*.

Lines se trata de una clase interna en la clase *PurchaseOrder*, por lo que se le puede llamar sin ningún problema. Una vez que ya tenemos este objeto, solo hay que seguir los mismos pasos que en la cabecera, hay que llamar al método *create*() y pasarle como parámetro un objeto *PurchaseOrderLinesData*. Como ahora estamos en la pestaña de *Lines*, hace falta que el objeto sea de este tipo para que se pueda rellenar en los campos necesarios. Una vez que se termina de rellenar los campos solo queda validar estos datos de la misma forma que en la cabecera y procesar el documento utilizando el método *book*() del objeto *PurchaseOrder*. Como el botón *Book* forma parte de la ventana el objeto *PurchaseOrder* es el que tiene el método para cerrar el documento.

Antes de pasar a crear la llegada del producto es necesario guardar en un *String* en un formato "%S-%S-%S", el número de documento, el día del pedido y el importe total bruto ya que se necesita para crear las líneas de una llegada de producto.

El siguiente paso es realizar un *Goods Receipt* para eso hay que crear un objeto del tipo *GoodsReceipt* para poder utilizar esta ventana. Una vez que tenemos el objeto, hay que rellenar los campos de la cabecera utilizando el método *create*() del objeto *GoodsReceipt* y después validarlos. Una vez que la cabecera está terminada solo queda insertar las líneas. En esta ocasión se utiliza el botón *Create Lines From* de la ventana *Goods Receipt*, este botón copia las líneas con los productos de la ventana *Purchase Order* y las copia a las líneas de la pestaña *Lines* de la ventana *Goods Receipt*.

Para hacer esto, hace falta utilizar el método *createLinesFrom*() del objeto *GoodsReceipt*. Este método necesita dos parámetros, uno es el indicador del almacén y el otro es el *String* que se ha guardado antes en la ventana *Purchase Order*. Con esto, conseguimos crear la pestaña *Lines* de una forma más rápida.

Para terminar este flujo de compra, solo falta crear una factura de venta. Para esto se sigue una metodología similar a la anterior, habría que crear la cabecera de la ventana *Purchase Invoice* con el método *create*() introduciéndole un objeto tipo *Data* para que rellene los campos y por ultimo añadir las líneas, como hemos hecho antes con el comando *createLinesFrom*().

Dentro de la clase de este flujo existen otros métodos internos pero que no serán comentados para no ser redundante.

PaymentOutFlow.java

En esta clase, se realizan dos pasos. El primero es crear los pagos de los proveedores, para ello hay que ir a la ventana *Payment Proposal* y seguir los pasos que hemos realizado antes. Habría que crear una cabecera, insertar los datos, crear las líneas y completar el documento. De esta forma se crean los pagos de las facturas de compra para el mes correspondiente.

Para realizar el segundo paso se ha necesitado crear varias clases nuevas ya que la ventana *G/L Journal* no estaba en el framework de Selenium de Openbravo y era necesaria construirla. Para poder utilizar esta ventana se han tenido que crear las clases: *GLJournal.java*, *GLJournalData.java*, *GLJournalHeaderData.java*, *GLJournalHeadertab.java, GLJournalLinesData.java, GLJournalLinesTab.java* y *GLJournalTab.java*. Antes de seguir explicando este segundo paso comentaré en qué consiste cada una de estas clases para poder entender cómo interactuar con la ventana *G/L Journal*

- *GLJournal.java*: esta es la clase que va a representar a la ventana *G/L Journal* y para eso se crean los métodos propios que tendría una ventana como el *open*() para poder abrir la ventana desde el menú principal y el *selectTab*() para seleccionar la pestaña. Además, como esta ventana tiene dos pestañas por debajo, es necesario crear dos clases internas con los mismos métodos que la clase principal. La segunda pestaña, como genera documentos, tendrá un método adicional llamado *complete*() para poder completar los documentos.

En esta serie de clases que he comentado se encuentran las que acaban con la coletilla *Data*, como son parecidas y siguen la misma estructura explicaré solo la clase *GLJournalHeaderData.java* por no repetir información.

- *GLJournalData.java:* En este tipo de clases se va a *mapear* el valor de los campos. Esta clase es la que se encarga de asociar cada valor que le introducimos a un campo de cada ventana. En concreto, *mapea* los valores de los campos de la cabecera de la ventana *G/L Journal*. Al tener un Builder dentro con un mapa se puede crear un objeto tipo *Builder* y llamar a sus métodos para insertarle un valor para que lo asocie con un campo de la ventana.

El resto de clases acabadas en *Data* tienen el mismo comportamiento, solo que se utilizan para otras pestañas. Gracias al nombre que se les ha asignado se puede saber a cuáles.

Al igual que ocurre con las clases acabadas en *Data*, pasa también con las clases acabadas en *Tab*, por ello explicaré una y el resto seguirán la misma estructura.

- *GLJournalTab.java*: Esta clase sirve para identificar la ventana y la pestaña sobre la que se va a actuar. Básicamente, se especifica el identificador para saber qué ventana es, el nombre de la pestaña, y el nivel. Si hay algún botón en esa pestaña se especifica dónde se encuentra mirando en el HTML de la aplicación. Además se crea un método para que haga la función que debe de cumplir. En esta clase en concreto no hay, pero en la clase *GLJournalHeader* se encuentra el botón de completar y un método *complete*() para que complete un registro. Por último, solo queda decir que en el constructor hay que instanciar el botón y, en el caso de que haya alguna pestaña en un nivel inferior, utilizar el método *addChildTab*() para añadirla.

- *GLJournalWindow.java*: Esta clase es de la que implementa *GLJournal.Java* para poder utilizar los métodos específicos para poder definir las acciones que tiene esa ventana en concreto. Además, se concreta cuál es la ruta que hay que seguir para ir a ella desde la ventana principal de la aplicación.

Una vez comentada la función de cada una de estas clases, para poder interactuar con la ventana *G/L Journal* volvamos a la explicación de esta segunda parte del flujo.

En este último paso, hay que asignar el coste de cada uno de los proveedores a las organizaciones F&B US y F&B Spain. Para esto lo primero que hay que hacer es abrir la nueva ventana que hemos creado con el método *open*(); después, hay que crear una cabecera, con los datos deseados. Para eso, se utilizan los métodos ya explicados con anterioridad para insertar, guardar y validar datos. Una vez terminada la cabecera hay que crear unas líneas para poder separar la cantidad de los proveedores entre cada una de las organizaciones. Una vez finalizado el proceso hay que completar el documento con el método *complete*().

SalesFlow.java

Esta clase es muy parecida a la clase *ProcurementFlow.java* ya que ambos se tratan de flujos muy similares en este caso, en vez de ser un flujo de compra, es un flujo de venta.

Igual que en *ProcurementFlow.java*, en esta clase también va a haber tres pasos.

El primero es crear un envío. Para eso, se necesita utilizar la ventana *Sales Order*. Como el resto de ventanas del framework de Selenium de Openbravo va a tener una serie de métodos para poder interactuar con ella. Se puede ver que para todas las ventanas los métodos son los mismos, lo que hace muy intuitiva su automatización. El primer paso será abrir la ventana *Sales order* utilizando el método *open*() del objeto *SalesOrder;* después, se creará la cabecera y se insertarán los valores deseados en los campos utilizando el método *create*() de la cabecera; además, como siempre hay que hacer, se utilizan los métodos *assertSaved*() y *assertData*() para validar que se ha guardado bien el registro y que el valor de los campos que se rellenan automáticamente son los que se esperan. Cuando ya se ha terminado con la cabecera, se crea la pestaña *Lines* y se introducen los productos. Esto se hace de la misma forma cuando insertamos líneas en el resto de los flujos. Una vez que se han acabado de insertar los productos con los precios, las cantidades y otros datos, se procesa el documento utilizando el método *book*() de la cabecera.

Al igual que en el flujo de compra, aquí también hay que guardarse en un *String* el número de documento, la cantidad total y la fecha de la factura, ya que la vamos a necesitar para poder realizar la siguiente parte.

El segundo paso es crear un envío de mercancía. Para eso hay que crear un objeto *GoodsShipment* igual que en el resto de ventanas. Después de construir la cabecera hay que crear las líneas, esta vez es necesario utilizar el botón *Create Lines From* de la cabecera de la ventana. Para eso el objeto *GoodsShipment* tiene el método *createLinesFrom*(), a este método hay que pasarle como parámetros el identificador del almacén y el *String* que hemos guardado en el anterior paso. De esta forma, se rellenará la pestaña de *Lines* con los productos del pedido.

Antes de introducir el *String* dentro del método *createLinesFrom*() se ha tenido que tratar este *String* ya que, debido a un problema que tiene la empresa en el código, se eliminan los decimales y se mantiene la coma de los miles y es necesario mantener los decimales y eliminar la coma para que la ventana pueda reconocer que líneas tiene que copiar. Para eso se ha utilizado el método propio de esta clase *tratarTotalGroosAmount*(), lo que permite eliminar la coma de los miles de la cantidad.

Lo último de este paso sería completar el documento con el método *complete*() del objeto *GoodsShipment* y verificar que se ha completado correctamente con el método *assertProcessCompletedSuccessfully2*().

En el tercer paso hay que crear una factura de venta. Para eso como en los anteriores casos, se crea la ventana *Sales Order* y se rellena la cabecera con los datos deseados. Una vez que se ha creado la cabecera, se insertan las líneas utilizando el ya mencionado método *createLinesFrom*(). En este caso solo necesita que se le introduzca el *String* de los pasos anteriores, tratado con el método *tratarTotalGrossAmount*(). Una vez que se han creado las líneas hay que completar el documento con el método *complete*().

Además de estos tres pasos que crean el flujo de venta hay un paso adicional para poder realizar el flujo para cobrar. Al final, la ventana *Sales Invoice* tiene el botón *addPayment* para poder realizar los cobros de una forma muy rápida. Es por eso que en esta clase se ha añadido este paso extra de crear los pagos entrantes. Como acabo de decir la ventana *Sales Invoice* es la que tiene el botón, por lo tanto, una vez que se ha completado la factura de venta, hay que utilizar el método *addPayment2*() del objeto *SalesInvoice* modificado por mí, ya que el método *addPayment*() de la infraestructura de Selenium de Openbravo no funcionaba como debería. Este método pulsa el botón de *addPayment,* abre su popup e introduce la cantidad del cobro, la fecha y la acción que se debe desempeñar. Una vez que inserta estos valores le da al botón *Done* y concluye habiendo creado el cobro.

El método *addPayment*() solo insertaba la cantidad y la acción esperada. Por eso, para poder realizar el flujo tal y como lo pedía Openbravo tuve que añadir un método extra que insertara la fecha porque no se puede modificar un método que pueda utilizar otro test automático ya que, si se cambia la funcionalidad del método el test automático que utiliza ese método podría estropearse.

AmortizationFlow.java

En esta clase se realiza el flujo de amortización. Como esta ventana tampoco estaba creada en la infraestructura de Selenium de Openbravo, se han creado las clases: *Amortization.java, AmortizationHeaderTab.java* y *AmortizationWindow.java*. Básicamente, hay que construir estas clases de la misma forma que las que se crearon para poder utilizar la ventana *G/L Journal*. Una vez que se construyen estas clases ya se puede interactuar con la ventana *Amortization*, por lo que solo queda utilizarla para realizar el flujo.

Lo primero que hay que hacer es *instanciar* el Objeto *Amortization* y abrir la ventana con el método *open*(), después hay que insertar los datos en la cabecera utilizando el método *create*(). Una vez que se guarda la cabecera solo queda completarla con el comando *complete*(). De esta forma, se realiza el flujo de amortización que se realizará solo una vez al final de año. Esta condición se específica en la clase *TFGTestSuit.java*, en la que se indica que esta clase solo se ejecute si el mes en el que se están insertando los datos es diciembre.

FinancialAccountWindow.java

En esta clase se va a realizar la reconciliación de las transacciones. Lo primero que hay que hacer es ir a la ventana *Financial Account* para eso hay que *instanciar* el objeto *FinancialAccount* y utilizar el método *open*(). Una vez que ya estamos en la ventana, hay que seleccionar el registro de la cabecera que tiene como nombre *Bank – Account 1*, esto se realiza con el comando *select*() del objeto *FinancialAccount*, este método necesita un objeto del tipo *Data* con el nombre del banco para poder seleccionar el registro.

Una vez que tenemos seleccionado el banco, hay que ir a la pestaña *transactions* para poder mirar que transacciones están pendientes de reconciliar en el mes que toca. Para esto, hay que filtrar con la fecha deseada en la pestaña *transactions* para que solo aparezcan aquellas transacciones pendientes que tienen que reconciliarse en dicha fecha.

Seguido, hay que coger los valores de cada transacción y crear un nuevo registro en la pestaña *Bank Statement Lines* con los valores de estas transacciones. Para poder recoger los valores de estos registros se utiliza el método *getData*() del objeto *Transaction*, este método tiene como parámetro el nombre del campo del que se quiere recoger los valores. Una vez que se tiene el valor de los campos es muy fácil rellenar las líneas de la otra pestaña utilizando el método *create*(). Esto hay que repetirlo para las fechas deseadas.

Cuando ya tenemos todos los registros en la pestaña *Bank Statement Lines* hay que realizar la reconciliación. Para esto la cabecera de la ventana *Financial Account* tiene un botón que se llama *Match Statement* que realiza la reconciliación utilizando algoritmos de comparación entre los campos de las transacciones y las líneas del banco.

Si los campos son similares realiza la reconciliación de forma correcta. Para poderlo llevar a cabo, hace falta crear un objeto *MatchStatementProcess* y utilizar los métodos *openMatchStatement*() y *pressOkButton*(). Así la reconciliación se efectuaría.

PeriodsFlow.java

Esta clase fue un poco distinta al resto ya que no había un framework de Selenium para poder utilizarla, mejor dicho, no se había actualizado el framework de Selenium para esta clase. Hace años, cuando empezaron a desarrollar los test de automáticos, se creaban ventanas de una forma muy distinta a como lo hacen ahora. Muchas de esas ventanas, como las han utilizado en la actualidad, no han sido actualizadas y por lo tanto no tienen la misma funcionalidad que tienen otras. La

ventana *Fiscal Calendar* es una de estas ventanas y solo tenía la funcionalidad justa como para ejecutar un test automático.

Para realizar este flujo se añadió la funcionalidad extra que se necesitaba. Simplemente, hubo que añadir métodos que no existían para poder seleccionar una pestaña, para crear una ventana, para filtrar un registro y para seleccionar un botón dentro de una ventana. Después de este breve comentario explicaré en que consiste esta clase.

Esta clase se debe de ejecutar solamente a final de año, por lo que en la clase *TFGTestSuit.java* se especifica que solo puede añadirse en diciembre.

Su función es ir a la ventana *Fiscal Calendar* y mirar si están los periodos de los siguientes dos años creados y abiertos. En caso de que lo estén no pasa nada, simplemente lo verifica, pero en el caso de que no estén los tiene que crear.

Para esto, lo primero que hay que hacer es abrir la ventana *Fiscal Calendar* y seleccionar el calendario de España o de US. Como se utiliza código antiguo se utiliza una sentencia distinta, pero parecida a la de otros flujos. Para realizar lo que acabo de comentar hay que utilizar el método *select*() del objeto *CalendarTab*. El método *select*(), a diferencia del resto de flujos, crea la ventana *FiscalCalendar* y selecciona el registro indicado. Esto es posible porque, en vez de tener que pasarle un único parámetro haciendo referencia al registro que se quiere seleccionar, existen dos parámetros, uno que indica el contexto de la página, que sirve para poder crear la ventana, y otro parámetro para seleccionar el registro deseado.

Una vez que ya tenemos el calendario seleccionado, el siguiente paso es comprobar si los dos siguientes años están creados o no. Para poder hacer esto se utilizan los métodos que añadí. Para eso se va a utilizar el método *select*() de del objeto *YearTab,* pasándole como parámetros el contexto y un objeto tipo *Data* con los años que se quieren mirar. Al tratar de seleccionar el registro pueden pasar dos cosas, la primera es que se pueda seleccionar el registro, en ese caso no habrá que hacer nada, la segunda es que no se pueda seleccionar porque el año no exista, en ese caso "saltará" una excepción que es lo que se utiliza para poder detectar que el año no está y así se pueda crear. Al utilizase código antiguo de Openbravo no había una

mejor forma para poder ver si existía el registro, por lo que se utiliza un *try* y un *catch.*

Dentro del *try* se intentará seleccionar el registro con el año deseado y dentro del *catch* los pasos a realizar en el caso de que no lo encuentre. Si no lo encuentra primero se tendrá que eliminar el filtro de búsqueda que introduce el método *select*() porque no se podría pulsar el botón *create periods* ya que no se refrescaría la nueva pestaña con el botón. Lo segundo que hay que hacer es crear el nuevo registro con el método *create*() y después utilizar el método *createPeriods*() y el *verifyCount*() para crear los periodos en la pestaña *periods* y verificar que se han creado los doce meses del año.

Esto se tiene que hacer para los dos bancos y para los dos siguientes años, por lo que se tiene que repetir este proceso cuatro veces.

PhysicalInventoryFlow.java

Esta es la última clase del flujo funcional, una vez que se han creado el resto de flujos ahora hay que mantener unos productos elegidos por la empresa a un nivel estable. Para hacer esto hay que aumentar el número de productos en el inventario en la ventana *Physical Inventory.* En esta clase hay que crear primero un objeto del tipo *PhysicalInventoryHeader* y utilizar el método *open*() para abrir la ventana de *Physical Inventory.* Después, hay que ingresar la información dentro de los campos, utilizando el método *create(),* como en otras ocasiones. Cuando ya se ha introducido la información hay que asegurarse de que se ha guardado bien y validar que los campos son los esperados con los métodos *assertSaved*() y *assertData()*. Lo mismo habría que hacer para la pestaña *Lines,* que es donde se especifica la cantidad de producto que se va a ingresar. Para terminar solo queda hacer clic en el botón *process inventory count* para que se procese la acción, esto se puede hacer utilizando el método *processInventoryCount*() de la cabecera.

TFGTestSuit.java

Esta clase es una clase especial en la que se van a ejecutar todos test automáticos que he creado en la secuencia adecuada, repitiéndose tres veces y obteniendo los tres meses del trimestre como parámetros y pasándoselos a las clases en cada ejecución

para modificar las fechas esenciales de los documentos. De esta forma, en cada ejecución de esta clase se realizarán los flujos tres veces con un mes diferente en cada ejecución.

Para poder utilizar esta clase hizo falta hablar con el equipo de QA, ya que las suites de los test que tenían en la infraestructura de Selenium no estaban preparadas para poder realizar varias ejecuciones parametrizadas. Como esto supuso una funcionalidad nueva que Openbravo no tenía antes, y les resultaba interesante poder parametrizar secuencialmente una suite, investigué, como poder realizarlo. Finalmente, gracias a una librería que encontré, pudimos adaptar la *suite* para ejecutar los test secuencialmente. De esta forma, las clases podían saber en qué iteración se encontraban y así cambiar la fecha. Para poder trabajar mejor con las fechas cree nuevos métodos dentro de la clase *OBDate* que tiene Openbravo para manejar fechas. Tuve que crear métodos para calcular cuál es el primer mes del trimestre, el segundo y el tercero para poder modificar las fechas en el resto de clases. Es por eso que todos los métodos que cree dentro de *OBDate* reciben como parámetro un entero que hace referencia al mes en el que toca ejecutarse el test para poder crear las fechas deseadas y así poder modificar los distintos registros de cada flujo.

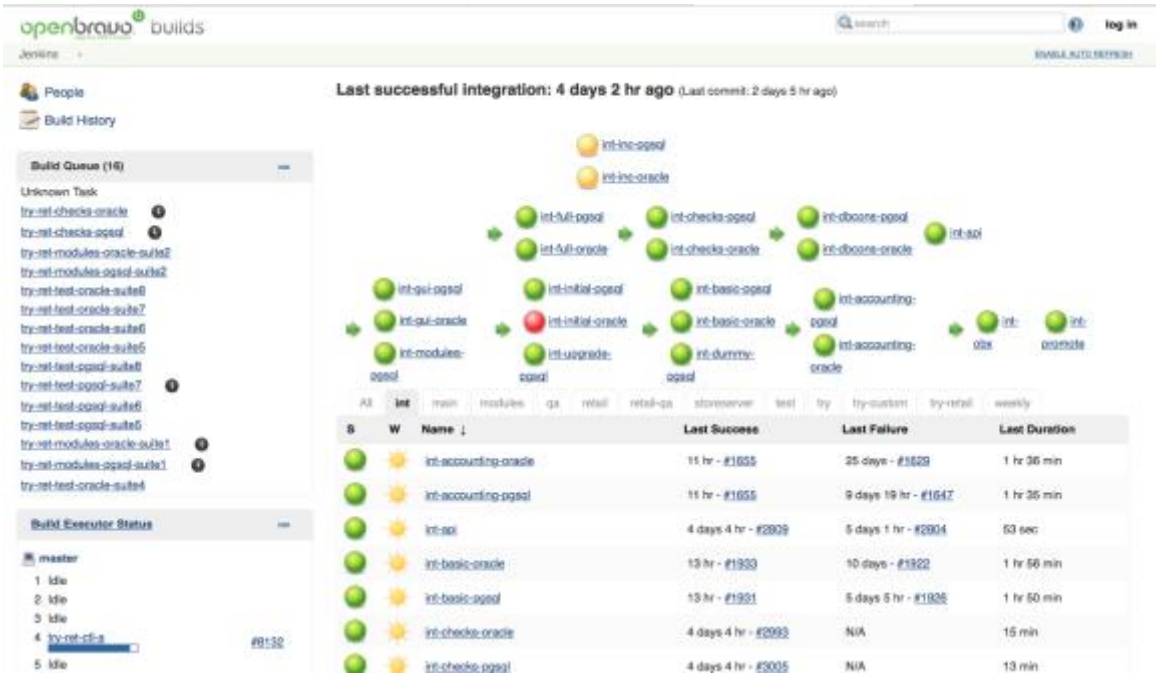
(Imagen 4.13)

Para poder entender un poco mejor cómo es esta clase en la imagen 4.13 se puede ver que están todos los flujos para reproducir dentro de la anotación *@SuiteClasses*. Además, también se llaman a los métodos OBDate.firstMonth(), OBDate.secondMonth() y OBDate.thirdMonth(), que son los parámetros con los que se ejecutan los test en cada iteración. En la primera iteración se calculará cuál es el primer mes del trimestre y se pasará un entero a cada test haciendo referencia al mes que toca ejecutarse. Para la segunda y tercera ejecución será lo mismo, solo que pasando como parámetro los dos meses siguientes. De esta forma se consigue que el flujo se ejecute una vez por mes y que se generen los documentos con distintas fechas.

5. Validación del producto

Una vez que el test automático cumple su función y consigue insertar y actualizar los datos dentro del ERP de Openbravo para cada trimestre, hay que introducir el proyecto en el proceso de integración continua de Openbravo para que se le someta a una serie de test que tiene la empresa para validar y verificar que la calidad del producto final es óptima.

Openbravo utiliza la herramienta Jenkins para realizar estas tareas.



(Imagen 5.1)

Como se puede ver en la imagen, esta es la página en la que se puede ver el estado de las tareas que hay en Jenkins para validar el proyecto, cada una de las bolas verdes realiza un proceso de testeo sobre el proyecto para validar y verificar una función específica. Si el proyecto no funcionase bien Jenkins avisaría mediante un Email especificando en cuál de estas tareas se ha producido el error. Una vez que el proyecto se pasó por los test y se vio que era estable se introdujo en una tarea específica dentro de Jenkins. Más concretamente, se introdujo en una tarea que se ejecuta al final del proceso de integración continua y que además solo se realiza en

cada nueva actualización de la aplicación, esta tarea es la última que se ve en la imagen(int-promote). De esta forma, todos los flujos que he desarrollado se ejecutarán cuando esta tarea se inicie para crear, actualizar y validar los datos para la aplicación. Además, de esta forma también se validan los flujos que he desarrollado y se demuestra que funcionan.

6. Conclusión y Lineas futuras

Al final, el resultado de este proyecto es la creación de una herramienta automática para poder insertar, actualizar y validar los datos dentro de la aplicación de Openbravo. Una vez que el test se ejecuta, consigue crear una serie de flujos muy concretos que necesitaba la empresa para poder ahorrar tiempo de producción, para tener un nuevo método con los que validar estos flujos, para añadir funcionalidad a la infraestructura de selenium de Openbravo…

Este trabajo no es solo un proyecto de aprendizaje o un análisis del comportamiento de una materia, sino que se trata de la creación de una herramienta útil que una empresa va a utilizar en su producto más importante.

Sin embargo, aunque el desarrollo de este trabajo cumple con las necesidades y expectativas que tenía Openbravo, aún podría evolucionar hacia una herramienta con mayor funcionalidad. Esta sería una interesante línea de investigación y mejora para el futuro, en el caso de que la empresa decida seguir invirtiendo en su desarrollo

Este proyecto se diseñó de manera ordenada para que si, en cualquier momento, un trabajador de Openbravo o yo mismo tuviera que realizar un *refactor* de la aplicación web fuera fácil de realizar. Además, está diseñado para que se le pueda insertar tanto nuevas funcionalidades como nuevos flujos o datos.

No solo esto, sino que como los datos que se están insertando dentro de la aplicación son unos valores fijos que Openbravo eligió y quiso que fueran siempre los mismos, se podrían cambiar por variables que recogieran valores de una base de datos y así poder cambiar los datos cuando se quisiera y así flexibilizar mucho mas la solución.

De esta forma se ha desarrollado una solución que cubre todas las especificaciones y necesidades que la empresa necesitaba, pero, a la vez, dejando la

puerta abierta a un posible desarrollo futuro en el cual se pudiera añadir más características.

7.Referencias

[1] Documento AECA · Nuevas Tecnologías y Contabilidad · N.o 6

[2] Openbravo Issues https://issues.openbravo.com/my_view_page.php (8-6-2017)

[3] Openbravo Wiki http://wiki.openbravo.com/wiki/How_to_setup_Eclipse_IDE (8-6-2017)

[4] Openbravo Wiki http://wiki.openbravo.com/wiki/ERP_2.50:Developers_Guide/Concepts/Development_Build_Tasks (8-6-2017)

[5] Postgresql documentacion https://www.postgresql.org/files/documentation/pdf/9.5/postgresql-9.5-A4.pdf (8-6-2017)

[6] Mercurial. https://www.adictosaltrabajo.com/tutoriales/mercurial/#1 (8-6-2017)

[7] http://usuariodebian.blogspot.com.es/2012/01/openjdk-version-libre-de-java.html (8-6-2017)

[8] Wikipedia. Tomcat. https://es.wikipedia.org/wiki/Tomcat (8-6-2017)

[9] Wikipedia. Selenium. https://es.wikipedia.org/wiki/Selenium (8-6-2017)