Proyecto Fin de Carrera
Ingeniería de Telecomunicación

Diseño e Implementación de un módulo en Spring-Batch dentro de un proyecto J2EE

Autor: Anouar EL Qadim

Tutor: Sierra Collado, Antonio Jesus

Departamento de Ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2018

Proyecto Fin de Carrera
Ingeniería de Telecomunicación

Diseño e Implementación de un módulo en Spring-Batch dentro de un proyecto J2EE

Autor:
Anouar, EL Qadim

Tutor:
Sierra Collado, Antonio Jesus
Profesor titular

Departamento de ingeniería Telemática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2018

Proyecto Fin de Carrera: Diseño e Implementación de un módulo en Spring-Batch dentro de un proyecto J2EE

Autor: Anouar EL Qadim

Tutor: Sierra Collado Antonio Jesus

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2017

El Secretario del Tribunal

A mi familia

A mis maestros

Agradecimientos

En primer lugar, quiero agradecer a mi Madre, que me ha ayudado con todo lo que ha podido y más, para que a día de hoy pueda presentar este trabajo.

En segundo lugar, agradecer a mi familia, mi pareja y amigos de siempre, todo el apoyo que también me han aportado, así como todas las experiencias vividas que hemos tenido juntos y que me han llevado a ser como soy y a poder estar donde estoy. Finalmente, quiero agradecer a todos los profesores que he tenido en esta universidad, aunque a algunos los haya conocido de manera fugaz, todos los conocimientos y valores que me han transmitido en mi paso por la universidad, ya que espero y estoy convencido de que los mismos me serán de gran ayuda en mi futuro profesional y personal.

Anouar EL Qadim

Alumno de la Escuela Técnica superior de Ingenieria

Sevilla, 2018

Resumen

Este trabajo fin de carrera destaca la explicación de manera detallada las tecnologías utilizadas, los procesos y las fases de la implementación de un módulo dentro de un sistema informático J2EE en una de las empresas multinacional líder del sector del desarrollo de software.

El presente proyecto fin de carrera trata los diferentes pasos dados para la realización de este módulo, comenzando con la evolución de la capa de presentación, luego las correcciones de errores y creando nuevas funciones. De hecho, la aplicación se basa principalmente en una arquitectura Java / J2EE con la contribución de Framework Spring, se detallara la explicación framework Spring-Batch, y las diferente tecnologías utilizadas dentro de la aplicación.

La aplicación trata de armonizar, asegurar y compartir datos sobre la gestión de las campañas arancelarias ARS (Agencias regionales de sanidad). Desarrollado en un formato web y centralizado a nivel nacional, el sistema de información de Armonización e Intercambio de Información (HAPI, “HA” viene de armonización, “P” de intercambio y “I” de informacion), ha estado en funcionamiento desde 2012. Es una aplicación de "asignación de recursos médico-sociales" diseñada para armonizar prácticas de fijación de precios entre regiones y para asegurarlas.

A continuación vamos a dar un breve resumen del sistema HAPI, este último tiene dos objetivos principales:

- A nivel regional, fijar el precio de los establecimientos y servicios hasta la publicación de las notificaciones arancelarias y seguir el consumo de los sobres regionales en tiempo real. La aplicación hace confiables las bases de datos y armoniza el proceso de fijación de los precios independientemente de la organización de las agencias. Garantiza la armonización de los métodos de cálculo de tarifas basados en el Código de Acción Social y las familias.
- A nivel nacional, tener visibilidad sobre el consumo de los fondos delegados a las agencias de sanidad para financiar el funcionamiento de los establecimientos y servicios "en tiempo real". En cada etapa del precio de una renovación, actualización, nuevas medidas, impacto del resultado, se "ajusta" para tener una parte de los importes tasados

La aplicación o el sistema de información web están divididos en tres capas, más una capa de consulta de datos:

- Presentación de la vista frontal
- Servicios comerciales
- Acceso a datos
- Consulta de datos (QlickView)

Abstract

This thesis highlights the detailed explanation of the technologies used, the processes and the phases of the implementation of a module within a J2EE computer system in one of the leading multinational companies in the software development sector.

The present final project deals with the different steps taken for the realization of this module, beginning with the evolution of the presentation layer, then the corrections of errors and creating new functions. In fact, the application is mainly based on Java / J2EE architecture with the contribution of Framework Spring, the Spring-Batch framework explanation, and the different technologies used within the application.

The application tries to harmonize, secure and share data on the management of the ARS tariff campaigns. Developed in a web format and centralized at national level, the information system of Harmonization and Exchange of Information (HAPI) has been in operation since 2012. It is an application of "allocation of medical-social resources" designed to harmonize practices of setting prices between regions and to insure them.

Next we will give a brief summary of the HAPI system, the latter has two main objectives:

- At the regional level, set the price of establishments and services until the publication of tariff notifications and follow the consumption of regional envelopes in real time. The application makes databases reliable and harmonizes the process of setting prices regardless of the organization of the agencies. It guarantees the harmonization of the methods of calculating rates based on the Social Action Code and families.
- At the national level, have visibility over the consumption of the funds delegated to the agencies to finance the operation of establishments and services "in real time". At each stage of the price of a renewal, update, new measures, impact of the result, it is "adjusted" to have a part of the assessed amounts

The application or web information system is divided into three layers, plus a data query layer:

- Presentation of the front view
- Commercial services
- Data access
- Data query (QlickView)

Índice

Agradecimientos **9**

Resumen **11**

Abstract **13**

Índice **14**

Índice de Tablas **16**

Índice de Figuras **17**

1 Contexto del proyecto **20**

2 Metodología, herramientas y tecnologías **23**

2.1. Scrum *23*

2.1.1. ¿Por qué Scrum? 23

2.1.2. Gestor de tareas 24

2.1.3. Ilustración de Asociación de un Sprint 26

2.1.4. Modificación y planificación del perímetro 27

2.2. Herramientas *27*

2.2.1. Herramienta de desarrollo 28

2.2.2. Sistema de control de versiones 28

2.2.3. Sistema de control e integración continua 29

2.2.4. Comunicación con equipos remotos 32

2.2.5. Herramienta de supervisión y calificación 33

2.2.6. Mantis Beug Tranker 35

2.3. Tecnologías *35*

2.3.1. Spring Framework 35

2.3.2. Spring Batch 36

2.3.3. Linux Redhat 6 38

2.3.4. Windows Server 2012 38

2.3.5. Tomcat 38

2.3.6. Qlikview 11 39

2.3.7. Hibernate 39

2.3.8. Atomikos 40

2.3.9. Log4j 40

2.3.10. IText 40

2.3.11. Jasper Informa 40

2.3.12. JUnit 41

2.3.13. Apache Tiles 41

2.3.14. Ajax 42
2.3.15. Tecnologías WEB 42

3 Arquitectura General De la Aplicación 39
3.1. Capa Web 40
3.3.1. Ejemplo de configuración de capa 41
3.2. Capa de negocio 44
3.2.1. Definición de servicios 44
3.2.2. Declaración de contexto spring 44
3.3. Capa de Persistancia 46
3.3.1. Definición de DAO 46
3.3.2. Declaracion de context spring 46
3.4. Capa de modelo 48
3.4.1. Mapeo Objeto-relacional 48
3.4.2. BusinessEntity 48
3.5. Capa de Batch 50

4 Tareas Realizadas 52
4.1. Evolución de la capa Batch 52
4.1.1. Tasklet 53
4.1.2. Planificación y requisites: 53
4.1.3. Plan de calidad 56
4.2. Etapas de implementación 58
4.2.1. Creación de proceso de lanzamiento 59
4.2.2. Recibir y enviar fichero vía ftp 60
4.2.3. Lectura de un fichero Excel 62
4.2.4. Etapas de gestión de nuestro modulo 64
4.2.5. Gestión des mensajes y excepción 65
4.2.6. Envió de Email 66
4.2.7. Generación JAR 67
4.2.8. Creación de cron 69
4.3. Pruebas unitarias y de integración 69

5 Conclusión 71

Referencias 74

Glosario 77

ÍNDICE DE TABLAS

Tabla 2–1. Ilustración de un sprint en JIRA 26

ÍNDICE DE FIGURAS

Figura 1.1. Arquitectura de la aplicación, descripción general. 21
Figura 2.1. Muestra el ciclo de vida del proyecto de **Scrum**. 24
Figura 2.2. Ejemplo de JIRA – Seguimiento de tarea. 25
Figura 2.3. Ejemplo de JIRA – BurnDown de JIRA de tarea. 25
Figura 2.4. Diferentes plataformas. 27
Figura 2.5. Repositorio común a los desarrolladores. 29
Figura 2.6. Ejemplo de repositorio para nuestra Aplicación. 29
Figura 2.7. Ejemplo de la visualización Jenkins. 30
Figura 2.8. Imagen XL-deploy. 31
Figura 2.9. Vista de SonarQube. 31
Figura 2.10. Herramienta de MoabaXterm. 33
Figura 2.11. Herramienta de supervisión XYMON. 34
Figura 2.12. Herramienta de calificación HP-Quality Center. 34
Figura 2.13. Arquitectura general de la aplicación [16]. 35
Figura 2.14. Arquitectura general Spring Framework. 36
Figura 2.15. Componentes de Spring-Batch. 37
Figura 2.16. Base de datos Microsoft Windows Server 2012. 38
Figura 2.17. Plataforma de inteligencia QlickView. 39
Figura 2.18. Mostración de Template final de la aplicación. 41
Figura 3.1. Estructura de la aplicación en Eclipse. 39
Figura 3.2. Servidores usados en las aplicación. 40
Figura 3.3. Fichero de propiedad en la estructura de módulos de la aplicación. 48
Figura 4.1. Configuración en Eclipse lanzamiento de Batch. 52
Figura 4.2. Esquema de Job de spring batch. 52
Figura 4.3. Diagrama EDT. 54
Figura 4.4. Diagrama Gantt. 55
Figura 4.5. Configuración guardar-formatear 57
Figura 4.5. Configuración de step en spring Batch. 59
Figura 4.6. Repertorio en la comunicación FTP. 60

Figura 4.7. Ejemplo de Excel de entrada al módulo Batch. 63
Figura 4.8. Ejemplo de Modelo conceptual y relacional. 65
Figura 4.9. Vista Jenkins de configuración de tareas 68

1 CONTEXTO DEL PROYECTO

Los sistemas interoperan, las personas se relacionan, o estaría bien que lo hicieran, ¿verdad?".

- Paula Keen-

Este proyecto ha sido desarrollado en el departamento de software de **SopraSteria Spain**.

La aplicación donde desarollaremos el módulo trata de armonizar, asegurar y compartir datos sobre la gestión de las campañas arancelarias de los ARS (agencias regionales de sanidad). Desarrollado en un formato web y centralizado a nivel nacional, el sistema de información de Armonización e Intercambio de Información (HAPI) ha estado en funcionamiento desde 2012. Es una aplicación de "asignación de recursos médico-sociales" diseñada para armonizar prácticas de fijación de los precios entre regiones y para asegurarlas.

HAPI tiene dos objetivos principales:

- A nivel regional, fijar el precio de los establecimientos y servicios hasta la publicación de las notificaciones arancelarias y seguir el consumo de los sobres regionales en tiempo real. La aplicación hace confiables las bases de datos y armoniza el proceso de fijación de los precios independientemente de la organización de las agencias. Garantiza la armonización de los métodos de cálculo de tarifas basados en el Código de Acción Social y las familias.
- A nivel nacional, tener visibilidad sobre el consumo de los fondos delegados a las agencias para financiar el funcionamiento de los establecimientos y servicios "en tiempo real". En cada etapa del precio de una renovación, actualización, nuevas medidas, impacto del resultado, se "ajusta" para tener una parte de los importes tasados.

La aplicación o el sistema de información web están divididos en tres capas, más una capa de consulta de datos que más adelante explicaré en detalle cada parte:

- Presentación de la vista frontal.
- Servicios comerciales.
- Acceso a datos.
- Consulta de datos (QlickView).

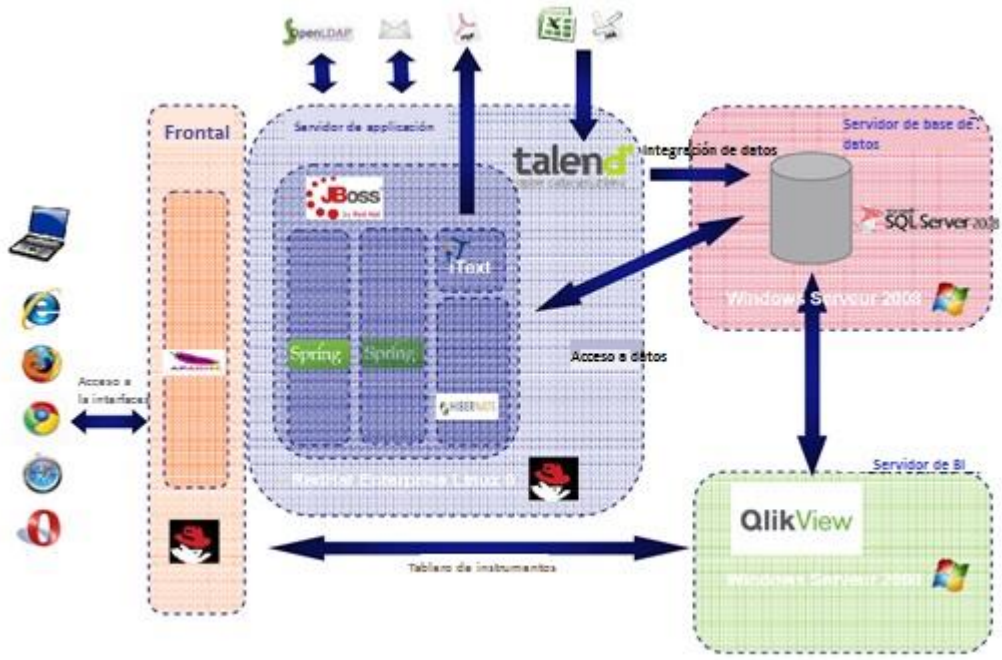


Figura 1.1. Arquitectura de la aplicación, descripción general.

- **Presentación**

La base de datos no contiene información redundante. Todos los datos relacionados con una campaña o un año de trabajo, se relacionan directamente o indirectamente con un tipo de fondo. La base de datos se encarga de llevar la información del año de la campaña.

El modelo conceptual de datos (MCD) tiene como objetivo formal escribir datos, que será utilizado por el sistema de información. Por lo tanto, es una representación de los datos, fácilmente comprensible, para describir el sistema de información usando entidades. La entidad es la representación de un elemento material o inmaterial que tiene un papel en el sistema que queremos describir. Una clase de entidad es un conjunto de entidades del mismo tipo, es decir, cuya definición es la misma.

Los modelos conceptuales de nuestra aplicación apuntan a resaltar los vínculos entre las diferentes entidades. En cuanto a los datos de cada entidad, el detalle de cada entidad se actualiza en el MCD. El mapeo de cada entidad especifica todos los datos de cada entidad.

Para la alimentación de la mayoría de los Datos en nuestra aplicación se va utilizar los procesos BATCH (o procesos por lotes) .Son aquellos programas que se lanzan generalmente de manera programada y que no requieren ningún tipo de intervención humana. Suelen ser procesos relativamente pesados, que tratan una gran cantidad de información, por lo que normalmente se ejecutan en horarios con baja carga de trabajo para no influir en el entorno transaccional.

Algunos ejemplos de este tipo de procesos podrían ser los destinados a la generación o tratamiento de ficheros de facturación o la generación masiva de documentos (ej.: cartas de bienvenida a nuevos clientes).

El papel de la parte de generación del módulo BATCH dentro de la aplicación será lo siguiente:

- Agregar o modificar datos del repositorio en la base de datos
- Eliminar y agregar datos comerciales a la base de datos
- Consiste en buscar archivos en un entorno de transferir archivos por ejemplo Protocolo de Transferencia de Archivos FTP, (formato Excel, csv, xml, zip) cuyo nombre responde a reglas específicas, cargarlos en la base de datos, enviar en el FTP los intercambios de rechazos relacionados con la carga, para enviar los archivos FTP con los datos ingresados en la aplicación (archivos XML).
- Mejora de la aplicación del lote para asegurar su funcionamiento

Mi aportación dentro del módulo ha sido el desarrollo de sistemas de diversa complejidad, efectuando análisis

de requerimientos, diseños físicos, diseños lógicos y ejecutando tareas de programación. También diseñar pruebas de las aplicaciones y ejecutar pruebas de las aplicaciones después del desarrollo de las mismas, documentar o revisar la documentación de las aplicaciones según los estándares establecidos.

En resumen y en relación con el modulo a diseñar mi trabajo era la recolecta de información para los programadores de las aplicaciones, la estimación el tiempo necesario para su desarrollo, la orientación a los codificadores, la supervisión y coordinación del plan de trabajo de los mismos, y luego evaluar el software desarrollado por ellos antes de elaborar informes periódicos de las actividades realizadas antes de entregar todo el proyecto.

2 METODOLOGÍA, HERRAMIENTAS Y TECNOLOGÍAS

Como parte de este proyecto, hemos estado interesados en adoptar un enfoque Ágil. "Un método ágil es un enfoque iterativo e incremental, que se lleva a cabo en un espíritu colaborativo, con el formalismo justo. Genera un producto de alta calidad teniendo en cuenta las necesidades cambiantes de los clientes".

Estas metodologías son esencialmente dedicadas a la gestión de proyectos de TI (Tecnologias de la Informaci), confían en ciclos de desarrollo iterativos y se adapatan basándose en las necesidades cambiantes del cliente.

En particular, permiten involucrar a todos los empleados así como a cliente en el desarrollo del proyecto. Estos métodos generalmente ayudan a cumplir mejor las expectativas del cliente en un corto período de tiempo limitado (en parte debido a su participación) mientras se desarollan las habilidades de los empleados.

Algunos ejemplos de métodos de desarrollo Agile son:

- **Scrum.**
- Programación extrema Desarrollo de software adaptable (ASD).
- Método de desarrollo de sistema dinámico (DSDM).

2.1. Scrum

El método SCRUM [1], define un marco para la realización de proyectos complejos. Inicialmente planeado para el desarrollo de proyectos de tipo "Software", este método se puede aplicar a cualquier tipo de proyecto, desde el más simple al más innovador, y de una manera muy simple.

Esta metodología permite adaptarse rápidamente a cambios de un cliente a frecuencia regular. Cuando una iteración finaliza, el equipo y el cliente vuelven a evaluar las especificaciones del software.

2.1.1. ¿Por qué Scrum?

Antes, el cliente no veía la evolución del trabajo y apenas participaba en la ejecución de su proyecto. De hecho, no pudo comenzar a hacer pruebas antes de que todo estuviera casi terminado. Por otro lado, en Scrum, el cliente está más involucrado en el proceso porque un proyecto contiene varios entregables que debe aprobar. Con cada entregable, las funcionalidades están en constante mejora y el cliente ve regularmente la evolución del trabajo y si algo no le conviene, es más fácil ajustarlo para satisfacer sus necesidades. Así que Scrum se trata esencialmente de optimizar la previsibilidad de un proyecto y controlar mejor los riesgos.

Scrum se basa en la división de proyectos en iteraciones que todavía se llaman "***Sprints***". Un Sprint es un período durante el cual se realiza una funcionalidad (o incremento del proyecto).
El método **Scrum** define tres roles para un proyecto:

- *Product Owner:* Este es el representante oficial del cliente dentro de un proyecto de **Scrum**. Define las necesidades del producto y escribe especificaciones. También es responsable de definir y priorizar historias de usuarios para cada sprint.
- ***Scrum** Master:* Es una persona responsable de garantizar la aplicación del método y el respeto de sus objetivos. Este no es un gerente de proyecto, sino una persona responsable de eliminar cualquier obstáculo que impida el avance del equipo y del proyecto durante los diferentes sprints.

- *Team Members:* Estas son las personas responsables del sprint y un producto que se puede usar en el sprint. Pueden ser desarrolladores, arquitectos, personas a cargo de hacer pruebas funcionales...

Puede tener una duración que generalmente varía entre dos semanas y un mes. Esta corta duración hace posible entregar rápidamente al propietario del producto un incremento de software con el mayor valor de negocio. El software entregado es valioso para el propietario del producto, ya que las características desarrolladas son las que tienen la más alta prioridad, las que se encuentran en la parte superior del folleto del producto. Otra ventaja de la corta duración de las iteraciones es permitir al propietario del producto modificar las prioridades de las funcionalidades solicitadas a medida que avanza el proyecto de desarrollo. De hecho, si desea agregar características, estos se tendrán potencialmente en cuenta durante la próxima iteración. Al final de cada sprint, los miembros del equipo obtienen un producto parcial potencialmente entregable y los comentarios recopilados ajustan la acumulación de pedidos para el próximo sprint. En sí mismo, **Scrum** solo sugiere tres hitos: *Sprint Planning, Daily Scrum y Sprint Review*. Durante estos últimos, se producen artefactos de "gestión": la acumulación de productos, el registro de iteraciones

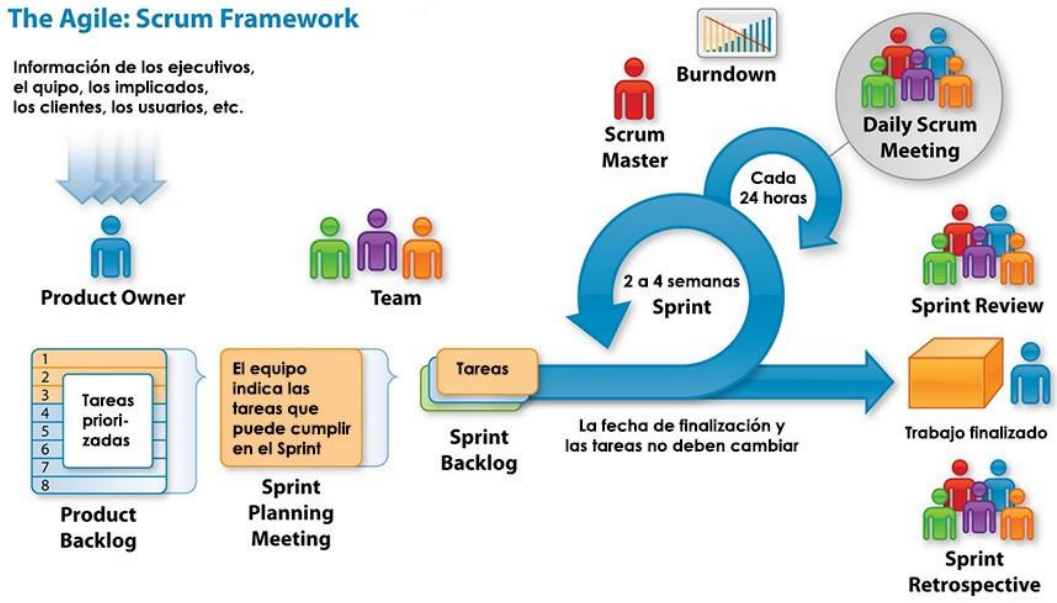


Figura 2.1. Muestra el ciclo de vida del proyecto de **Scrum**.

2.1.2. Gestor de tareas

Toda empresa dispone de un gestor de tareas e incidencias, que favorece la organización de un proyecto favoreciendo los tiempos de desarrollo. Antes de empezar un proyecto es imprescindible un claro planteamiento de las tareas que va a suponer, con el análisis de lo que supone, el tiempo que se ha estimado para invertir en su desarrollo y las relaciones entre ellas si las hubiese además de la asignación de la misma al desarrollador que encargará de realizarla. Una vez comenzado el proyecto, el responsable de la realización de la tarea se encargar ir actualizando su estado y las horas invertidas en la misma. Todo esto facilitará la gestión del proyecto.

JIRA [2] se deriva del nombre japonés para Godzilla. Es la aplicación escogida para realizar el seguimiento por ser uno de los softwares de gestión de tareas, incidencias y proyectos más potentes del mercado.

JIRA se utiliza dentro de una organización de desarrollo de software, como apoyo para la gestión de requisitos, seguimiento del estatus y más tarde para el seguimiento de errores:

- Asignar trabajo, mejorar la planificación de los proyectos y hacer un seguimiento de la actividad del equipo. Además, te asegurarás de que tu equipo sepa exactamente qué debe hacer y cuándo.
- Capturar y organizar incidencias, buscando siempre la excelencia en el servicio a las peticiones de los clientes.

- Generar y monitorizar los flujos de trabajo o work-flows para que se ajusten a los procesos existentes, pero que a la vez se puedan adaptar a la vez que el equipo evoluciona.
- Permitir la colaboración entre todos los miembros del equipo para que puedan trabajar más eficientemente.
- Mejorar la visibilidad: con JIRA siempre tendrás toda la información actualizada, a través de notificaciones por email, chat o en el móvil. Siempre estarás al tanto de lo que pasa en el proyecto. Podrás hacer seguimiento de las tareas más importantes, monitorizar los flujos de trabajo y compartir información con potentes cuadros de mando.

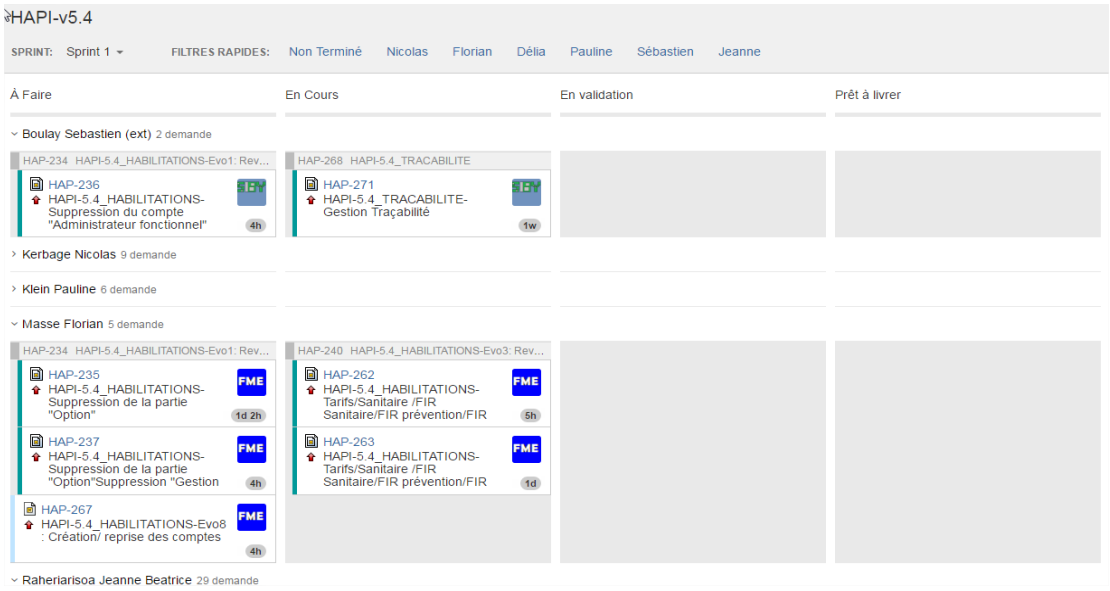


Figura 2.2. Ejemplo de JIRA – Seguimiento de tarea.

La curva que vemos abajo es una ilustración del tiempo recorrido y los objetivos alcanzados para el desarrollo. Esa curva gris representa la curva "Objetivo" que traza la disminución de los tiempos restantes proyectada para aterrizar en una fecha determinada. La curva roja representa la curva "real Tiempo restante" que rastrea la evolución de los tiempos consolidado en la versión. La curva verde representa la curva "Consumación real" que rastrea la evolución del consumo total en la versión.

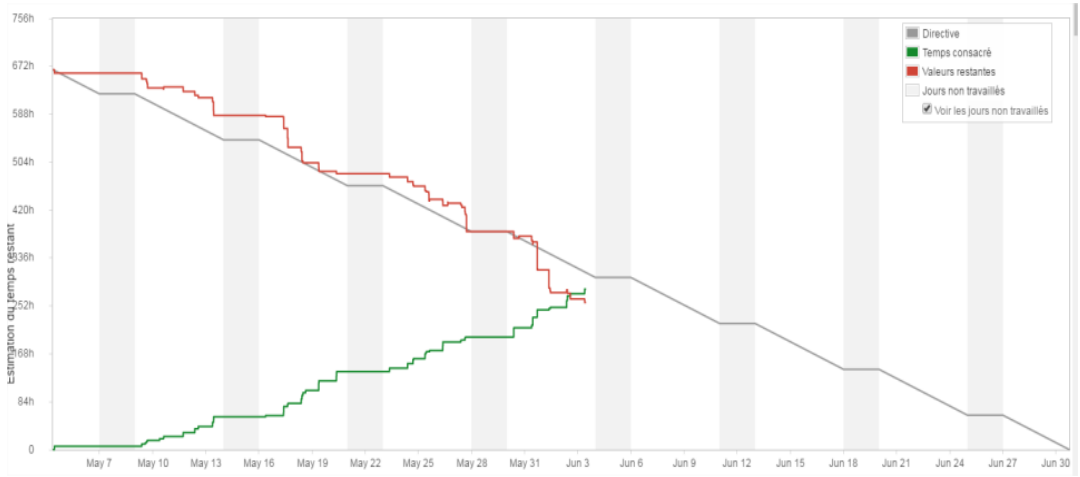
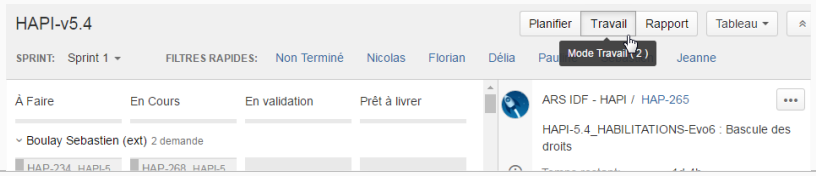
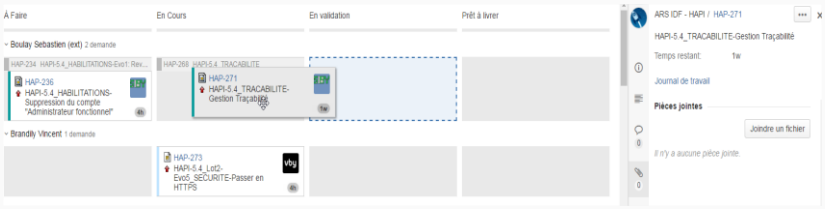
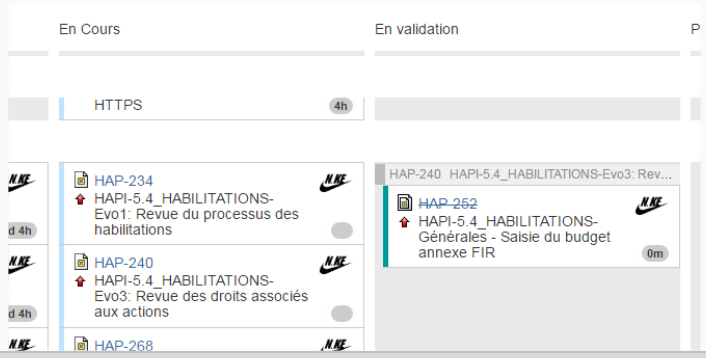
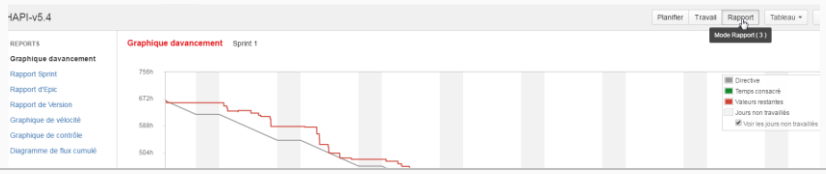
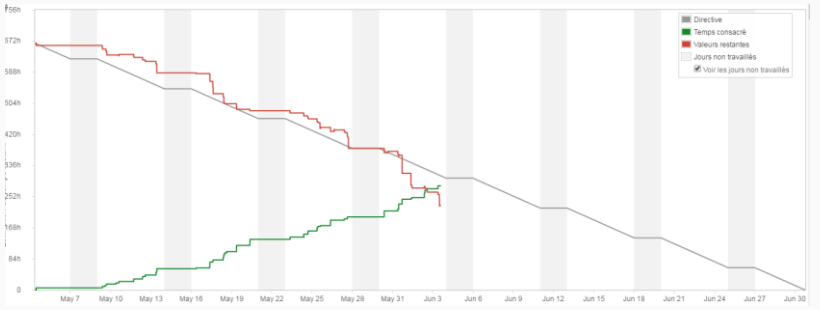


Figura 2.3. Ejemplo de JIRA – BurnDown de JIRA de tarea.

2.1.3. Ilustración de Asociación de un Sprint

Tabla 2–1. Ilustración de un sprint en JIRA

Etapas	Pantallas ilustrativas
Administrar un Sprint	
Ve a "Trabajo".	
Asignar y avanzar tareas después de su avance.	
Actualice tiempos restantes a través del Registro de trabajos de JIRA.	
Sigue el Sprint	
En cada reunión, las tareas de tiempo de "En curso" deben cuestionarse y actualizarse según sea necesario utilizando el campo "Tiempo restante" en el resumen de JIRA.	
Asegúrese de que ninguna tarea de "Validación" sea RAE.	
Reporting du Sprint	

Ve a "Informar".	
Siga la tabla BurnDown para identificar si la versión está siguiendo el objetivo, alejándose o avanzando. Cualquier desviación del objetivo debe desencadenar una acción piloto: dotación de personal, apoyo, planificación, etc.	

2.1.4. Modificación y planificación del perímetro

Un cambio en el alcance conduce a la creación, modificación o eliminación de una o más tareas de JIRA. Los tiempos restantes son revisados.

Es necesario actualizar el contenido del sprint a través del menú "Plan" y revisar el objetivo del sprint según los tiempos restantes actualizadas:

Actualizar la fecha de inicio y finalización de Sprint al considerar el comienzo como la fecha de modificación del perímetro y la fecha de finalización del nuevo objetivo que sigue a los riesgos

2.2. Herramientas

En este Proyecto se ha utilizado un entorno de diferentes plataformas que normalmente se utilizan en un entorno de trabajo conectándolas entre sí para empezar a beneficiarse de las ventajas que el trabajo con Integración Continua aporta.

Se dispondrá de las siguientes herramientas:

- Un gestor de tareas e incidencias.
- Sistema de control de versiones con un repositorio donde se encontrarán todos los proyectos de programación guardados.
- La herramienta de Integración continúa.

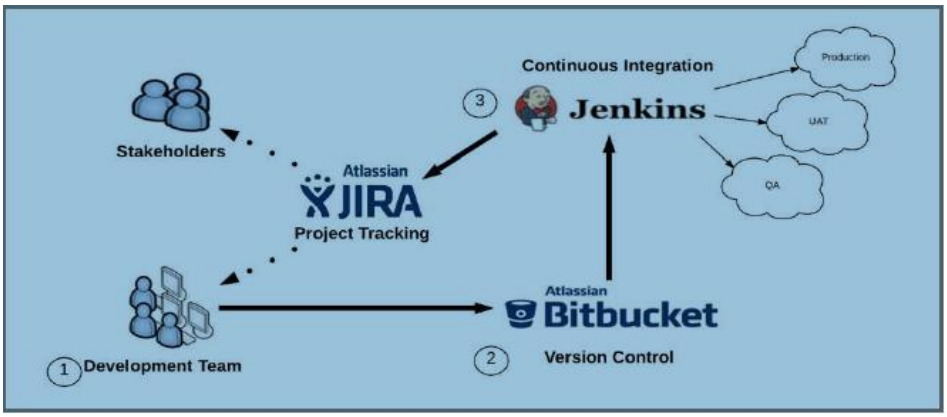


Figura 2.4. Diferentes plataformas.

2.2.1. Herramienta de desarrollo

2.2.1.1. Apache Maven 2.2.1

Maven [3], una palabra en yiddish que significa acumulador de conocimiento, se inició originalmente como un intento de simplificar los procesos de construcción en el proyecto de la turbina de Jakarta. Hubo varios proyectos, cada uno con sus propios archivos de compilación Ant, que eran ligeramente diferentes y los JAR se registraron en CVS. Queríamos una forma estándar de construir los proyectos, una definición clara de en qué consistía el proyecto, una manera fácil de publicar información del proyecto y una forma de compartir JAR en varios proyectos.

El resultado es una herramienta que ahora se puede utilizar para crear y administrar cualquier proyecto basado en Java [4]. Esperamos que hayamos creado algo que facilite el trabajo diario de los desarrolladores de Java y, en general, ayude en la comprensión de cualquier proyecto basado en Java.

El objetivo principal de Maven es permitir que un desarrollador comprenda el estado completo de un esfuerzo de desarrollo en el período de tiempo más corto. Para alcanzar este objetivo, hay varias áreas de preocupación que Maven intenta abordar:

- Haciendo el proceso de construcción fácil.
- Proporcionando un sistema de construcción uniforme.
- Proporcionar información de calidad del proyecto.
- Proporcionar lineamientos para el desarrollo de mejores prácticas.
- Permitir una migración transparente a nuevas funciones.

2.2.1.2. Eclipse y JDK

Eclipse [5] es una plataforma de software compuesto por un conjunto de herramientas de programación de código abierto multiplataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados (del inglés IDE), como el IDE de Java llamado Java Development Toolkit (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse).

Java es un lenguaje de programación a través del que es posible crear cualquier tipo de programa y es también, en la actualidad, uno de los más utilizados dadas sus múltiples ventajas, unos beneficios que lo hacen único. La principal característica -y ventaja- de este lenguaje de programación es que se trata de un lenguaje independiente de la plataforma, es decir, cualquier programa creado a través de Java podrá funcionar correctamente en ordenadores de todo tipo y con sistemas operativos distinto.

El lenguaje de programación utilizado es: Java 1.7 y el JDK utilizado para la compilación y ejecución del código fuente de Java es: Oracle JDK 1.7.0_24

2.2.2. Sistema de control de versiones

Subversión (SVN) [6] es el sistema de control de revisiones utilizado por la empresa para organizar y guardar el código del aplicativo HAPI. Es un software de código abierto basado en repositorios, el funcionamiento del cual se asemeja a un sistema de ficheros. El Eclipse se conecta a un repositorio remoto mediante la dirección de éste. Entendiendo por repositorio un depósito o archivo centralizado donde se almacena y mantiene información digital, en nuestro caso archivos informáticos.

El uso de un repositorio es vital, debido a varias razones:

- Permite el trabajo colaborativo: un proyecto se basa en el trabajo de grupos de desarrolladores que accederán y realizarán cambios sobre los mismos ficheros.

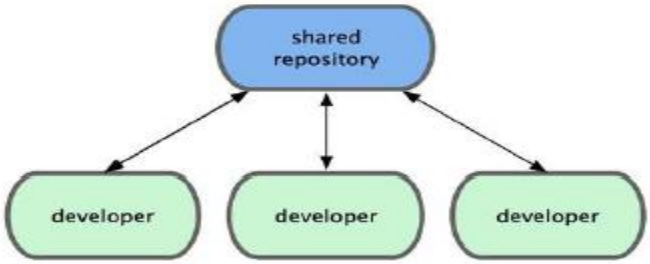


Figura 2.5. Repositorio común a los desarrolladores.

- Permite volver a versiones anteriores de forma sencilla, o comprobar los cambios exactos realizados en un fichero comparando con las anteriores.
- No es necesario SFTP o FTP para subir los archivos al servidor lo que convierte el proceso en mucho más cómodo y rápido.
- Se pueden crear "ramas" que permiten trabajar con una base de código paralela al proyecto en sí, donde se pueden bugs o desarrollar nuevas características para el producto sin afectar a la rama "master" hasta que se tenga una versión estable y se desee
- Dispone de un sistema de etiquetas que permite identificar las distintas versiones del proyecto. En vez de tener distintas copias de respaldo o back-ups de versiones, disponemos de punteros dentro de la misma base de código.

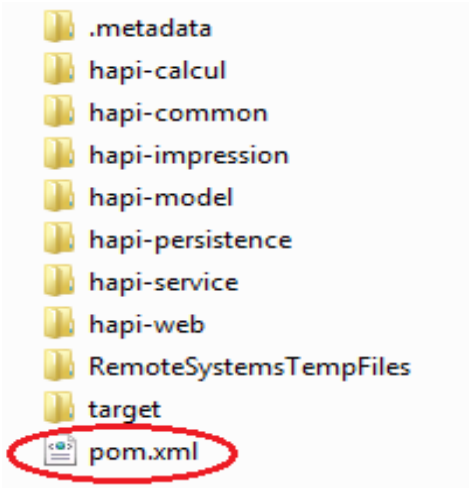


Figura 2.6. Ejemplo de repositorio para nuestra Aplicación.

2.2.3. Sistema de control e integración continua

2.2.3.1. Jenkins

Como herramienta de Integración Continua se ha escogido Jenkins [7]. Los motivos que se ha decidido realizar este proyecto con Jenkins son:

- Gratuita.
- Amplia información en internet sobre su manejo.
- Alto nivel de cantonización.
- Fácil manejo y gran abanico de posibilidades.

Es la herramienta que utiliza la empresa para tener control del código, ya sea para hacer actualización de código, borrar una rama, crear una rama desde el repositorio padre y otras funcionalidades. Cada acción la hace una tarea en concreto (La tarea en Jenkins se llama JOB), de manera que el programador si tiene permisos de ejecución puede hacer la construcción (BUILD) de dicha tarea.

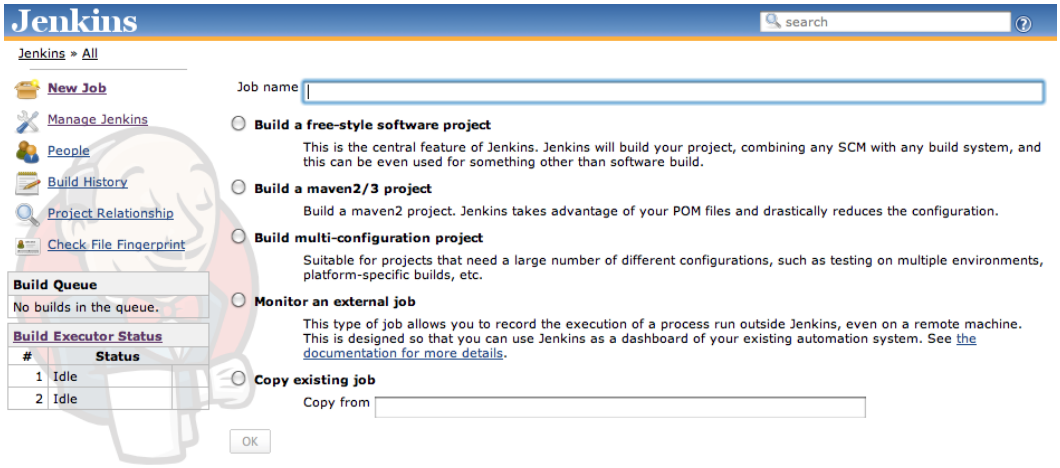


Figura 2.7. Ejemplo de la visualización Jenkins.

Las tareas del Jenkins se organizan en una vista como la que vemos en la anterior imagen. Los grupos de tareas se organizan por pestañas. Por cada Job se muestra el estado de la última ejecución, el estado de las últimas 5 ejecuciones, el tiempo que hace del último acierto, del último fallo y la duración de la última ejecución. Desde esta vista se puede abrir el log o ejecutar el Job. En la parte inferior izquierda de la ilustración se muestran las tareas que están pendientes de ejecución, o en cola, y los nodos activos donde se ejecutan estas. El Jenkins también permite diferentes funciones relacionadas con los nodos, la creación de tareas, edición de vista, etc.

Hay ventanas de configuración de una tarea. Los campos más determinantes aparecen marcados en rojo:

- El nombre de la tarea.
- Los usuarios que pueden interaccionar con el Job, y que tipo de permisos tienen.
- La versión de la JDK que utiliza el Job.
- El nodo en el que se va a ejecutar la tarea, en caso de estar vacío se ejecuta en el nodo master por defecto.
- Los repositorios que se requieren descargar y en que ubicación.
- Las ejecuciones que se van a producir. En el caso del ejemplo es una función programada con Ant con una versión

2.2.3.2. XL-Deploy

Para la mayoría de las organizaciones la entrega de aplicaciones es un proceso largo, estresante y caro. Inevitablemente, los problemas surgen desde el momento del lanzamiento de una aplicación, lo que conlleva nuevos ciclos de entrega.

Cada nueva entrega supone alargar los plazos, incrementar los costes y aumentar los riesgos. Por todo ello, se hace necesario transformar los procesos de entrega en una cadena (pipeline de entrega) que permita responder a las exigencias de competitividad de las empresas. Este proceso debe estar automatizado, ser fiable y continuo.

XL Deploy [8], solución de automatización de entrega de las aplicaciones, se integra con las mejores herramientas de aprovisionamiento y de construcción, lo que permite poner en marcha un verdadero mecanismo de entrega continua.

El objetivo último de la entrega continua es lograr una entrega rápida y con una gran calidad, conectando integración continua, con la construcción, los test y los despliegues automáticos. En MTP consideramos que XL Deploy proporciona la plataforma que permite integrar todos los componentes necesarios para comenzar a trabajar con entrega continua.

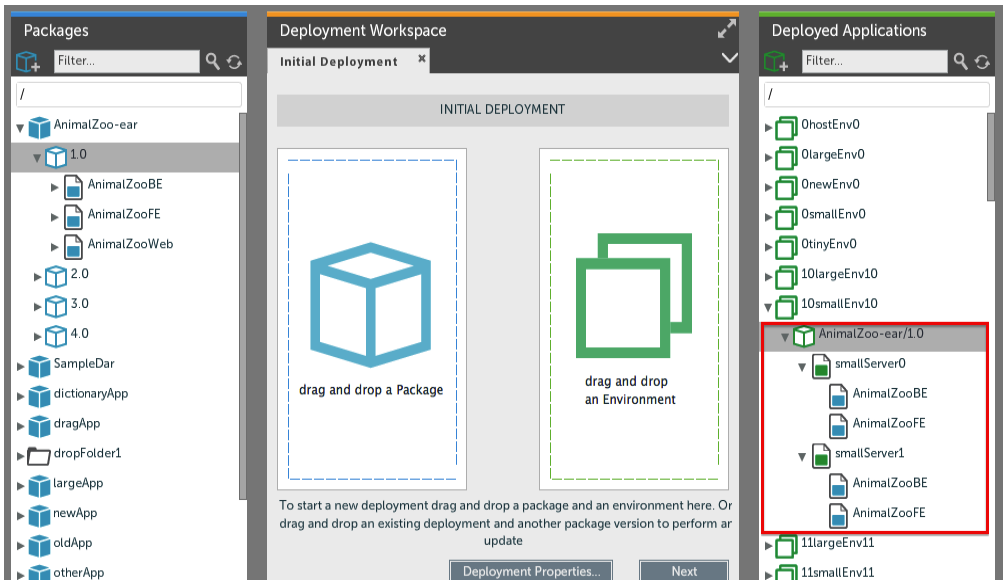


Figura 2.8. Imagen XL-deploy.

Las principales ventajas de XL-deploy son:

- Reducción de los retrasos.
- Validación de las entregas en tiempo real.
- Costes más previsibles.

2.2.3.3. Sonar

SonarQube [9] es una herramienta de análisis estático que evalúa nuestro código fuente. Se trata de software libre que empleo diversos plugins de análisis estático de código fuente como Checkstyle, PMD o FindBugs para obtener métricas que pueden ayudar a mejorar la calidad del código de un desarrollo software y entender que problemas tenemos en nuestro código fuente.

Entre sus funciones están la de dar métricas acerca de código duplicado, pruebas unitarias, cobertura del código, potenciales errores o estándares. Inicialmente, la herramienta estaba pensada para Java pero acepta extensiones para otros lenguajes. En nuestro caso, lo estamos usando para analizar proyectos desarrollados en Symfony. Además, se integra correctamente con herramientas de integración continua como Jenkins, por lo que podemos automatizar este análisis cada vez que un desarrollador sube código.

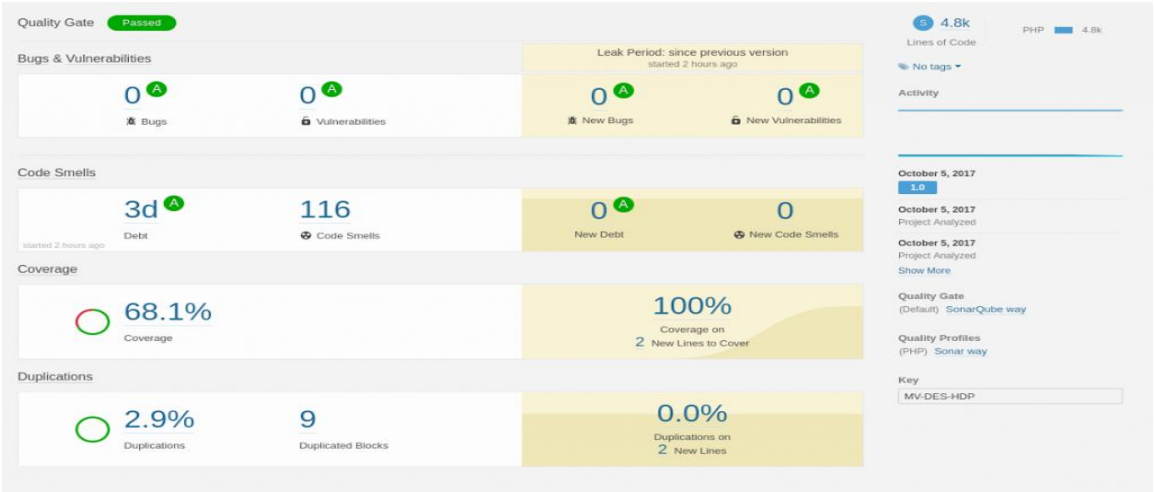


Figura 2.9. Vista de SonarQube.

Un análisis estático del código no es suficiente, necesitamos también otro tipo de herramientas como los test unitarios para verificar el correcto uso y calidad de nuestro software, pero haciendo un análisis del código sin estar en ejecución podemos detectar elementos como:

- Problemas de Diseño
- Duplicidad de Código:
- Detección de Vulnerabilidades
- Standard de codificación
- Monitorización de Cobertura
- Comunicación con equipos remotos

2.2.4. Comunicación con equipos remotos

En este apartado podremos encontrar las diferentes comunicaciones que se han utilizado para recoger los diferentes datos de los entornos de trabajo

2.2.4.1. SSH

SSH (Secure SHell) [10] es el nombre que recibe un protocolo que se utiliza para acceder a maquinas remotas a través de una red. Permite manejar de forma completa la maquina remota mediante un intérprete de comandos, en caso de ser requerido también se puede redirigir el tráfico a un servidor X para visualizar de manera gráfica el terminal.

SSH establece un túnel hacia el servidor remoto. El cliente y el servidor utilizan firmas digitales para verificar su identidad. Lo cual dificulta a un terminal desconocido escuchar el tráfico. Este tipo de comunicación es el que se utilizarán en la aplicación HAPI para la conexión a la parte Batch de los entornos y a las máquinas donde está instalado el Tomcat. Debido a que son máquinas con sistema operativo Linux y permiten este tipo de comunicación.

2.2.4.2. FTP

FTP [11] es un protocolo de transferencia de archivos, su objetivo es transmitir exitosamente archivos entre máquinas en una red sin que el usuario tenga que iniciar sesión en el host remoto. Es por lo tanto un protocolo inseguro.

Existen dos tipos de modos de funcionamiento el activo y el pasivo:

- Modo activo: Es el método original en el que el cliente FTP inicia la transferencia de datos, el servidor abre una conexión desde el puerto 20 para la dirección IP y un puerto aleatorio sin privilegios (mayor que 1024) especificado por el cliente. Debido a la existencia de internet muchos cortafuegos rechazan las peticiones a puertos mayores que el 1024. Por este motivo utilizare el modo pasivo, debido a la imposibilidad de transferencia de ficheros.
- Modo pasivo: El proceso es similar, el cliente FTP indica que quiere acceder a los datos de modo pasivo y el servidor proporciona la dirección IP y el puerto aleatorio mayor que 1024 sin privilegios. Posteriormente, el cliente se conecta al puerto en el servidor y descarga la información requerida. Y de esta manera el cliente resuelve la interferencia de los cortafuegos en el lado cliente.

2.2.4.3. MobaXterm

MobaXterm [12] es una herramienta todo en uno que busca llevar a los usuarios más profesionales de Windows determinadas funciones de Linux como el uso de los comandos más habituales para controlar el sistema operativo desde el teclado (por ejemplo cd, ls, bash, grep, awk, sed, rsync, etc.).

Las funciones de MobaXterm no se limitan únicamente a poder utilizar nuestro Windows con comandos Linux, sino que van mucho más allá. Gracias a esta herramienta vamos a poder disponer de una completa suite de herramientas todo-en-uno tales como:

- Cliente de conexión remota por terminal SSH, Telnet, rlogin, Mosh.
- Clientes de escritorio remoto como RDP, VNC, Xdmcp.

- Otras conexiones remotas: FTP, SFTP, etc.
- Servidor X integrado con aceleración gráfica.
- Posibilidad de utilizar varias sesiones con diferentes protocolos desde una ventana.
- Posibilidad de añadir nuevas funciones y herramientas mediante el uso de plugins.

Con este Herramienta se puede acceder a todos los entornos y servidores instalados en el proyecto.

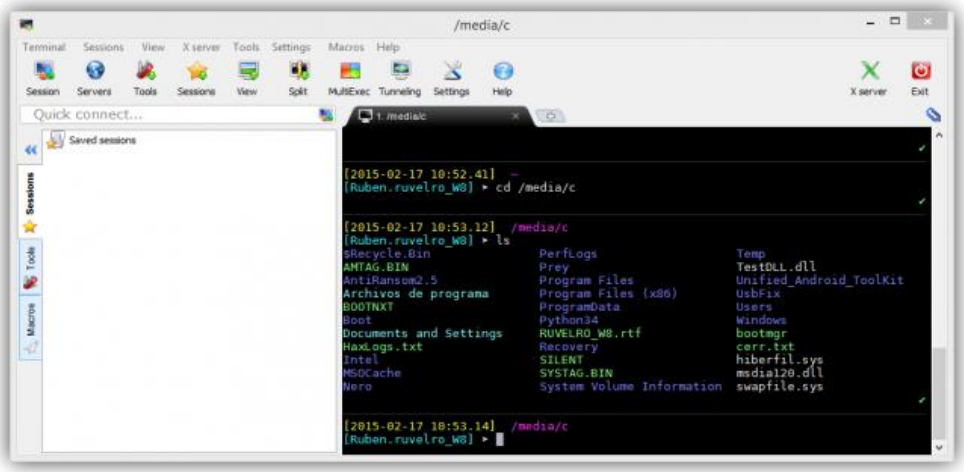


Figura 2.10. Herramienta de MoabaXterm.

2.2.5. Herramienta de supervisión y calificación

2.2.5.1. XYMON

La supervisión de los servidores HAPI es compatible con XYMON [13]. Su funcionamiento es en modo cliente-servidor. La elevación de las alarmas de advertencia y error está dirigida a un servidor con una ubicación precisa.

Xymon es una herramienta para monitorear servidores, aplicaciones y redes. Recopila información sobre el estado de sus computadoras, las aplicaciones que se ejecutan en ellas y la conectividad de red entre ellas. Toda esta información se presenta en un conjunto de páginas web simples e intuitivas que se actualizan con frecuencia para reflejar los cambios en el estado de sus sistemas. Es capaz de monitorear un amplio conjunto de servicios de red, servidores de correo, servidores web (HTTP simple y HTTPS encriptado), registros de aplicaciones de servidores locales, utilización de recursos y mucho más.

Gran parte de la información se procesa y almacena en archivos resolución reducida, que luego forman la base para proporcionar gráficos de tendencia que muestran cómo, los tiempos de respuesta del servidor web varían con el tiempo.

Las alarmas se pueden consultar en un interfaz WEB.

Figura 2.11. Herramienta de supervisión XYMON.

2.2.5.2. HP Quality Center

La herramienta de HP [14] para la gestión de pruebas nos ayuda a la hora de gestionar, organizar y trabajar en los clientes que optan por ella para sus proyectos. Para tener un control total y una organización completa para que la calidad de un proyecto sea excelente, Quality Center nos organiza el trabajo de forma sencilla con los diferentes módulos:

- El módulo de requisitos, que permite una organización total de los requisitos que el cliente define para que los cumpla el desarrollo.
- El módulo de pruebas, que se separa en Test Plan y Test Lab, en los cuales dejaremos todas las pruebas que vamos a realizar en base a esos requisitos y sus ejecuciones respectivamente.
- El módulo de defectos, permite crear y vincular defectos a diferentes requisitos y a las ejecuciones de los casos de prueba. De esta manera podremos ver de un solo vistazo nuestro trabajo y olvidarnos de dejarnos notas, apuntes en cuadernos y centralizarlo todo en una herramienta.

Figura 2.12. Herramienta de calificación HP-Quality Center.

2.2.6. Mantis Beug Tranker

La herramienta Mantis Bt [15] es ampliamente utilizada para la gestión de anomalías (errores de aplicación), es gratuita y está completamente desarrollada en PHP.

Es un software basado en una interfaz web que requiere una base de datos y un servidor web. Su principio es simple, consiste en la declaración de error de la computadora, la actualización del progreso de su resolución hasta su cierre.

Esta herramienta es muy útil cuando estás en una gran compañía o en compañías separadas. De hecho, permite optimizar el trabajo, también permite un gran ahorro de tiempo entre los desarrolladores, el gerente de proyecto y los evaluadores. La herramienta Mantis BT permite:

- gestionar usuarios y sus derechos de acceso;
- para administrar diferentes proyectos;
- para informar y seguir anomalías (anomalías o evoluciones);
- realizar resúmenes estadísticos de anomalías y producir informes

2.3. Tecnologías

En este aparatado hablaremos y describimos generalmente las diferentes capas de software implementadas, así como sus interacciones con nuestro modulo a implementar:

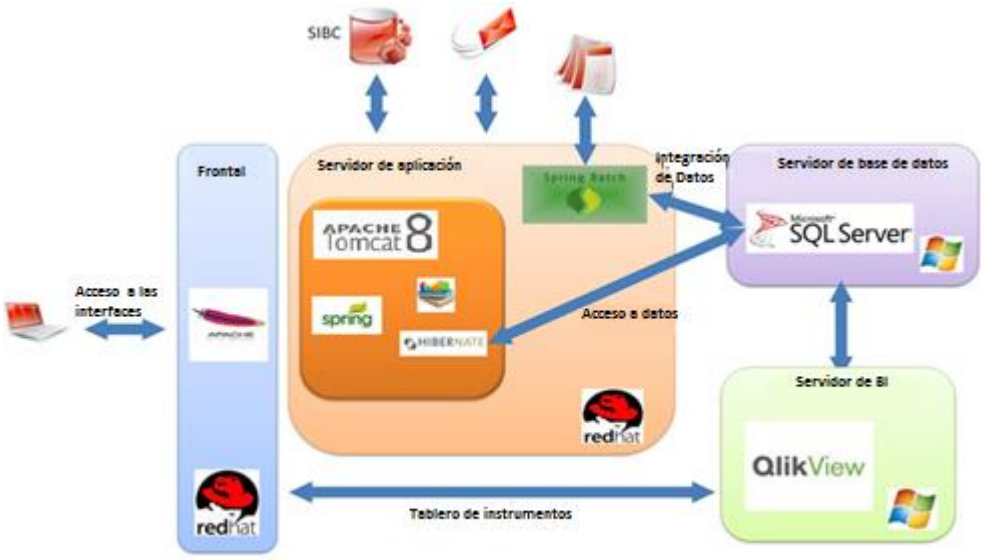


Figura 2.13. Arquitectura general de la aplicación [16].

2.3.1. Spring Framework

Spring [17]es un framework modular que permite construir aplicaciones Java ofreciendo soluciones a los problemas más comunes a los que se enfrentan los desarrolladores. Para ello dispone de multitud de módulos, que se pueden incluir de forma individual en la aplicación. La Figura muestra los módulos más importantes y cómo se agrupan dentro del framework. Una de las principales ventajas del marco de trabajo del marco de referencia, que permite a los desarrolladores poder encontrar un marco adecuado para el desarrollo del desarrollo de aplicaciones J2EE

La base de este framework es la Inversión del Control, o IoC por su nombre en inglés, también conocida como inyección de dependencias o Dependency Injection. En sentido amplio se puede decir que una aplicación Java consiste en un conjunto de objetos, o instancias de clases, que colaboran entre sí ofreciendo una serie de funcionalidades. Estos objetos tienen cierta dependencia unos de otros. Spring framework brinda soporte e integración a diversas tecnologías, por ejemplo:

- Soporte para la gestión de transacciones
- Soporte para interacción con diferentes bases de datos
- Integración con los marcos de relación de objetos para, por ejemplo, Hibernate, iBatis, etc.
- Soporte para Inyección de Dependencia
- Soporte para el estilo de servicios web REST

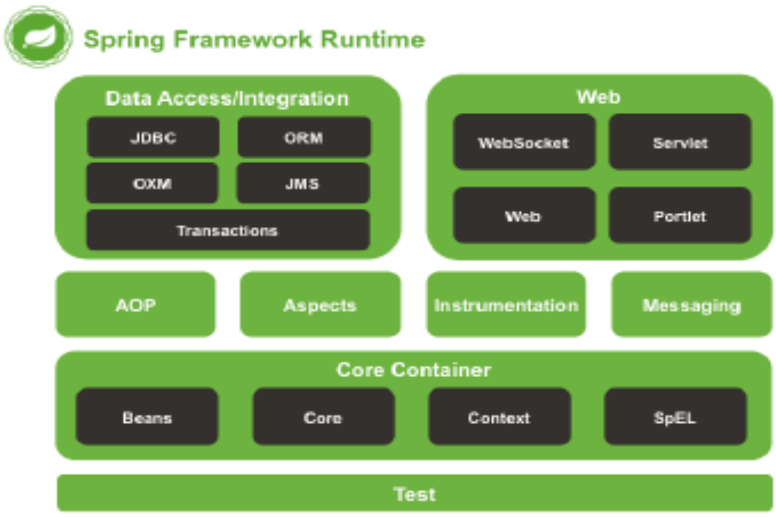


Figura 2.14. Arquitectura general Spring Framework.

Las versiones de Spring utilizadas en la aplicación HAPI son:

- Spring-core / spring-security: 4.3.2/4.3.2.RELEASE

2.3.2. Spring Batch

Spring Batch [18] es un marco de lotes ligero y completo diseñado para permitir el desarrollo de aplicaciones robustas de lotes vitales para las operaciones diarias de los sistemas empresariales.

Spring Batch proporciona las funciones reutilizables que son esenciales en el procesamiento de grandes volúmenes de registros, incluidos el registro / seguimiento, la administración de transacciones, las estadísticas de procesamiento de trabajos, el reinicio del trabajo, la omisión y la administración de recursos.

También proporciona los servicios técnicos y técnicos que permitirán los trabajos por lotes de alto volumen y alto rendimiento a través de las técnicas de optimización y división.

Las versiones de Spring utilizadas en la aplicación HAPI son:

- Spring Batch: 2.1.7.RELEASE

2.3.2.1. ¿Cómo funciona Spring Batch?

Un trabajo de Spring Batch consta de los siguientes componentes:

- ***Job***: Un trabajo representa el trabajo de Spring Batch. Cada trabajo puede tener uno o más pasos

- ***Step***: un paso hacia su trabajo. Esta es una gran herramienta para administrar las dependencias entre los trabajos, y también para modular izar la compleja lógica de pasos que se puede probar de forma aislada. El trabajo se ejecuta con los siguientes parámetros:
- ***ItemReader***: es una interfaz de estrategia para proporcionar datos. Se espera que la implementación aquí sea multicapa para cada lote. Cada uno puede llamar al método read () que regresará a un valor diferente
- ***ItemProcessor***: es una interfaz para transformaciones de elementos. Esta interfaz proporciona una extensión de la interfaz de la aplicación
- ***ItemStreamWriter***: es una interfaz para las operaciones de salida genéricas. La clase asume que esta interfaz será responsable de serializar los objetos según sea necesario. En general, es responsabilidad de la clase implementadora decidir qué tecnología utilizar para el mapeo y cómo se debe configurar. El método write () es responsable de asegurarse de que cualquier búfer interno se vacíe y si una transacción está activa, también será necesario descartar el resultado en una reversión posterior. El recurso para el cual el escritor normalmente podría manejar esto

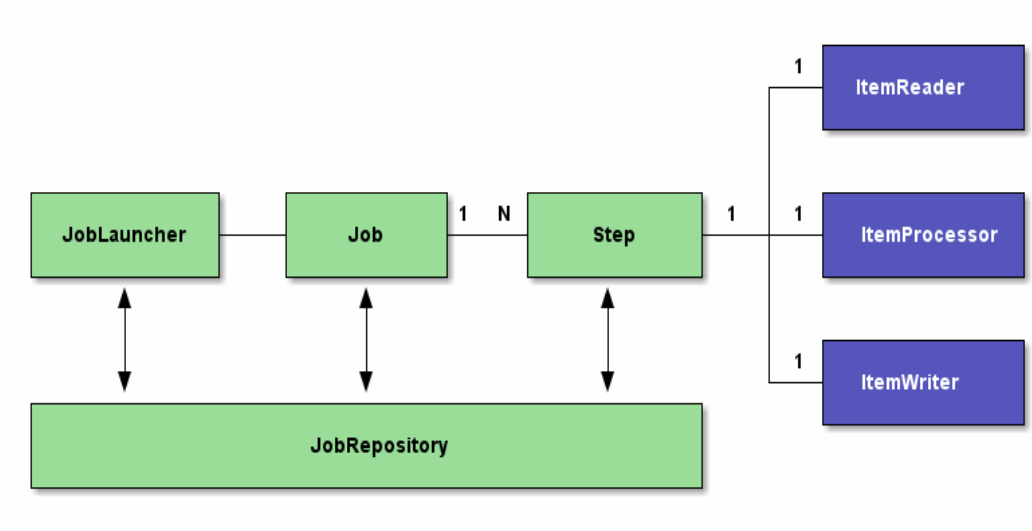


Figura 2.15. Componentes de Spring-Batch.

2.3.2.2. ¿Cómo Spring Batch nos puede ayudar?

Un Spring Batch proporciona las siguientes funciones para ayudarlo a resolver problemas múltiples:

- Ayuda a los desarrolladores a estructurar el código al proporcionar la infraestructura que se usa para implementar, configurar y ejecutar trabajos por lotes.
- Utiliza el procesamiento orientado a los fragmentos donde se compromete la transacción cuando se cumple el tamaño del fragmento. En otras palabras, proporciona a los desarrolladores una manera fácil de administrar el tamaño de las transacciones.
- Proporciona el manejo correcto de errores. Por ejemplo, los desarrolladores pueden omitir elementos si se lanza una excepción y configura la lógica que se usa para determinar si el trabajo por lotes debe reintentarse con la operación fallida. Los desarrolladores también pueden configurar la lógica que se utiliza para decidir si nuestra transacción se retrotrae o no.
- Escribe los registros completos en la base de datos. Estos registros contienen los metadatos de cada ejecución de trabajo y ejecución de paso.

En otros apartados explico cómo se ha utilizado este Framework y su manera de configuración dentro del IDE de Eclipse y veamos cómo implementar de Spring Batch.

2.3.3. Linux Redhat 6

Un sistema operativo seguro de superficie mínima optimizado para ejecutar contenedores Linux. Combina las capacidades flexibles, ligeras y modulares de los contenedores Linux con la fiabilidad y seguridad de Red Hat Enterprise Linux en un tamaño de imagen reducido.

Con Red Hat Enterprise Linux se puede:

- Distribuir sus aplicaciones con mayor rapidez.
- Reducir los esfuerzos de soporte y mantenimiento.
- Ampliar la portabilidad de contenedores en infraestructuras de nube híbrida abierta.
- Heredar la estabilidad y la madurez de Red Hat Enterprise Linux.

Uso dentro de la aplicación HAPI:

- Servidores frontales.
- Servidores de aplicaciones.

2.3.4. Windows Server 2012

Microsoft Windows Server 2012 es un sistema operativo de Microsoft orientado a servidor.

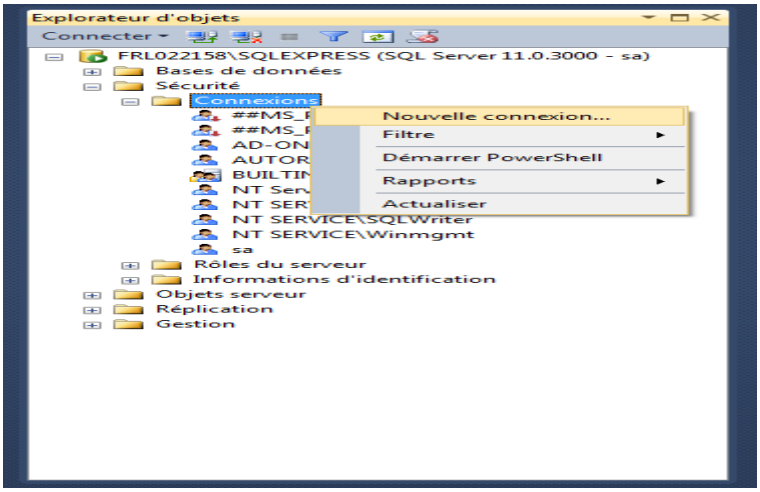


Figura 2.16. Base de datos Microsoft Windows Server 2012.

Uso dentro de la aplicación:

- Servidores de bases: SQL Server 2012.

SQL Server es un sistema de gestión de bases de datos relacionales (RDBMS) de Microsoft que está diseñado para el entorno empresarial. SQL Server se ejecuta en T-SQL (Transact -SQL), un conjunto de extensiones de programación de Sybase y Microsoft que añaden varias características a SQL estándar, incluyendo control de transacciones, excepción y manejo de errores, procesamiento fila, así como variables declarada

- Servidores que alojan la herramienta de informes y control: QlickView.

2.3.5. Tomcat

Apache Tomcat [19] funciona como un contenedor de servlets y es el escogido para desplegar la aplicación Web. Tomcat es una herramienta de código abierto que implementa un servlet Java, JavaServer Pages, Java Expression Lenguaje y tecnología WebSocket Java.

En el caso del proyecto se utiliza la versión 7.0.57 porque es una versión que es compatible con los frameworks y herramientas utilizados. Se ha escogido esta tecnología debido a que dispone de los mínimos requisitos que permiten desplegar la aplicación. Al llevar mucho tiempo en el mercado posee una documentación extensa. También es fácil de utilizar.

Tomcat es un servidor de aplicaciones Java desarrollado por Apache Software Fundación siguiendo el proyecto de Yakarta y capaz de alojar aplicaciones orientadas a la empresa:

- Respetuoso con los estándares J2EE (Webservices SOAP y Rest, JPA, JDBC, JMX, ...).
- Asegura consistencia e interoperabilidad (Spring, Hibernate, ...).
- Rendimiento y disponibilidad (clúster, caché, ...).

Uso: Servidor de aplicaciones que aloja Asignaciones de recursos IS (Aplicación HAPI).

2.3.6. Qlikview 11

QlikView [20] es una solución propuesta por el editor QlikTech. Permite realizar todas las funcionalidades de informes:

Es una plataforma de inteligencia empresarial (BI), permite crear aplicaciones decisionales de QlikView, conectadas a bases de datos o archivos planos (xls, csv, txt, qvd, xml...).Las aplicaciones QlikView BI le permiten

- Se ofrece a los encargados de la toma de decisiones todos los informes que deseen, y aún mejor, déjelos jugar con esos datos, para que puedan buscar lo que funciona o lo que necesita mejorarse.
- Proporcione al personal operativo herramientas claras para monitorear sus actividades (resultados de la semana, cuenta que debe reiniciarse, ...).

Visualmente, las aplicaciones de QlikView se parecen mucho a Excel: tablas de datos y gráfico.

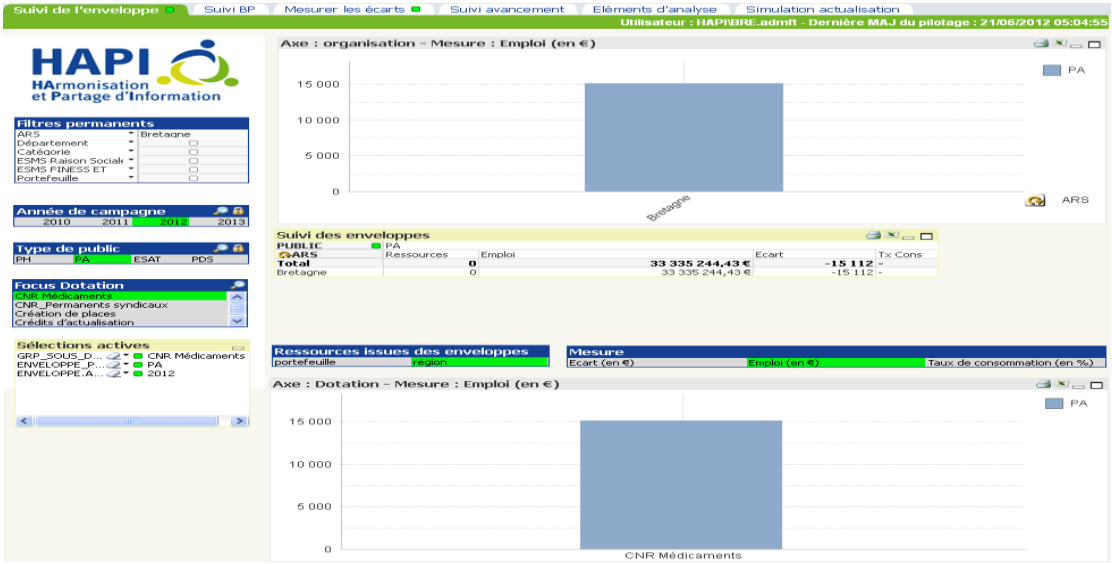


Figura 2.17. Plataforma de inteligencia QlickView.

2.3.7. Hibernate

Hibernate [21] es una herramienta de Mapeo objeto-relacional (ORM) para la plataforma Java. Este framework facilita el mapeo de atributos entre la base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos XML y anotaciones.

Hibernate es un framework para persistir objetos en una base de datos relacional, es decir, una solución para no tener que tratar directamente con la base de datos. Además proporciona:

- Productividad: Evita mucho código confuso en la capa de persistencia, permitiendo centrarse en la lógica de negocio.
- Mantenibilidad: Gracias a tener pocas líneas de código permite que sea más claro y fácilmente modificable.

- Rendimiento: Ofrece el mismo rendimiento que puede ofrecer una solución "manual".
- Independencia del proveedor: Es una solución ORM y esto permite abstraerse del SGBD (Sistema de Gestión de Bases de Datos) y desarrollar en local, con base de datos ligeras sin implicación en el entorno de producción.

La versión de Hibernate utilizada es 3.3.2.GA.

2.3.8. Atomikos

La biblioteca de Atomikos [22] permite administrar transacciones de confirmación en dos fases. Este mecanismo hace posible administrar la confirmación y la retrotracción de transacciones en las bases de datos HAPI y SIBC de forma agrupada.

La versión de Atomikos utilizada es 3.9.3.

2.3.9. Log4j

Log For Java [23] es una API de registro (logging) disponible por Apache Software Fundación.

- Muy común en el mundo de Java.
- Varios niveles de rastreabilidad: INFO, WARN, ERROR, DEBUG.
- Altamente configurable: archivo rotativo, salidas múltiples.

La versión log4J utilizada es 1.2.16.

2.3.10. IText

Es una PDF [24] que nos permite crear, adaptar, revisar y mantener documentos en el formato de documento PDF, algunas de las principales características son:

- Generar documentos y los informes extraídos de un fichero XML o de una base de datos
- Crear mapas y libros, incorporando características interactivas en formato PDF
- Añadir marcadores, números de página, marcas de agua, y otras características a los documentos PDF existentes
  Split o concatenar las páginas de los archivos PDF existentes
- Rellenar formularios interactivos.
- Servir generado dinámicamente o manipular documentos PDF en un navegador web.

La biblioteca iText utilizada es la versión 2.1.5.

2.3.11. Jasper Informa

Es una poderosa y flexible solución de código abierto para la generación y gestión de informes. JasperReports [25] es un módulo que dispone de un depósito de archivos que usa un sistema de carpetas, una aplicación web que muestra todos los informes que están en el depósito y un visor de dichos informes. La aplicación web permite subir todos los informes creados, e inmediatamente estos estás disponibles para todos los usuarios. Si por ejemplo, un informe dispone de algún parámetro, se puede crear un nuevo formulario que permita al usuario introducir nuevos parámetros.

- Generación de documentos RTF.

La versión de los informes jasper utilizada es 5.0.1.

2.3.12. JUnit

JUnit [26]es un conjunto de clases (framework) que permite realizar la ejecución de clases Java de manera controlada, para poder evaluar si el funcionamiento de cada uno de los métodos de la clase se comporta como se espera. Es decir, en función de algún valor de entrada se evalúa el valor de retorno esperado; si la clase cumple con la especificación, entonces JUnit devolverá que el método de la clase pasó exitosamente la prueba; en caso de que el valor esperado sea diferente al que regresó el método durante la ejecución, JUnit devolverá un fallo en el método correspondiente.

JUnit es también un medio de controlar las pruebas de regresión, necesarias cuando una parte del código ha sido modificado y se desea ver que el nuevo código cumple con los requerimientos anteriores y que no se ha alterado su funcionalidad después de la nueva modificación.

JUnit designa un marco para escribir y ejecutar pruebas unitarias.

- Marco común en modo Java.
- Facilidad de implementación.
- Automatización de pruebas unitarias.

La versión JUnit utilizada es 4.8.2.

JUnit se utiliza con easyMock versión 3.0, así como Spring-dbunit versión 1.1.12 y hsqldb versión 2.0.0 /

2.3.13. Apache Tiles

Apache Tiles [27] es un marco de composición de plantilla. Tiles se creó originalmente para simplificar el desarrollo de las interfaces de usuario de aplicaciones web, pero ya no está restringido al entorno web JavaEE.

Tiles permite a los autores definir fragmentos de página que se pueden ensamblar en una página completa en tiempo de ejecución. Estos fragmentos, o mosaicos, se pueden usar como simples incluye para reducir la duplicación de elementos comunes de la página o incrustados dentro de otros mosaicos para desarrollar una serie de plantillas reutilizables. Estas plantillas agilizan el desarrollo de una apariencia consistente en toda una aplicación. Es un framework de témplate para simplificar el desarrollo de interfaces de usuario. Tiles permite definir fragmentos y crear páginas que se ensamblan en tiempo de ejecución con estos fragmentos

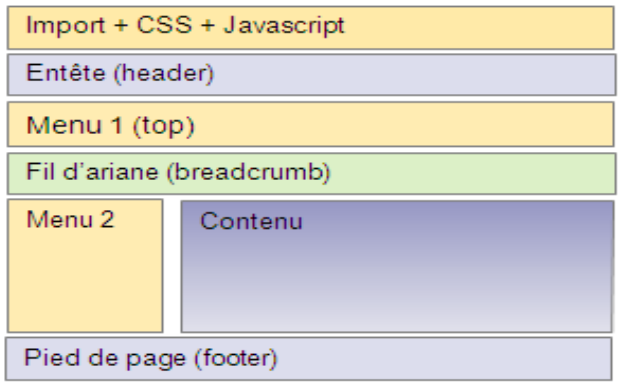


Figura 2.18. Mostración de Template final de la aplicación.

2.3.14. Ajax

Ajax [28] (Asynchronous JavaScript and XML), también conocido como asincrónico XML y JavaScript, es una arquitectura que permite construir sitios web dinámicos interactivos basados en diferentes tecnologías agregadas a los navegadores web. La idea de AJAX es hacer que una página web se comunique con un servidor web evitando la descarga de páginas completas y reduciendo el tráfico entre el cliente y el servidor. Un motor Ajax es en realidad un objeto JavaScript o función que se llama siempre que se solicite información del servidor. En lugar del modelo tradicional, que proporciona un enlace a otro recurso (como otra página web), cada enlace utiliza el motor Ajax, que planifica y ejecutar la consulta.

Este último se establece de forma asíncrona, lo que significa que la ejecución del código no espera una respuesta antes de continuar.

2.3.15. Tecnologías WEB

Las páginas web de la aplicación web están hechas de las siguientes tecnologías:

- Representación de los datos: HTML.
- Formato de datos: CSS.
- Interacciones dinámicas del usuario: JavaScript. El marco de JavaScript utilizado es JQuery.

3 ARQUITECTURA GENERAL DE LA APLICACIÓN

Este capítulo está dedicado a presentación del proyecto existente, la descripción detallada de proyecto, sus objetivos, su necesitad.

La aplicación HAPI (HA: armonización, P: Intercambio, I: Información) consta de los siguientes componentes:

- Aplicación web como un archivo WAR.
  Un archivo WAR debe implementarse en un servidor de aplicaciones Java.
- Aplicación Java como un archivo JAR.
  Un archivo JAR debe dejarse en un servidor de aplicación JAVA.

Apache Maven nativamente le permite dividir un proyecto en submódulos. Cada módulo es un elemento autónomo con su propio ciclo de vida. El concepto de módulo permite refinar la granularidad de los entregables y hacer que la arquitectura de la aplicación sea más flexible y escalable.

Cada módulo respeta el árbol recomendado por Maven:

- / src / main / java
  - código fuente de Java.
- / src / main / resources
  - Archivos de recursos (propiedades, xml, ...).
- / src / main / webapp
  - Aplicación web.
- / src / test / java
  - Código fuente de prueba java.
- / src / test / recursos
  - Probar archivos de recursos (propiedades, xml, ...).

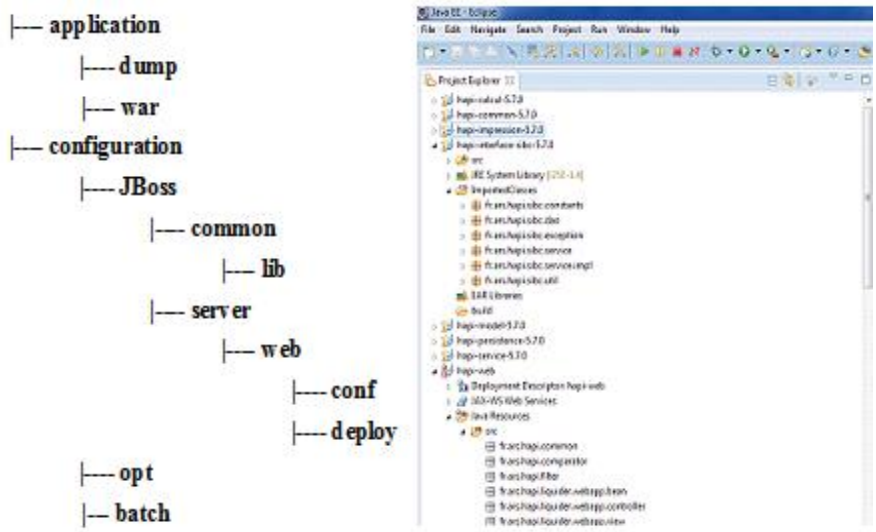


Figura 3.1. Estructura de la aplicación en Eclipse.

El archivo ‘pom.xml’ contiene, por un lado, el package del padre, el nombre del artefacto, el tipo de empaquetado (POM), y el nombre de la versión.

Por otro lado, el archivo contiene las dependencias (librerías necesarias) y los módulos de los que se compone. Las dependencias están definidas dentro del dependencyManagement. De esta manera, si se ha de cambiar la versión de una dependencia se puede hacer mediante la modificación de una línea y no de cada archivo ‘pom.xml’ que la contenga.

La siguiente figura muestra los servidores que tenemos en nuestra aplicación y en cada servidor lo que existe.

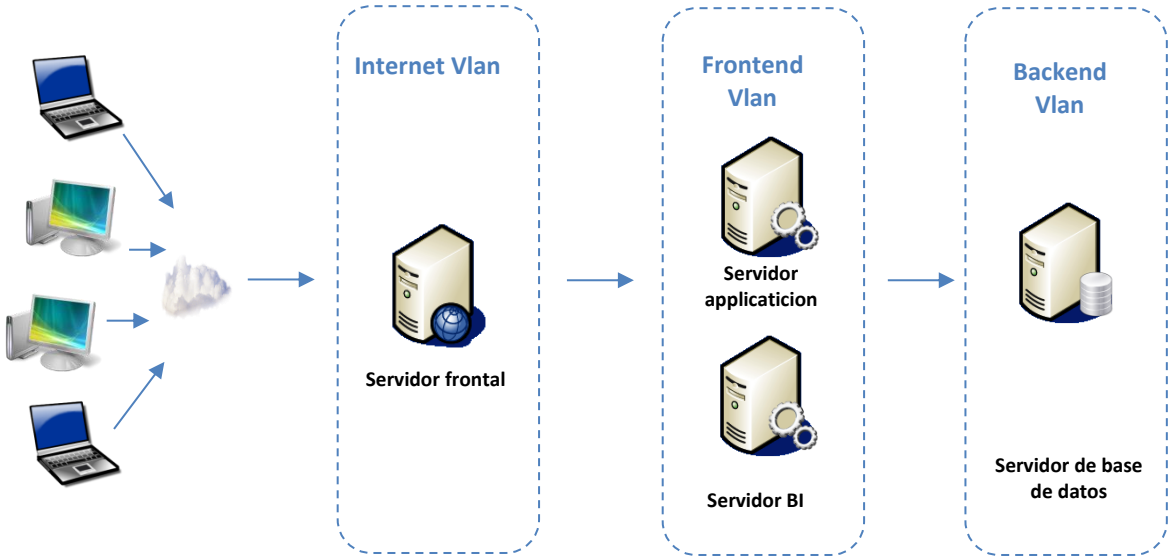


Figura 3.2. Servidores usados en las aplicación.

3.1. Capa Web

Se encuentra en el servidor web y contiene la lógica de presentación que se utiliza para generar una respuesta al cliente. Recibe los datos del usuario desde la capa cliente y basado en éstos genera una respuesta apropiada a la solicitud. J2EE utiliza en esta capa las componentes Java Servlets y JavaServer Pages para crear los datos que se enviarán al cliente. Este módulo contiene todos los elementos web:

- Implementando el patrón de diseño MVC con String MVC.
- Controladores.
  - Gestión de eventos.
  - Navegación.
  - Llamadas a la capa de servicio.
- Páginas web.
  - Archivos JSP: Ubicación: \ src \ main \ webapp \ WEB-INF \ jsp
  - Archivos JavaScript: Ubicación: \ src \ main \ webapp \ js
  - Archivos CSS: Ubicación: \ src \ main \ webapp \ styles
- Antecedentes de la aplicación web.
  - web.xml
  - hapi-servlet.xml
- Contexto de seguridad.
  - applicationContext-security.xml

3.3.1. Ejemplo de configuración de capa

- La declaración de context web : Archivo: src / main / resources / WEB-INF / web.xml
  - En nuestra aplicación web la configuración del contenedor se hace con la clase WebApplicationContext, definiéndola como un listener en el fichero descriptor de despliegue, web.xml. Contexto global.

    La clase ContextLoaderListener carga el fichero o ficheros XML especificados en el <context-param>llamado contextConfigLocation (suponemos que el fichero misBeans.xml está en el directorio WEB-INF). Como <param-value> se puede poner el nombre de varios ficheros XML, separados por espacios o comas.

```
<!-- Context configuration -->
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
        classpath:applicationContext-service.xml
        classpath:applicationContext-mail.xml
        classpath:applicationContext-authentication.xml
        classpath:applicationContext-security.xml
        classpath:applicationContext-qlikview.xml
    </param-value>
</context-param>
```

  - Declaración de página de error utilizada en la aplicación Las páginas de error sirven para informar a los visitantes de tu sitio web sobre los problemas del sitio. Cada tipo de problema se corresponde con un código establecido. Un visitante que acceda a una URL inexistente verá un *error 404*, mientras que un usuario no autorizado que intente acceder a un archivo restringido verá un *error 401*.y también se puede configurar un página personalizada como en nuestro ej

```
<error-page>
    <error-code>404</error-code>
    <location>/notfound.do</location>
</error-page>
<error-page>
    <exception-type>UnhandledException</exception-type>
    <location>/error.do</location>
</error-page>
```

  - Declaración de nuestro servlet de spring security. Declararemos un único filtro indicando que delegamos en *Spring Framework* odas las operativas de seguridad de la aplicación. *DelegatingFilterProxy* nos proporcionará el vínculo entre el *web.xml* y la *aplicación contexto* de *Spring*, y así aprovechar las facilidades a nivel de inyección de dependencias que el *framework* nos brinda.

```
<!-- Security -->
<filter>
    <filter-name>springSecurityFilterChain</filter-name>
    <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>
<filter-mapping>
    <filter-name>hibernateSessionFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
<filter-mapping>
    <filter-name>springSecurityFilterChain</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
<listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener>
```

- Declaración del servlet controlador principal de spring MVC. Para realizar la tarea de control de peticiones hace uso de un HttpServlet de implementación propia denominado DispatcherServlet. Podríamos decir, a groso modo, que este DispatcherServlet es la clase principal de Spring MVC, Este Dispatcher encapsula y nos abstrae de muchísimas funcionalidades estandarizadas en Spring que configuramos mediante la declaración de beans específicos o el uso de anotaciones.

```
<!-- Controller Principal MVC -->
<servlet>
    <servlet-name>hapi</servlet-name>
    <servlet-class>fr.ars.hapi.webapp.utils.HapiDispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>hapi</servlet-name>
    <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

- Declaración de contexto spring: Archivo: Hapi-servlet.xml
  - Gestión de capa de navegador: Spring proporciona algunos controladores que implementan la funcionalidad que a menudo se necesita en las aplicaciones web. uno de ellos es el intérprete de contenido web, que proporciona funcionalidad para la modificación fácil del comportamiento de almacenamiento en caché de una aplicación web. la configuración de este interceptor se haría de la siguiente manera en nuestra aplicación. la cantidad de segundos que se especifica para los encabezados de almacenamiento en caché de contenido, si esos encabezados se generan.

```
<!-- Gestion du cache navigateur -->
<!-- Modification du HTTP headers permettant de désactiver le cache du navigateur afin
    de permettre l'utilisation des boutons precedent / suivant du navigateur -->
<bean class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter">
    <property name="cacheSeconds" value="0" />
    <property name="useExpiresHeader" value="false" />
    <property name="useCacheControlHeader" value="false" />
    <property name="useCacheControlNoStore" value="false" />
</bean>
```

  - Declaración de vistas. Utilizando una configuración Tiles 2 en los vean de spring.

```
<!-- View Resolvers -->
<bean id="viewResolver"
    class="org.springframework.web.servlet.view.UrlBasedViewResolver">
    <property name="viewClass"
        value="org.springframework.web.servlet.view.tiles2.TilesView" />
    <property name="order" value="1" />
</bean>
<bean id="excelViewResolver"
    class="org.springframework.web.servlet.view.XmlViewResolver">
    <property name="order" value="2" />
    <property name="location" value="/WEB-INF/views-defs.xml" />
</bean>

<!-- Configuration du TilesConfigurer -->
<bean id="tilesConfigurer"
    class="org.springframework.web.servlet.view.tiles2.TilesConfigurer">
    <property name="definitions">
        <list>
            <value>/WEB-INF/pages-defs.xml</value>
        </list>
    </property>
</bean>
```

```
<!-- Layout de base -->
<definition name="layout" template="/WEB-INF/jsp/template/layout.jsp">
    <put-attribute name="top" value="/WEB-INF/jsp/template/top.jsp" />
    <put-attribute name="header" value="/WEB-INF/jsp/template/header.jsp" />
    <put-attribute name="footer" value="/WEB-INF/jsp/template/footer.jsp" />
    <put-attribute name="css" value="/WEB-INF/jsp/template/import_commons_css.jsp" />
    <put-attribute name="javascript" value="/WEB-INF/jsp/template/import_commons_js.jsp" />
    <put-attribute name="import" value="/WEB-INF/jsp/template/import.jsp" />
    <put-attribute name="breadcrumb" value="/WEB-INF/jsp/template/breadcrumb.jsp" />
    <put-attribute name="content"   value="" />
</definition>
```

- Declaración du contexto de seguridad: Archivo applicacionContext-security.xml
  - Poner url autorizadas.
  - Declaración de formularios para autentificarse.

```
<security:http use-expressions="true" access-denied-page="/denied.do"
    auto-config="false">
    <!-- no restriction -->
    <security:intercept-url pattern="/login.jsp"
        access="permitAll" />
    <security:intercept-url pattern="/security/**"
        access="permitAll" />
    <!-- static resources -->
    <security:intercept-url pattern="/styles/**"
        filters="none" />
    <security:intercept-url pattern="/images/**"
        filters="none" />
    <security:intercept-url pattern="/js/**"
        filters="none" />

    <!-- Restrict access to all other pages -->
    <security:intercept-url pattern="/**"
        access="isAuthenticated()" />

    <!-- login form -->
    <security:form-login login-page="/login.jsp"
        authentication-failure-url="/login.jsp?error=redirect" default-target-url="/load.do"
        always-use-default-target="true" />
    <security:logout logout-success-url="/login.jsp" />
</security:http>
```

  - Gestión de autentificación y contexto de usuario. El bean “authenticationManager” normalmente mantiene una referencia un bean "daoAuthenticationProvider" que realiza la autenticación mediante consultas a BD. Esta referencia se sustituye por nuestra clase generada manualmente en Java.

```
<!-- User Details Service -->
<bean id="customUserDetailsProvider" class="fr.ars.hapi.security.CustomUserDetailsProvider" />

<!-- Authentication provider use the data base-->
<bean id="daoAuthenticationProvider"
    class="fr.ars.hapi.security.CustomAuthenticationProvider">
    <property name="userDetailsService" ref="customUserDetailsProvider" />
    <property name="passwordEncoder" ref="passwordEncoder" />
</bean>

<!-- Authentication manager declaration -->
<security:authentication-manager>
    <security:authentication-provider
        ref="daoAuthenticationProvider" />
</security:authentication-manager>
```

3.2. Capa de negocio

La capa de negocio expone la lógica necesaria a la capa de presentación para que el usuario a través de la interfaz interactúe con las funcionalidades de la aplicación. La plataforma Java EE define el uso de componentes de negocio EJB para abstraer la lógica de negocio de otros problemas generales de las aplicaciones empresariales como la concurrencia, transacciones, persistencia, seguridad, etc.

3.2.1. Definición de servicios

- Se pueden usar desde la capa web (hapi-web) o desde otro servicio.
- Permiten acceder a los datos a través de la DAO.
- Implementan lógica de negocios y procesos complejos.
- Envío de correos electrónicos.
- Una interfaz Java.
  - Declaración de firma de métodos específicos (que no sean CRUD).
  - Package « **fr.ars.hapi.*.service** ».
  - Hereda la interface « **fr.ars.hapi.parametrage.service.GenericService** ». Permite el acceso directo para los métodos CRUD.

```
public interface CampagneService extends GenericService<Campagne, Long> {
```

- Una clase de implementación de Java.
  - Implementación de métodos de acceso a datos específicos (que no sean CRUD).
  - Implementación de servicios empresariales y procesos complejos.
  - Package « fr.ars.hapi.*.service.impl ».
  - Hereda la interface « **fr.ars.hapi.parametrage.service.impl.GenericManager** ». Permite el acceso directo para los métodos CRUD.
  - Implementa la interfaz de servicio.
  - Tiene la anotación "@Service". Permite la declaración del servicio en el contexto de la aplicación.

```
@Service("campagneManager")
public class CampagneManager extends
        GenericManager<Campagne, Long> implements CampagneService {
```

3.2.2. Declaración de contexto spring

Todo esto se puede utilizar en conjunto con las transacciones estándar del framework. Spring e Hibernate es una combinación muy popular. Algunas de las ventajas que brinda Spring al combinarse con alguna herramienta ORM son:

- Manejo de sesión: Spring hace de una forma más eficiente, sencilla y segura la forma en que se manejan las sesiones de cualquier herramienta ORM que se quiera utilizar.
- Manejo de recursos: se puede manejar la localización y configuración de los SessionFactories de Hibernate o las fuentes de datos de JDBC por ejemplo. Haciendo que estos valores sean más fáciles de modificar.

- Anejo de transacciones integrado: se puede utilizar una plantilla de Spring el para las diferentes transacciones ORM.

```
<bean id="txManager" class="org.springframework.orm.hibernate3.HibernateTransactionManager" >
    <property name="sessionFactory"  ref="sessionFactory" />
</bean>

<tx:advice id="txAdvice" transaction-manager="txManager" >
    <tx:attributes>
        <tx:method name="get*" read-only="true" isolation="READ_COMMITTED" propagation="REQUIRED" />
        <tx:method name="find*" read-only="true" isolation="READ_COMMITTED" propagation="REQUIRED" />
        <tx:method name="search*" read-only="true" isolation="READ_COMMITTED" propagation="REQUIRED" />
        <tx:method name="*" isolation="READ_COMMITTED" propagation="REQUIRED" />
    </tx:attributes>
</tx:advice>

<aop:config>
    <aop:pointcut id="servicePointcut" expression="execution(* fr.ars.hapi.parametrage.service.impl.*Manager.*(..))" />
    <aop:advisor advice-ref="txAdvice" pointcut-ref="servicePointcut" />
</aop:config>

<aop:config>
    <aop:pointcut id="servicePointcutSecurity" expression="execution(* fr.ars.hapi.security.impl.*Manager.*(..))" />
    <aop:advisor advice-ref="txAdvice" pointcut-ref="servicePointcutSecurity" />
</aop:config>
```

- Envolver excepciones: con esta opción se pueden envolver todas las excepciones para evitar las molestas declaraciones y los catch en cada segmento de código necesarios.
- Evita limitarse a un solo producto: Si se desea migrar o actualizar a otra versión de un ORM distinto o del mismo, Spring trata de no crear una dependencia entre la herramienta ORM, el mismo Spring y el código de la aplicación, para que cuando sea necesario migrar a un nuevo ORM no sea necesario realizar tantos cambios.
- Facilidad de prueba: Spring trata de crear pequeños pedazos que se puedan aislar y probar por separado, ya sean sesiones o una fuente de datos (datasource).
- Aunque hayamos proporcionado persistencia de las credenciales, si examinamos la base de datos podremos ver que las contraseñas todavía se almacenan en texto plano, algo intolerable para una aplicación corriendo en entorno de producción. Por suerte Spring Security trae un abanico de cifradores de contraseñas listos para usar. Podemos escoger entre algoritmos md4, md5, sha, sha-256, etc. Para enlazar el cifrado con el proveedor de autenticación debemos declarar el bean de la clase org.springframework.security.authentication.encoding.ShaPasswordEncoder (hemos elegido el algoritmo sha) y hacer la referencia de la siguiente forma:

```
<!-- Sha Password Encoder Service-->
<bean id="passwordEncoder" class="org.springframework.security.authentication.encoding.ShaPasswordEncoder" />
```

Y para cargar las propiedades de autentificación, se hace referencia a un fichero de propiedad donde declaramos variables que cambian dependiendo de los entornos de trabajo

```
<bean class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
    <property name="ignoreUnresolvablePlaceholders">
        <value>true</value>
    </property>
    <property name="locations">
        <list>
            <value>
                classpath:authentication.properties
            </value>
        </list>
    </property>
</bean>
```

3.3. Capa de Persistancia

3.3.1. Definición de DAO

El uso de DAO le permite alejarse de la manera en que se almacenan los datos en el nivel de objeto comercial. Los DAO del módulo hapi-persistence permiten un fácil acceso a los datos presentes en la base de datos. Por defecto, cada DAO tiene los métodos por defecto llamados CRUD:

- Creación
- Supresión
- Actualización

Estos métodos se heredan del módulo hapi-common. Es posible agregar métodos específicos distintos de los métodos CRUD predeterminados. Cada DAO se compone de:

- Una interfaz Java
- Declaración de firma de métodos de acceso a datos específicos (que no sean CRUD)
- Package « **fr.ars.hapi.*.dao** »
- Hereda de la interface « **fr.ars.hapi.dao.GenericDAO** » (hapi-common)

```
public interface ArsDao extends GenericDAO<Ars, Long> {
```

- Implementación de métodos específicos (distintos de CRUD)
- Paquete « **fr.ars.hapi.*.dao** »
- Hereda la clase « **fr.ars.hapi.dao.hibernate.GenericDAOHibernate** » ((hapi-common)
- Implementa la interfaz DAO. tiene la anotación "@Repository". Permite la declaración DAO en el contexto de la aplicación

```
@Repository("arsDao")
public class ArsDaoHibernate extends GenericDAOHibernate<Ars, Long>
    implements ArsDao {
```

3.3.2. Declaracion de context spring

Para solventar este problema y organizar mejor todos los grupos de parámetros que nuestro fichero de configuración pueda tener Spring nos provee de una clase de utilidad denominada *PropertyPlaceholderConfigurer*.

Esta clase nos permite extraer parametrizaciones del fichero de configuración de Spring y ubicarlas en ficheros de propiedades. De tal forma que tanto para nosotros como para los administradores sea más sencillo realizar modificaciones. En principio nos puede parecer bastante cómodo configurarlo de esta manera ya que podremos cambiar los parámetros a nivel fichero de configuración de Spring cuando lo necesitemos.

Sin embargo la realidad es que nosotros no seremos los que modifiquemos estos parámetros sino que habitualmente lo hará un perfil de administrador cuando la aplicación se despliegue o se mantenga en un entorno de producción. Esto complica un poco el tema ya que los administradores no suelen estar habituados a manejarse con este framework y no les resultara sencillo abrir un fichero XML que puede tener una estructura compleja y realizar modificaciones.

- Cargar las propiedades de la base en nuestra aplicación : « **database.properties** »

```
<bean class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
    <property name="ignoreUnresolvablePlaceholders">
        <value>true</value>
    </property>
    <property name="locations">
        <list>
            <value>classpath:database.properties</value>
        </list>
    </property>
</bean>
```

- Para la configuración de la parte hibernate de nuestra aplicación se va a utilizar la siguiente configuración. especificándole el **nombre** que le queramos dar y el tipo de **classe** con el que queremos. Después, especificamos las cuatro propiedades necesarias para el acceso a la base de datos y ya está, así de simple. Como en mi caso estoy utilizando una base de datos SQL el driver que utilizare puede diferir del que debéis utilizar vosotros según la base de datos que estáis usando.

```
<property name="hibernateProperties">
    <props>
        <prop key="hibernate.dialect">${hibernate.dialect}</prop>
        <prop key="hibernate.show_sql">${hibernate.show_sql}</prop>
        <prop key="generateDdl">${generateDdl}</prop>
        <prop key="hibernate.hbm2ddl.auto">${hibernate.hbm2ddl.auto}</prop>
        <prop key="hibernate.default_schema">${hibernate.default_schema}</prop>
    </props>
</property>
```

- Para configurar el **SessionFactory** debemos establecer su **id**, su **classe**, como en cualquier **bean**, y además tres propiedades. La primera es el Datasource que es obtenido o declarado en el servidor de la aplicación. simplemente lo referenciamos y listo, lo siguiente son los distintos archivos de mapeo que vayamos a utilizar, en nuestro caso también lo hemos creado ya, y por último la propiedad **hibernateProperties**, en la que se configurara nuestra SessionFactory.

```
<import resource="classpath:applicationContext-datasource.xml" />

<bean id="sessionFactory"
    class="org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean">
    <property name="dataSource" ref="dataSource" />

    <property name="annotatedClasses">
        <list>
            <value>fr.ars.hapi.model.BusinessEntity</value>
            <value>fr.ars.hapi.model.generated.Ars</value>
            <value>fr.ars.hapi.model.generated.Beneficiaire</value>
            <value>fr.ars.hapi.model.generated.Campagne</value>

<jee:jndi-lookup id="dataSource" jndi-name="java:hapiDS"/>
```

3.4. Capa de modelo

3.4.1. Mapeo Objeto-relacional

Object-Relational mapping, o sus siglas O/RM, ORM, y O/R mapping) es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos relacional, utilizando un motor de persistencia. En la práctica esto crea una base de datos orientada a objetos virtual, sobre la base de datos relacional. Esto posibilita el uso de las características propias de la orientación a objetos (básicamente herencia y polimorfismo). Hay paquetes comerciales y de uso libre disponibles que desarrollan el mapeo relacional de objetos, aunque algunos programadores prefieren crear sus propias herramientas ORM.

En la programación orientada a objetos, las tareas de gestión de datos son implementadas generalmente por la manipulación de objetos, los cuales son casi siempre valores no escalares. Para ilustrarlo, considere el ejemplo de una entrada en una libreta de direcciones, que representa a una sola persona con cero o más números telefónicos y cero o más direcciones. En una implementación orientada a objetos, esto puede ser modelado por un "objeto persona" con "campos" que almacenan los datos de dicha entrada.

En HAPI, el mapeo relacional de objetos se realiza a través de entidades JPA. Una entidad es una estructura compuesta de atributos, que representa un componente identificable de un dominio funcional, y potencialmente relacionada con otras entidades en el dominio.

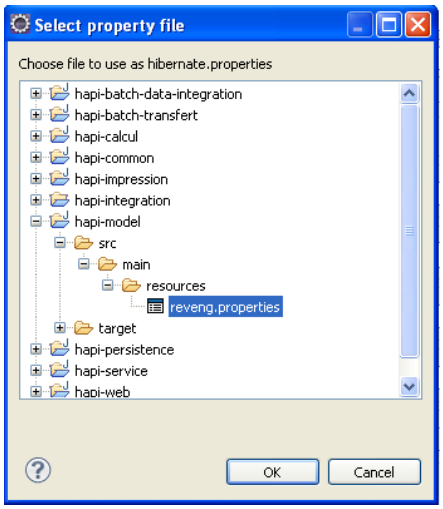


Figura 3.3. Fichero de propiedad en la estructura de módulos de la aplicación.

En Java, la API de persistencia (JPA) proporciona, entre otras cosas, las siguientes anotaciones:

- @Entity: permite convertir una clase Java en una entidad persistente
- @Table: especifica el nombre de la tabla afectada por el mapeo
- @Id: permite asociar un campo de la tabla a la propiedad como clave principal
- @Column: permite asociar un campo de la tabla con la propiedad

3.4.2. BusinessEntity

Una entidad es un objeto de dominio de persistencia ligero. Normalmente, una entidad representa una tabla en una base de datos relacional, y cada instancia de entidad corresponde a una fila en esa tabla. El artefacto de programación primario de una entidad es la clase de entidad, aunque las entidades pueden usar clases auxiliares. El estado persistente de una entidad se representa mediante campos persistentes o propiedades persistentes. Estos campos o propiedades usan anotaciones de mapeo relacional / de objeto para mapear las entidades y las relaciones de entidad con los datos relacionales en el almacén de datos subyacente.

En HAPI, cada entidad hereda de la clase BusinessEntity del paquete **fr.ars.hapi.model** del módulo hapi-model. Esta clase abstracta agrega para cada entidad los atributos "**idAuthor**" y "**dateEffet**" respectivamente correspondientes a las columnas "**ID_AUTHOR**" y "**DATE_EFFET**" de las tablas de la base de datos.

Estos atributos se completan con los métodos de la clase **fr.ars.hapi.service.impl.AuditChangeUtils** del módulo hapi-common.

- Ejemplo de una descripción general de la clase BusinessEntity:

```
@MappedSuperclass
public abstract class BusinessEntity < ID extends Serializable> implements Serializable {

    @Column(name = "ID_AUTHOR", length = 16)
    public String getIdAuthor() {
        return this.idAuthor;
    }

    @Temporal(TemporalType.TIMESTAMP)
    @Column(name = "DATE_EFFET", length = 23)
    public Date getDateEffet() {
        return this.dateEffet;
    }
```

- Ejemplo de implementación para una entidad:

```
@Entity
@Table(name = "ARS")
public class Ars extends BusinessEntity<Long> implements java.io.Serializable {

    @Id
    @Column(name = "CODE", unique = true, nullable = false, precision = 2, scale = 0)
    public Long getId() {
        return this.id;
    }

    @Column(name = "NOM", nullable = false, length = 40)
    public String getNom() {
        return this.nom;
    }

    @Column(name = "LIBELLE_COURT", length = 3)
    public String getLibelleCourt() {
        return this.libelleCourt;
    }
```

3.5. Capa de Batch

Nuestro sistema tiene una parte desplegada en una maquina Linux donde se encuentra la capa Batch de la aplicación, y otra parte la capa frontal de la aplicación y por último, su capa WAR está desplegada en un Tomcat. Cada una de las capas debería estar usando la misma versión sino la aplicación no tendría un funcionamiento correcto.

Es un aplicativo externo que carece de capa de presentación y tiene un único servicioque se encarga de persistir la información en la base de datos utilizando la funcionalidad

Esta capa esta creada independientemente de las otras capas con uso de Maven. Las dependencias en el pom.xml quedarían de la siguiente forma:

```
<project
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.
    xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi=
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <artifactId>hapi-main</artifactId>
        <groupId>fr.ars.hapi</groupId>
        <version>5.7.0</version>
    </parent>
    <artifactId>hapi-persistence</artifactId>
    <name>hapi-persistence</name>
    <dependencies>
        <dependency>
        <groupid>org.springframework.batch</groupid>
        <artifactid>spring-batch-core</artifactid>
        <version>2.2.0.RELEASE</version>
    </dependency>
    <dependency>
        <groupid>org.springframework</groupid>
        <artifactid>spring-context</artifactid>
        <version>3.2.2.RELEASE</version>
    </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
```

Para nuestro Batch-spring vamos a necesitar la configuración previa del contexto de este último, Contaremos con dos ficheros de configuración: BatchContext.xml. En ese fichero añadiremos los parámetros de configuración generales para cualquier proceso Batch y luego para cada necesidad se va actualizando con configuración concreta para nuestro proceso.

- JobRepository: es el componente encargado de la persistencia de metadatos relativos a los procesos tales como procesos en curso o estados de las ejecuciones.
- JobLauncher: es el componente encargado de lanzar los procesos suministrando los parámetros de entrada deseados.

```
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
http://www.springframework.org/schema/context
http://www.springframework.org/schema/context/spring-context.xsd">

    <context:annotation-config/>
    <context:component-scan base-package="fr.ars.hapi.model.*"/>

    <context:property-placeholder location="classpath:batch.properties.properties"/>


    <bean id="jobRepository"
          class="org.springframework.batch.core.repository.support.MapJobRepositoryFactoryBean">
        <property name="transactionManager" ref="transactionManager"/>
    </bean>

    <bean id="transactionManager"
          class="org.springframework.batch.suppor                                    ger"/>


    <bean id="jobLauncher"
          class="org.springframework.batch.core.launch.support.SimpleJobLauncher">
        <property name="jobRepository" ref="jobRepository"/>
    </bean>
```

Attribute : ref
A short-cut alternative to a nested "<ref bean='...'/>".

Data Type : string

Press 'F2' for focus

Vamos a necesitar también un fichero de propiedades que contiene la configuración general del Batch, que puede cambiarse de un entorno a otro. Utilizando variables que serán utilizadas en la implementación. En el ejemplo de la figura observamos las variables utilizadas para la base dato y el mail de envió.

```
hapi.batch.datasource.url=jdbc:sqlserver://sWinnapiqual01.ptx.fr.sopra:1433;tabas
hapi.batch.datasource.user=hapi_qual
hapi.batch.datasource.passsword =
mail.enabled    =true
mail.smtp.host=1
mail.smtp.port=25
mail.smtp.localhost =m
mail.from=ar
mail.to =   sebasti
mail.subject=   [HAPI - QUALIF] Probleme lors de l'execution du batch
```

4 PRUEBAS REALIZADAS

4.1. Evolución de la capa Batch

Como ya hemos hablado anteriormente para la implementación de nuestro proceso Batch del que constará el Módulo de Hapi-batch se utilizará el framework Spring Batch, explicado en partes anteriores de framework Spring.

Spring Batch provee muchas formas de ejecutar un Job. La más simple es utilizando la clase *CommandLineJobRunner*, el cual se puede ejecutar desde línea de comandos y recibe como parámetros iniciales el nombre del archivo de contexto XML donde está definido el Job y el nombre del bean del Job. En mi caso lo estoy ejecutando desde Eclipse y también se puede ejecutar en la máquina Linux donde están nuestros servidores de Calificación y producción, así que solo necesitamos crear una nueva "Aplicación Java" y en el "Main class" definir la clase CommandLineJobRunner.

Figura 4.1. Configuración en Eclipse lanzamiento de Batch.

El elemento principal de Spring Batch es el Job. Un Job está compuesto por varios Steps, que se ejecutan de forma secuencial. Todos los Jobs de Spring Batch deben implementar el interfaz JobLauncher. Esta interfaz presenta el método run, que recibe un Job y unos parámetros, y ejecutará el Job.

Spring Batch lleva un registro de todas las ejecuciones y parámetros con los que se lanzaron los Jobs. El encargado de guardar estos registros es el JobRepository, que cuenta con una implementación para almacenar esta información en base de datos o memoria.

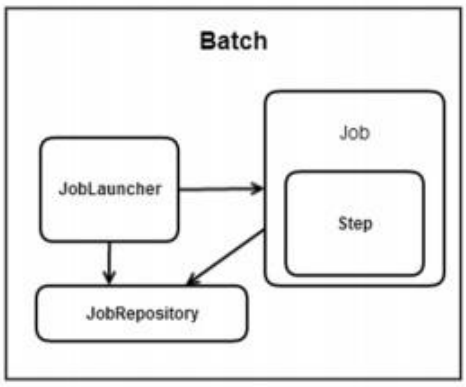


Figura 4.2. Esquema de Job de spring batch.

4.1.1. Tasklet

El procesamiento orientado a fragmentos no es la única forma de procesar en un Job. ¿Qué ocurre si un Job debe consistir en una simple llamada a un procedimiento almacenado? Puede implementar la llamada como un lector de elementos y devolver un valor nulo después del lote de Spring.

Pero es un poco antinatural ya que tendría que haber un ItemWriter que no funcionara. Spring Batch proporciona el TaskletStep [29] para este escenario. Tasklet es una interfaz simple que tiene un método, ejecute, que será repetidamente llamado por TaskletStep hasta que devuelva RepeatStatus.FINISHED o arroje una excepción para señalar un fallo. Cada llamada al Tasklet está envuelta en una transacción. Los implementadores de Tasklet pueden llamar a un procedimiento almacenado, un script o una simple declaración de actualización de SQL. Para crear un TaskletStep, el atributo 'ref' del elemento <tasklet /> debe hacer referencia a un bean que define un objeto Tasklet; no se debe usar ningún elemento <chunk /> dentro de <tasklet />.

```
<step id="step1">
    <tasklet ref="myTasklet"/>
</step>
```

Los Tasklets están destinados a realizar una única tarea dentro de un paso. Nuestro trabajo consistirá en varios pasos que se ejecutarán uno después del otro. Cada paso debe realizar solo una tarea definida. Nuestro trabajo constará de varios pasos. En este punto, todos nuestros pasos implementan la interfaz Tasklet. Eso nos obligará a implementar su método de ejecución, en el cuál es donde agregaremos la lógica para cada paso.

Fijamos que el método “execute” retorna un ExitStatus para indicar el estado de ejecución del Tasklet.

```
@Override
public RepeatStatus execute(StepContribution stepContribution,
  ChunkContext chunkContext) throws Exception {
    // ...
}
```

Para Nuestro Módulo Batch se creará un Job de varios Step para cada uno. El Job heredará de SimpleJob que es la implementación más simple de un Job y los Steps heredarán de Tasklet que también es la implementación más simple de un Step.

4.1.2. Planificación y requisitos:

Para la gestión de los tiempos de este proyecto se comenzará por identificar los trabajos necesarios y realizados mediante la realización de un EDT [30]. Con el EDT realizado se creará un diagrama de Gantt para determinar la estimación temporal de las tareas de manera aproximada a la realidad.

Desde el punto de vista del Proyecto HAPI, el ciclo de vida utilizado es un ciclo por incrementos. Sin embargo, a la hora de hablar del Módulo de gestión de datos (Batch) como proyecto, el ciclo de vida elegido es en cascada. Se utiliza un ciclo de vida distinto debido a que el desarrollo del Módulo se corresponderá con un incremento del proyecto que lo engloba, y como tal, constará de un solo entregable.

El ciclo de vida en cascada destaca por su sencillez en la gestión, ya que todas las actividades se desarrollan de manera lineal. El desarrollo secuencial de las actividades también favorece la estimación temporal, ya que, se basa exclusivamente en la suma de la estimación de cada actividad.

Uno de los posibles inconvenientes que podría tener la utilización de un ciclo de vida en cascada sería que si no se cuenta con todos los requisitos al inicio del proyecto, se pueden producir retrasos muy amplios debido a que haya que retomar etapas anteriores al detectar algún error.

Además el cliente tiene conocimientos técnicos que le permiten tener una compresión clara de los requisitos y podrá validarlos sin dificultad.

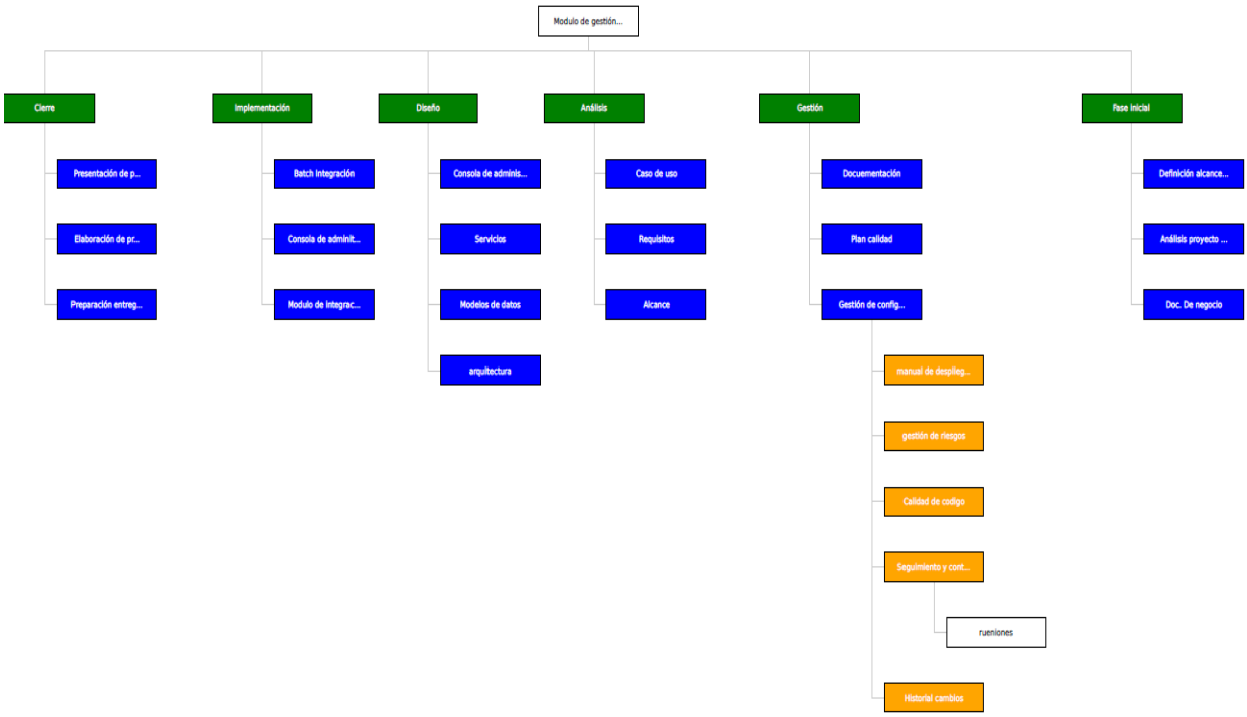


Figura 4.3. Diagrama EDT.

Respecto a la planificación temporal, el proyecto comienza el 28 de Mayo y se estima su finalización para el 09 de Julio de 2016. Se divide la planificación inicial en 5 fases: Iniciación, Estudio de Herramienta, Análisis, Diseño, Desarrollo, Pruebas y Documentación.

En todos los proyectos surgen cambios durante el desarrollo, bien sea por cambios en las estimaciones, en los requisitos, en los recursos… Muchas veces estos cambios provocan que haya que volver a planificar, con el fin de no perder el hilo del proyecto y tenerlo controlado.

En el Módulo se produjeron cambios que provocaron tener que volver a planificar. Los cambios que más influyeron fueron:

- Retrasos acaecidos en las tareas iniciales, respecto a la estimación original...
- La necesidad de definir de una manera más detallada las tareas planificadas para la fase de desarrollo, con el fin de poder realizar un seguimiento apropiado de esta fase del proyecto.

A continuación se muestra la planificación por separado de cada fase. La definición de cada tarea se pospone a la planificación final debida a que es la planificación que refleja con mayor realidad las actividades desarrolladas en el proyecto.

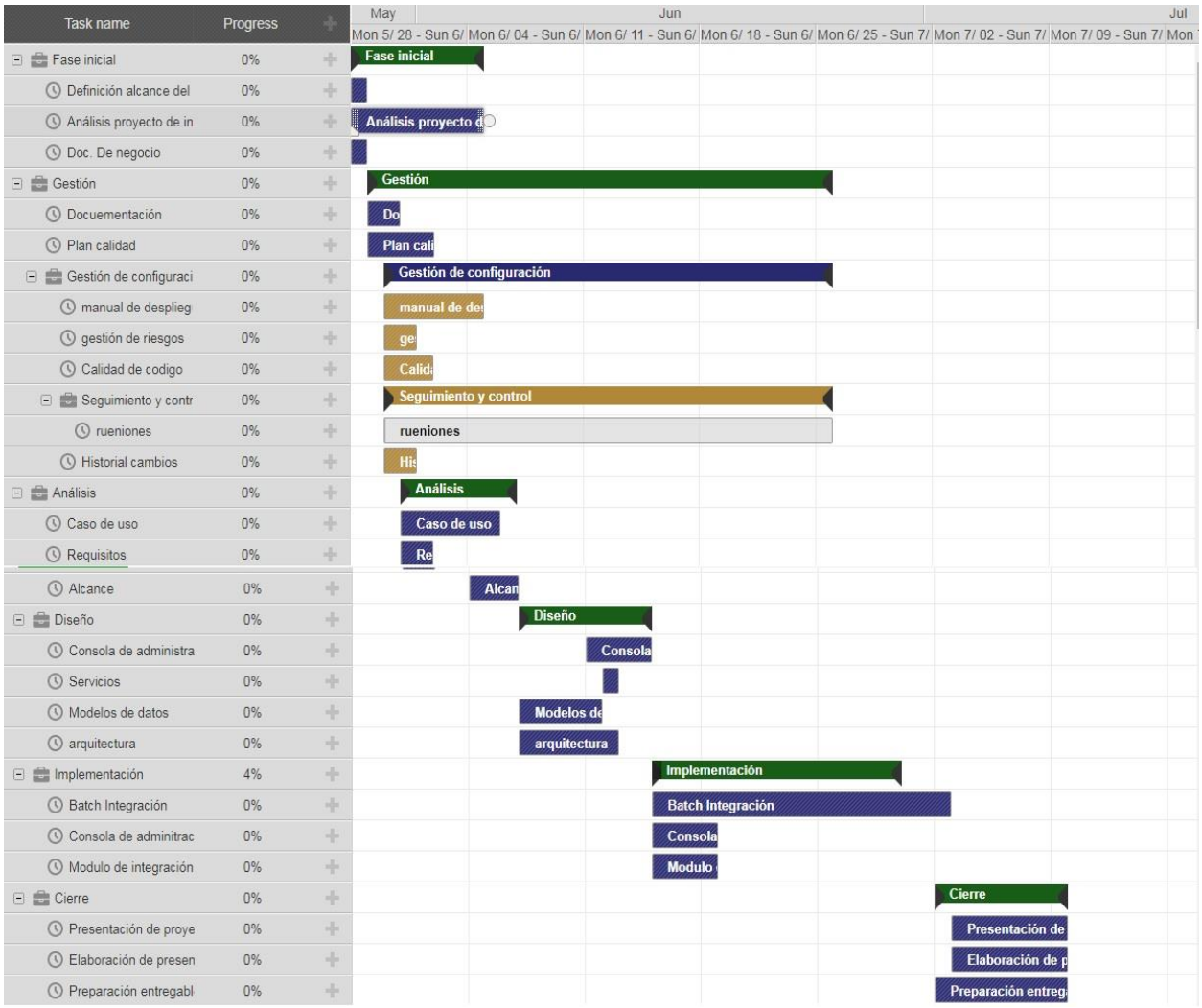


Figura 4.4. Diagrama Gantt.

Antes de comenzar por cualquier implementación, hay que tomar los requisitos del cliente. Los funcionales del proyecto generan un fichero con los detalles y el papel del módulo para ayudar de manera directa en el desarrollo de dicho módulo. Hay que tener en cuenta como resumen los siguientes puntos:

- Agregar o modificar datos del repositorio en la base de datos (en la modificación: no tocamos las claves principales de los datos).
- Eliminar y agregar datos comerciales a la base de datos (la eliminación se realiza de tal manera que la base de datos permanezca constante).
- La operación aquí es iniciar automáticamente este lote todas las noches para no tener que iniciarlo manualmente en las posiciones de desarrollador.
- Buscar archivos en el FTP (formato Excel) cuyo nombre corresponde a normas específicas, cargarlos en la base de datos, enviar en el FTP los intercambios de rechazos relacionados con la carga de documentos (los archivos sobresalen), para enviar los archivos FTP con los datos ingresados en la aplicación.

- Configurar la verificación de la buena transferencia de los archivos agregando una verificación de la suma de comprobación antes y después de la transferencia.
- Verificar el número de archivos antes y después de la transferencia.
- Si hay un problema durante la transferencia (por ejemplo, pérdida de conexión), el lote genera un registro de errores. No es necesario reiniciar el lote. El cliente volverá a enviar los archivos según sea necesario.
- Enviar de un correo electrónico con los registros de errores: Configurar este envío de correo, sabiendo que la aplicación web ya usa un servidor de correo. Implementación del servidor de correo.
- Bloquear el lanzamiento del lote en caso de que finalice un error o un lote en ejecución.
- Crear un archivo al comienzo del lanzamiento del lote, elimínar al final del proceso y no iniciar el lote si el archivo está presente.
- Operaciones de mantenimiento.
- Si hay un problema con el espacio en disco o la sobrecarga de la CPU: El lote puede ser eliminado (si se puede identificar), está programado para que pueda ser relanzado sin preocupaciones y reanudar el trabajo donde fue interrumpido.

4.1.3. Plan de calidad

A continuación se definirá el Plan de Calidad del proyecto, el objetivo de este es definir una serie de procedimientos y reglas que ayuden a que se han seguido para satisfacer las necesidades por las cuales se emprendió. Concretamente los procedimientos que formarán parte de la gestión de la calidad serán los procesos de Seguimiento y control del proyecto y el de Calidad de código.

4.1.3.1. Seguimiento

Debido a que la duración del proyecto es amplia y a que el tiempo de rectificación es muy pequeño es necesario definir varias actividades de seguimiento en la planificación del proyecto, con el fin de obtener un feedback del estado del proyecto en tiempo para aplicar cambios.

El proyecto sigue un ciclo de vida en cascada en el que podemos diferenciar claramente tres fases, Análisis, Diseño y Desarrollo/Pruebas. Se realizará un seguimiento, esto es así debido a que la duración de la fase de Desarrollo/Pruebas es muy amplia. Cada actividad de seguimiento constará de las siguientes tareas:

- *Evaluación de cambios*

Se revisará el Historial de cambios para evaluar todas aquellas peticiones de cambio que aún no hayan sido evaluadas. En el caso de que existan peticiones de cambio aceptadas se deberán de realizar, para ello a la hora de planificar las tareas de seguimiento y control se dejará un pequeño margen de tiempo.

- *Seguimiento de riesgos*

Es importante identificar los riesgos del proyecto en una fase inicial del proyecto pero más importante es realizar un seguimiento del estado de estos e identificar nuevos. En las tareas de seguimiento de riesgos se evaluarán aquellos riesgos a los que se les esté haciendo un seguimiento con el fin de ver si alguno se produjo y aplicar el plan de corrección. Además se eliminarán aquellos riesgos que ya no estén activos en el proyecto y se podrán identificar nuevos. La identificación de los riesgos se basará en las listas de riesgos identificados en anteriores proyectos de la empresa y en el análisis de los elementos del proyecto y sus posibles amenazas.

- *Reunión con directores*

La última tarea de seguimiento será realizar una reunión con los directores del proyecto. El objetivo de estas reuniones será primero informar a los directores del estado del proyecto y segundo obtener su visión del proyecto así como las peticiones de cambio oportunas.

Para organizar las reuniones el proceso será el siguiente:

- Se enviará la documentación a los directores vía email a medida que se vaya generando, con el fin de dar un margen de tiempo para leerla antes de la reunión.
- Se compartirá un documento de texto online con los directores en el que se represente el Acta de reunión.

  Las actas de reunión deberán tener los siguientes apartados:

  - Fecha y lugar de la reunión.
  - Nombre de los convocados.
  - Objetivos de la reunión, debe de ser el índice que guíe la
  - reunión. Conclusiones, permitirá tener un resumen de la reunión

4.1.3.2. Calidad de código

- ***Formato***

El Módulo de Batch se desarrolla dentro del entorno de desarrollo del Proyecto de HAPI, compartiendo recursos como son el repositorio de código y módulos comunes, esto provoca que para desarrollar ciertas funcionalidades haya que modificar módulos del Proyecto de Interoperabilidad.

La generación de código no puede ser descontrolada y debe de seguir unas reglas de formato, con el fin de facilitar la integración de los cambios con los demás desarrollos que se produzcan en la aplicación. De no seguir unas reglas de formateado, a la hora de integrar código nos encontraríamos con numerosos conflictos de código provocados por diferencias de formato.

Para que todos los desarrollos sigan un mismo formato se hará uso de las plantillas de formato que ofrece el entorno de desarrollo Eclipse. A continuación se muestra la configuración necesaria, dividida por lenguaje de programación. Para aquellos lenguajes que no se especifica la configuración deberá de definirse la configuración por defecto.

Con el fin de que el formateado de código sea una tarea automática se asociará la acción de Guardar con la de Formatear, de manera que cada vez que se guarde un archivo se le establezca el formato definido. En Windows->Preferences->Java->Editor->Save Actions definimos la siguiente configuración

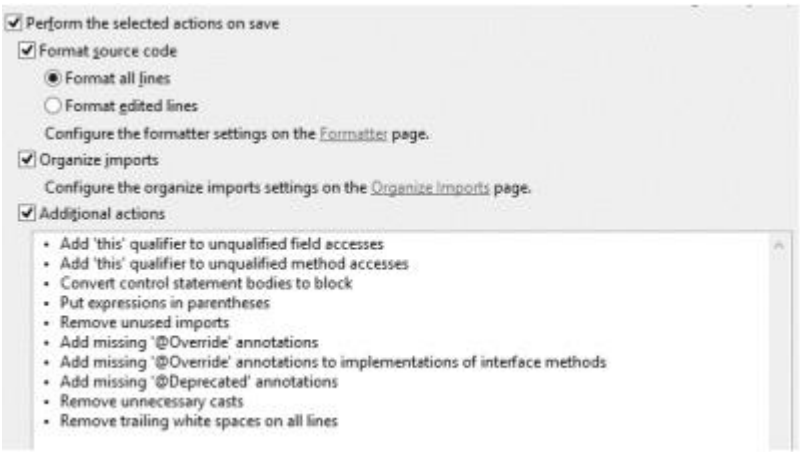


Figura 4.5. Configuración guardar-formatear

- ***Métricas y convenciones***

Para revisar la calidad del código implementado en función de métricas y convenciones utilizaremos la herramienta de código libre Sonar [31]. Sonar es una de las plataformas más populares para medir la calidad del código, permitiendo obtener métricas sobre diferentes lenguajes de programación. Su funcionamiento se basa en el uso de otras herramientas como PMD o Checkstyle.

En el entorno de desarrollo de la aplicación se cuenta con un servidor Sonar desplegado en la red interna del proyecto. Para integrar los proyectos del Módulo de Batch con este servidor hay una instalación del plugin de Sonar para Eclipse con configuración de la url del servidor. Una vez configurado el plugin de Sonar podremos analizar cualquier proyecto utilizando el comando de Maven sonar, este comando subirá el código del proyecto al servidor Sonar.

Con el proyecto importado al servidor Sonar podremos acceder a través de un navegador web y ver las diferentes métricas del código. De toda la información que nos ofrece Sonar la que vamos a tener en cuenta van a ser la Complejidad diplomática, Duplicación de código y las violaciones de buenas prácticas de programación.

La complejidad ciclomática es una métrica que nos aporta medición cuantitativa de la complejidad lógica de un programa. Sonar, calcula la complejidad ciclomática a través de un contador. Cuando el flujo de una función se altera, es decir, se produce un salto, este contador de complejidad aumenta en 1. Cada función tiene como mínimo una complejidad de 1. En el caso de Java el contador aumenta cada vez que se encuentra con alguna de las siguientes sentencias:

if, for, while, case, catch, throw, return, &&, ||, ?, else. Cabe destacar que las llamadas a getters y setters no aumentan el contador.

Sonar aporta información de la complejidad total, de la complejidad de clase y de la complejidad de método. Para el Módulo de Batch se establecerá un máximo de 2.5 de media de complejidad en los métodos. El motivo de fijar la complejidad sobre los métodos y no sobre las clases o sobre el sistema en general es facilitar las posibles acciones de corrección, ya que, a través de Sonar podemos identificar aquellos métodos cuya complejidad es más crítica, por lo que tendríamos el problema mucho más acotado. Además se establece un valor de 2.5 con el fin de al menos igualar la complejidad del resto de módulos ya implementados la aplicación

Uno de los puntos que más perjudicial a la hora de que un código sea fácil de mantener son las duplicaciones de código. Sonar aporta información acerca de la cantidad de líneas duplicadas, se utilizará esta información para identificar código que pueda ser reutilizado y así evitar la duplicación. En ciertas ocasiones la duplicación de código será inevitable por lo que se aceptará como máximo un 5% de código duplicado.

Sonar cuenta con una serie de reglas que permiten evaluar el cumplimiento de convenciones de programación Java, como pueden ser nomenclaturas, números mágicos, declaraciones, visibilidad de variables… Se deberán cumplir las convenciones de programación Java en al menos un 90%, siguiendo el grado de cumplimiento de los anteriores módulos de la aplicación

4.2. Etapas de implementación

Con la capacidad de agrupar los pasos dentro de un trabajo propio, surge la necesidad de poder controlar cómo el trabajo 'fluye' de un paso a otro. El fallo de un Paso no significa necesariamente que el Trabajo falle. Además, puede haber más de un tipo de "éxito" que determina qué Paso se debe ejecutar a continuación. Dependiendo de cómo se configure un grupo de Pasos, es posible que algunos pasos ni siquiera se procesen.

El escenario de flujo más simple es un trabajo donde todos los pasos se ejecutan secuencialmente. Para manejar escenarios más complejos, el espacio de nombres de Spring Batch permite que los elementos de transición se definan dentro del elemento de paso. Una de esas transiciones es el elemento "siguiente". Al igual que el atributo "Next", el elemento "Next" le indicará al Job qué Paso ejecutar luego. Sin embargo, a diferencia del atributo, se permite cualquier número de elementos "step" en un Paso dado, y no existe un comportamiento

predeterminado, el caso de error.

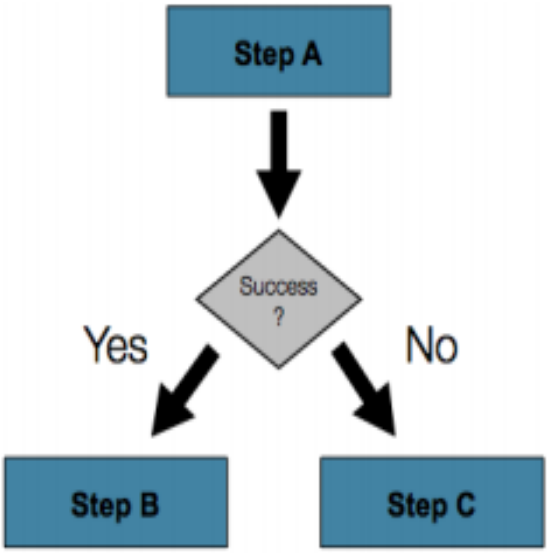


Figura 4.6. Configuración de step en spring Batch.

Esto significa que si se usan elementos de transición, entonces todo el comportamiento para las transiciones del Paso debe definirse explícitamente. Tenga en cuenta también que un solo paso no puede tener tanto un atributo "Next" como un elemento de transición.

```
<job id="job">
    <step id="stepA" parent="s1">
        <next on="*" to="stepB" />
        <next on="FAILED" to="stepC" />
    </step>
    <step id="stepB" parent="s2" next="stepC" />
    <step id="stepC" parent="s3" />
</job>
```

4.2.1. Creación de proceso de lanzamiento

Para nuestro lanzamiento de Batch vamos a crear como primera etapa, un Tasklet que creara por su parte el proceso que determinara se el Batch ha terminado su ejecución o no. Mientras este proceso está en marcha ningún otro Batch puede ejecutarse en nuestro servidor, como una medida de ahorro de JVM.

Cuando se lanza el primer step de nuestro Job, vamos a averiguar la referencia de este Job. Con esta última vamos a generar un fichero y lo dejaremos en una carpeta de trabajo

```
    private Long jobId;

    @BeforeStep
    public void getInterstepData(StepExecution stepExecution) {
        JobExecution jobExecution = stepExecution.getJobExecution();
        this.jobId = jobExecution.getJobId();
    }
```

Nuestro Tasklet va a averiguar si hay un fichero en una carpeta donde guardaremos los fichero de los ID proceso si hay algún Batch en ejecución. El Batch terminará con un mensaje de error indicando que no se pueden ejecutarse dos procesos a la vez. Si todo va bien se genera un fichero y continuará su trabajo el Batch a la siguiente etapa

```
<step id="PID"   >
      <tasklet ref="ConsultPIDTasklet" />
      <next on="COMPLETED" to="FtpStep"/>
      <next on="FAILED" to="DeletePID"/>
      <listeners>
            <listener ref="sampleListener"/>
      </listeners>
</step>
```

4.2.2. Recibir y enviar fichero vía ftp

En nuestro proyecto, he participado en la implementación de envió y la recepción de varios tipos de ficheros vía FTP de Spring Batch. Incluían recuperar archivos de un proveedor por FTP (generalmente con encriptación PGP) o SFTP y luego procesar los archivos. Tener el FTP en un solo paso lo hizo más fácil desde una perspectiva operativa ya que el FTP se convirtió en parte del trabajo. Entonces, por ejemplo, en el caso de un reinicio, el paso de FTP podría omitirse.

Para simplificar, la conexión FTP es común a todos los entornos de desarrollo existentes en el proyecto HAPI. Para distinguir los archivos que se procesarán para cada entorno disponible para el ARS, el servidor FTP se organizó de la siguiente manera.

En la raíz de FTP, hay un directorio por entorno:

- Entorno de integración (dedicado a desarrollo): integ.
- Entorno de calificación: calificado.
- Entorno de entrenamiento: forma.
- Entorno de producción: prod.

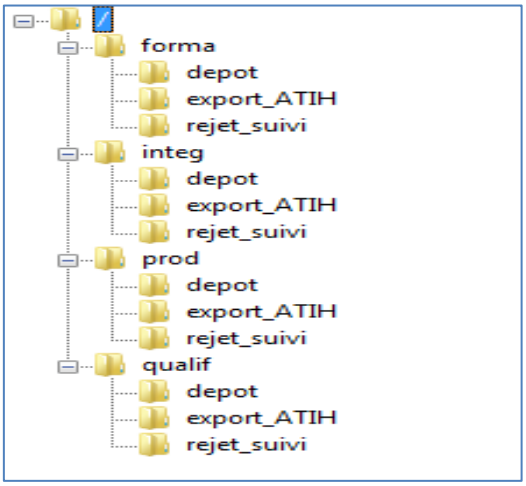


Figura 4.7. Repertorio en la comunicación FTP.

Para cada entorno, los archivos que se cargarán se deben colocar en el directorio de depósito. Cuando un archivo se recupera mediante el inyector de datos del entorno, se elimina del FTP.

Los archivos rechazados generados por el lote están disponibles en el directorio siguiente directorio. Estos solo están disponibles después de que se haya ejecutado el lote.

Para esto, configuramos nuestro fichero batchcontext.xml. Así que terminé creando un Tasklet que entre otras

```
<bean id="FtpGetRemoteFilesTasklet" class="fr.ars.hapi.batch.tasklet.FtpGetRemoteDepotTasklet" /

<job id="IntegrationAll">
    <step id="FtpStep"  >
        <tasklet ref="FtpGetRemoteFilesTasklet" />
        <listeners>
            <listener ref="sampleListener"/>
        </listeners>
    </step>
```

Sigue la implementación en Java del Tasklet de esta forma:

- Busqué en un sitio FTP archivos basados en un patrón de nombre de archivo y descargué los archivos.
- Configuré un intervalo de sondeo y una cantidad de intentos para ubicar un archivo (s).
- En la ejecución del Tasklet, utilicé FtpInboundFileSynchronizer y SftpInboundFileSynchronizer de Spring Integración para descargar los archivos desde el sitio remoto. También puede establecer el atributo retryIfNotFound en verdadero si desea volver a intentar la descarga. El comportamiento de reintento se puede configurar con los atributos downloadFileAttempts y retryIntervalMilliseconds.

```
import java.io.File;

public class FtpGetRemoteDepotTasklet implements Tasklet, InitializingBean
{
    private Logger logger = LoggerFactory.getLogger(FtpGetRemoteDepotTasklet.class);

    private File localDirectory;
/* (non-Javadoc)
 * @see org.springframework.batch.core.step.tasklet.Tasklet#execute(org.springframework.batch.core.StepContribution, o
 */
public RepeatStatus execute(StepContribution contribution, ChunkContext chunkContext) throws Exception
{
    deleteLocalFiles();

    ftpInboundFileSynchronizer.synchronizeToLocalDirectory(localDirectory);

    if (retryIfNotFound)
    {
        SimplePatternFileListFilter filter = new SimplePatternFileListFilter(fileNamePattern);
        int attemptCount = 1;
        while (filter.filterFiles(localDirectory.listFiles()).size() == 0 && attemptCount <= downloadFileAttempts)
        {
            logger.info("File(s) matching " + fileNamePattern + " not found on remote site. Attempt " + attemptCount +
            Thread.sleep(retryIntervalMilliseconds);
            ftpInboundFileSynchronizer.synchronizeToLocalDirectory(localDirectory);
            attemptCount++;
        }

        if (attemptCount > downloadFileAttempts && filter.filterFiles(localDirectory.listFiles()).size() == 0)
        {
```

4.2.3. Lectura de un fichero Excel

El lote reconoce los archivos usando su nombre. Por lo tanto, es importante que los archivos depositados para la carga se nombren de acuerdo con las reglas detalladas. Después de la carga de los ficheros Excel de nuestra Ftp y ponerla en nuestra carpeta de trabajo. La siguiente etapa de nuestro Batch será, detectar los nombres de ficheros que serán tratados en nuestras etapas de Batch y descomponer los ficheros si se contienen varias pestañas y colocar cada fichero en una carpeta de trabajo.

Para esto vamos a volver a configurar nuestro fichero xml, e implementar una nueva classe java con todas las reglas que el cliente nos ha exigido

```
<step id="DecoupExcel"  >
    <tasklet ref="DecoupExcelTasklet" />
    <next on="COMPLETED" to="MP24_P07A"/>
    <next on="FAILED" to="DeletePID"/>
    <listeners>
         <listener ref="sampleListener"/>
    </listeners>
</step>
```

En cualquier aplicación o desarrollo suele ser necesario procesar ficheros Excel u otro tipo de hojas de cálculo, en este caso nos vamos a centrar en Ficheros de Microsoft, y que manipularemos en este caso usando *Apache-POI*, que nos proporciona acceso a los diferentes tipos de ficheros de Microsoft que utilizan esta estructura como: Excel, Word o PowerPoint, también hay otros proyectos dentro de esta API para *Visio y Publisher* por ejemplo, de todos estos el más desarrollado es Excel Workbooks.

Con esta liberaría nos permite crear una hoja Excel, como añadir celdas, modificar su contenido y en definitiva manipular los datos según necesitemos, para ello se usa la librería POI-HSSF y POI-XSSF, donde HSSF es el proyecto POI de implementación total en Java para ficheros Excel.

- *HSSFWorkbook:* Representación de alto nivel de un libro (Workbook) que será nuestro documento excel. Es el primer objeto que construiremos si vamos a leer o escribir una hoja excel.
- *HSSFSheet:* representación de alto nivel de una hoja excel, podemos elegir la hoja de la excel usando el HSSFWorkBook.
- *HSSFRow:* representación de celda de una fila de la hoja excel, solo las filas que tienen filas se pueden añadir a la hoja.
- *HSSFCell:* representación de una celda en una fila de la un hoja de la excel, la utilizaremos para manejar el contenido de la celda.

Hay que tener cuenta que para utilizar esta librería la tenemos que añadir al proyecto con el que estamos trabajando, descargar la librería API Apache Poi Java para añadirla.

Después de la lectura de los ficheros Excel lo que vamos a hacer es transformar cada pestaña de Excel en un fichero CSV. CSV significa Comma-Separated-Values y es un formato muy común utilizado para intercambiar datos entre diversas aplicaciones. Si bien el formato del archivo Excel Spreadsheet es complejo (ya que tiene que acomodarse mucho más), CSV es un formato más simple que representa solo datos tabulares.

En este codigo, se muestra una forma de exportar los datos de una hoja de cálculo de Excel a CSV. Usamos la biblioteca de POI de Apache para este propósito.

```
Workbook wb = new XSSFWorkbook(new File(filnameXsl));
DataFormatter formatter = new DataFormatter();
PrintStream out = new PrintStream(new FileOutputStream(csvFile),
                                  true, "UTF-8");
for (Sheet sheet : wb) {
    for (Row row : sheet) {
        boolean firstCell = true;
        for (Cell cell : row) {
            if ( ! firstCell ) out.print(',');
            String text = formatter.formatCellValue(cell);
            out.print(text);
            firstCell = false;
        }
        out.println();
    }
}
```

Después de la transformación de las hojas Excel a fichero CSV, lo siguiente es poner cada fichero csv en una carpeta asociada a cada nombre de pestaña, que será una capeta de trabajo de las etapas de gestión de datos y comprobación de inserción

Estos archivos deben respetar las siguientes indicaciones:

- El formato de archivo debe estar en formato .xlsx solamente. Otros formatos de archivo no son aceptados
- En el nombre del archivo, los únicos caracteres permitidos son a-z, A-Z, 0-9,
- El nombre del archivo siempre debe estar precedido por el año de la campaña. Esto permite distinguir en qué campaña se cargarán los datos. Este número de campaña debe tener 4 caracteres numéricos (por ejemplo, 2018)
- El número de caracteres en las columnas "etiqueta corta" y "etiqueta larga" no debe exceder 256 (de lo contrario, la etiqueta no se trunca sino que se rechaza).
- La extensión del archivo debe ser xlsx
- El nombre del archivo debe contener: ReferentielMetier.

Todos estos archivos contienen un registro. Esto describe las pestañas utilizadas para cargar datos. El nombre de estas pestañas es muy importante. Si una pestaña no tiene el nombre correcto, no se cargará.

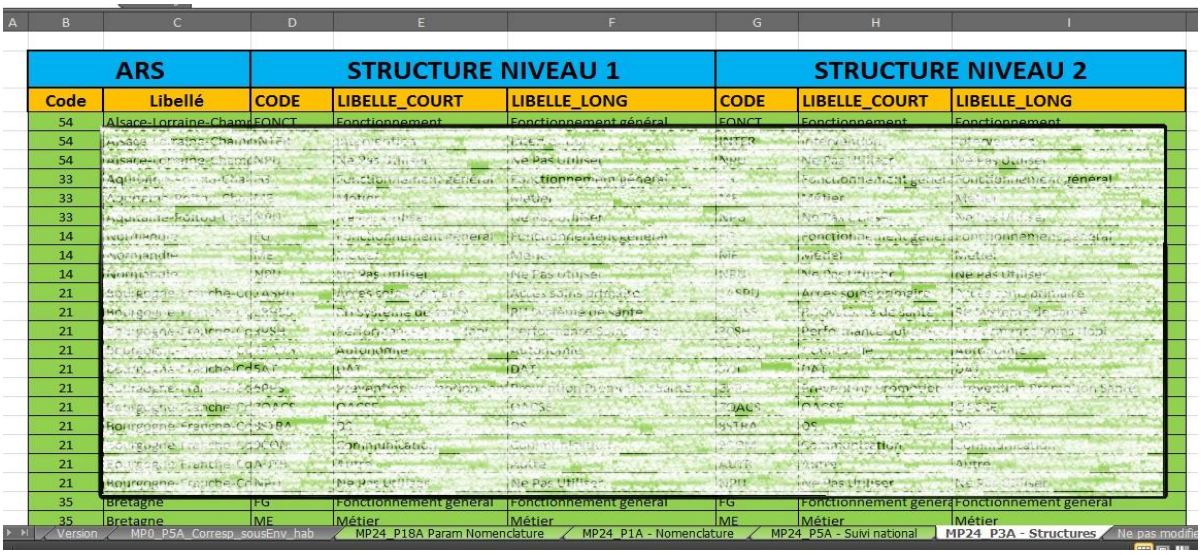


Figura 4.8. Ejemplo de Excel de entrada al módulo Batch.

4.2.4. Etapas de gestión de nuestro modulo

Las Etapas se lanzan de manera consecutiva, una tras otra. Solo si hay un fichero de CSV en la carpeta de trabajo de cada etapa, es ejecutada. Si no devolvemos un éxito en el tratamiento para poder seguir con el proceso hasta el último. Si hay algún problema blocante en cada etapa el proceso termina su trabajo.

El fichero xml se va a configurar igual que en las otras Etapas añadiendo el step adecuado para cada gestión.

```
.
.
.
<--! referentiel -->
<step id="MP24_P07A"  >
    <tasklet ref="MP24_P07ATasklet" />
    <next on="COMPLETED" to="COMMUNE_POSTAL"/>
    <next on="FAILED" to="DeletePID"/>
    <listeners>
         <listener ref="sampleListener"/>
    </listeners>
</step>

<--! referentiel -->
<step id="COMMUNE_POSTAL"  >
    <tasklet ref="ComunePostalTasklet" />
    <next on="COMPLETED" to="MP24_P8"/>
    <next on="FAILED" to="DeletePID"/>
    <listeners>
         <listener ref="sampleListener"/>
    </listeners>
</step>
.
.
.
.
```

Cada Etapa tendrá su propio Tasklet para la gestión del Excel con el modelo conceptual de nuestra base de datos de la aplicación. Para la gestión de base de datos (inserción, actualización, comprobación) se va a utilizar los servicios de capa de modelo y capa de persistencia. En cada Tasklet de cada etapa hay una previa comprobación en la base de datos de la existencia de los valores de entrada del Excel.

Para esto vamos a utilizar la tecnología de HQL hibernate para las consultas de la base de datos. Es el lenguaje de consultas que usa Hibernate para obtener los objetos desde la base de datos. Su principal particularidad es que las consultas se realizan sobre los objetos java que forman nuestro modelo de negocio, es decir, las entidades que se persisten en Hibernate. Esto hace que HQL tenga las siguientes características:

- Los tipos de datos son los de Java.
- Las consultas son independientes del lenguaje de SQL específico de la base de datos
- Las consultas son independientes del modelo de tablas de la base de datos.
- Es posible tratar con las colecciones de Java.
- Es posible navegar entre los distintos objetos en la propia consulta.

También en nuestros Tasklet generemos un Excel nuevo, donde guardamos las líneas de error de los ficheros de entrada. Este Excel será entregado por correo electrónico al final del proceso Batch. Como resumen de los errores de las líneas que no han sido tratadas.

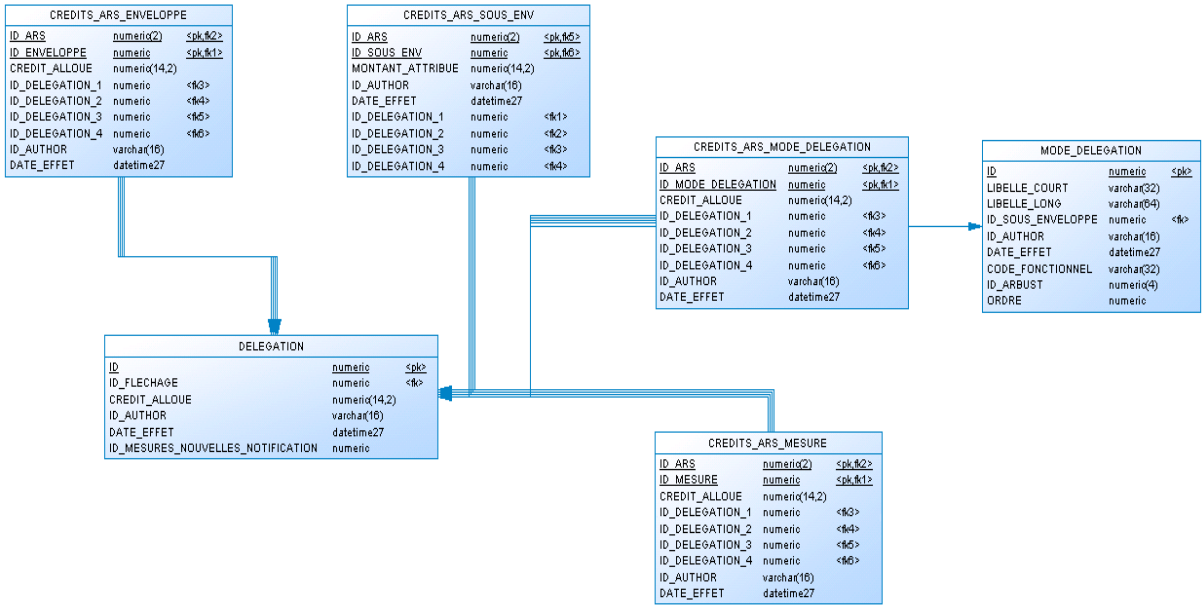


Figura 4.9. Ejemplo de Modelo conceptual y relacional.

4.2.5. Gestión de los mensajes de consola y excepción

Para la gestión de los mensajes de consola y la generación de un rastro de mensajes que servirán para el buen tratamiento del Batch, Vamos a utilizar Log4J es una estructura de registro diseñada para abordar los requisitos de registro de aplicaciones empresariales. Su predecesor, Log4J 1.x, existe desde hace más de una década y media y sigue siendo uno de los framework de registro de Java más utilizados. Proporciona varios componentes, como registradores, anexos y diseños que funcionan en conjunto para realizar el registro en una aplicación. Como diferentes aplicaciones tienen diferentes requisitos de registro, puede configurar LogJ 2 en consecuencia. Además, a menudo tendrá que seguir cambiando las configuraciones de Log4J 2 de una aplicación a lo largo de su ciclo de vida de implementación. Por ejemplo, es común establecer el nivel de registro en DEPURAR durante el desarrollo, y luego cambiarlo a ERROR para evitar llenar sus registros con información de depuración excesiva. De forma similar, durante el desarrollo local, puede trabajar con el apilador de la consola para evitar los gastos generales de E / S de archivos y en otros entornos de despliegue, establecer un apéndice de archivos u otro destino persistente para conservar los mensajes de registro.

Una de las configuraciónes que utilizaremos será, la generación de un fichero de log para cualquier traza que vamos a necesitar. A lo largo de nuestras Etapas de Batch en cada classe java, intentaremos en cada clase implementada tener una traca de lo que se ha hecho. Más cualquier error o algo similar a un error también serán trazados en el fichero de Log. La implementación en java para cada clase será de la siguiente forma.

```
private static Logger logger = LogManager.getLogger();

    logger.debug("Mensaje de depuracion de nuestro batch");
    logger.info("mensaje de informacion");
    logger.warn("mensaje de warning");
    logger.error("mensaje de error");
```

Considerando que algunas líneas del archivo de entrada podrían ser incorrectas desde el punto de vista del formato y ese no debería ser motivo para finalizar el Job, sino que se debería continuar con la siguiente línea

del archivo. Es importante tener en cuenta que cualquier excepción lanzada durante la ejecución del Job por defecto corta la ejecución del mismo.

```
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration debug="true" xmlns:log4j="http://jakarta.apache.org/log4j/">
   <Properties>
       <Property name="log-path">logs</Property>
       <Property name="archive">${log-path}/archive</Property>
   </Properties>
   <Appenders>
       <Console name="consoleAppender" class="org.apache.log4j.ConsoleAppender">
           <param name="Target" value="System.out"/>
           <PatternLayout>
               <pattern>%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n</pattern>
           </PatternLayout>
       </Console>
       <File name="fileAppender" fileName="${log-path}/integrationAll.log">
           <PatternLayout>
               <pattern>%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n</pattern>
           </PatternLayout>
       </File>

        <Loggers>
       <Logger name="com.example" level="debug">
           <AppenderRef ref="fileAppender" level="debug"/>
       </Logger>
       <Logger name="org.springframework" level="debug">
           <AppenderRef ref="fileAppender" level="debug"/>
       </Logger>
       <Logger name="org.springframework" level="debug">
           <AppenderRef ref="consoleAppender" level="debug"/>
       </Logger>
       <Root level="debug">
           <AppenderRef ref="consoleAppender"/>
       </Root>
   </Loggers>
```

4.2.6. Envió de Email

Una vez que la dependencia está en su lugar, el siguiente paso es especificar las propiedades del servidor de correo en el archivo application.properties.

```
hapi.batch.datasource.url=jdbc:sqls[redacted]
hapi.batch.datasource.user=hapi_qual
hapi.batch.datasource.passsword =[redacted]
mail.enabled    =true
mail.smtp.host=1[redacted]
mail.smtp.port=25
mail.smtp.localhost =mai[redacted]
mail.from=ars-quali[redacted]
mail.to =   s[redacted]
mail.subject=   [HAPI - QUALIF] Probleme lors de l'execution du batch
```

Vamos a utilizar el protocolo SMTP para el envío de correo. Es un protocolo que se usa para el envío de correo electrónico (Protocolo Simple de Transferencia de Correo Electrónico). SMTP transfiere los mensajes desde la máquina cliente al servidor, quedándose este último con el mensaje, si se trata del destino final.

Vamos a adaptar la configuración de nuestro servidor según la configuración y las credenciales de nuestro proveedor de correo electrónico. Una vez que se haya preparado JavaMailSender, lo único que queda es

preparar el mensaje en sí y luego enviarlo usando JavaMailSender.

Para preparar el mensaje en sí, Spring ofrece entre otras muchas posibilidades una útil interfaz Callback *MimeMessagePreparator*, que se puede usar para la preparación de mensajes MIME de JavaMail. Este preparador puede pasarse a javaMailSender mientras llama a send para enviar el mensaje.

```
public RepeatStatus execute(StepContribution contribution, ChunkContext chunkContext) throws Exception {
    //sendMailWithAttachment(from, to, title, message);
    sendMail(from, to, title, message);
    return RepeatStatus.FINISHED;
}


 */
private void sendMail(String from, String to, String title, String message) throws MailException {
    SimpleMailMessage msg = new SimpleMailMessage();
    msg.setFrom(from);
    msg.setTo(to);
    msg.setSubject(title);
    msg.setText(message);
    this.sender.send(msg);
}

/*
 *  Mail with attachment
 */
private void sendMailWithAttachment(String from, String to, String title, String message) throws MailException
    MimeMessageHelper mimeMessage = this.sender.createMimeMessage();
    try{
        MimeMessageHelper helper = new MimeMessageHelper(mimeMessage, true);
        helper.setFrom(from);
        helper.setTo(to);
        helper.setSubject(title);
        helper.setText(message);
        // Works!!
        ClassLoader classLoader = getClass().getClassLoader();
        File file = new File(classLoader.getResource("axslxttachment.xlsx").getFile());
        helper.addAttachment("", file);

    } catch (MessagingException e) {
        throw new MailParseException(e);
```

4.2.7. Generación JAR

A la hora de construir el proyecto con Maven hay que tener en cuenta los distintos build lifecycles o ciclos de vida de construcción. Existen tres ciclos de vida principales: default, que se encarga de la construcción y despliegue del proyecto en repositorios Maven; clean, encargado de limpiar el proyecto; y site que se encarga de generar el sitio web de documentación del proyecto.

Cada lifecycle está dividido en fases, y éstas a su vez en plugin goals u objetivos de plugins. Por ejemplo, el default lifecycle contiene las siguientes fases entre otras:

- validate: donde se valida que el proyecto es correcto y toda la información necesaria está disponible.
- compile: se compila todo el código fuente del proyecto.
- package: se empaqueta el código compilado en su correspondiente formato: jar, war,rar, ear, …

- install: se instala el paquete en el repositorio Maven local, permitiendo su uso en otros proyectos locales.
- deploy: se instala el paquete en el repositorio Maven remoto para compartirlo con otros desarrolladores y proyectos.

Después de terminar con toda la implementación de nuestro módulo usando svn vamos a subir todo nuestro trabajo en el repositorio. Después con la ayuda del pom de la parte Batch y Jenkins vamos a generar el punto JAR y ponerlo en la máquinas de prueba y de integración.

Nuestro entorno de Integración Continua estará compuesto por:

- Un contenedor "master" de Jenkins, que dirigirá las tareas
- Uno o varios contenedores "agentes" (también conocidos como "esclavos"), que ejecutarán las tareas

En los sistemas de Integración Continua lo habitual es tener un master (o varios en sistemas muy grandes) que coordina tareas y varios agentes que se encargan de ejecutarlas. Es habitual encontrar la palabra esclavos (slaves) o nodos (nodes) para referirse a los agentes. En Jenkins estos conceptos son sinónimos, en este trabajo haremos referencia a los mismos como agentes.

Para esto vamos a necesitar crear un nueva tarea en Jenkins en el gestor de tareas de Jenkins, donde lo más importante a tener en cuenta es:

- Configurar los datos de repositorio (Svn)
- Seleccionar código fuente en el repositorio
- Poner la ruta en el servidor linux donde se va a generar dicho JAR (mediante script sh)

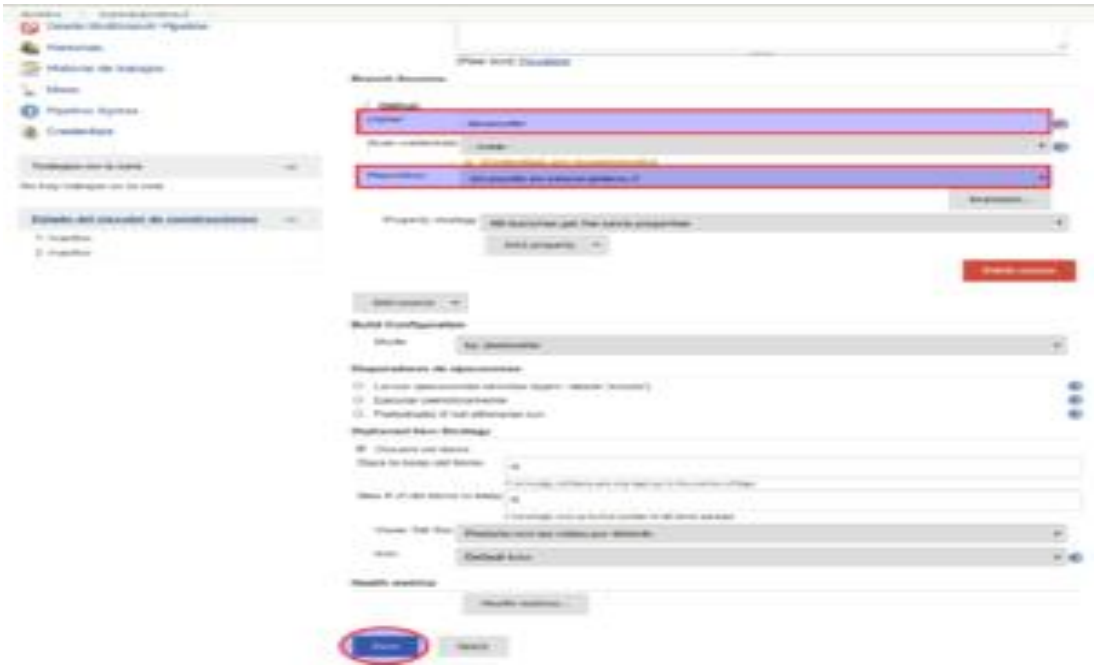


Figura 4.10. Vista Jenkins de configuración de tareas

4.2.8. Creación de cron

Para ejecutar un archivo JAR periódicamente. En el sistema * UNIX, puede usar el CRON [32] incorporado para programar un trabajo de planificador fácilmente. Cron es una utilidad básica disponible en sistemas basados en Unix. Permite a los usuarios programar tareas para ejecutar periódicamente en una fecha / hora especifica. Y, naturalmente, es una gran herramienta para automatizar muchas ejecuciones de procesos que de otro modo requerirían la intervención humana.

Cron se ejecuta como un proceso de "daemon", lo que significa que solo se debe iniciar una vez y se seguirá ejecutando en segundo plano. Este proceso hace uso de crontab para leer las entradas de los horarios y comienza las tareas.

Con el tiempo, el formato de expresión de cron se adoptó ampliamente y, a menudo, se puede usar en muchos otros programas y bibliotecas.

Para la configuración de una tarea periódica vamos a necesitar de implementar un fichero de extensión .sh que su role va ser la llamada a nuestro Batch para el arranque automático.

```
java -cp ".:/opt/batch/batch_hapi/batchIntegration/hapi-batch-data-integration-executable.jar:/opt/batch/batch_hapi/batchIntegration/"
org.springframework.batch.core.launch.support.CommandLineJobRunner batchContext.xml IntegrationAll
```

Un ejemplo de configuración de cron en una máquina linux, usando por Mohatren explicado en la parte de tecnologías es el siguiente:

<second> <minute> <hour> <day-of-month> <month> <day-of-week> <year> <command>

```
[hapibatch@arslinfe3871:~]# crontab -l
00 6 * * 1-4 /opt/batch/batch_hapi/start_batchs.sh
```

4.3. Pruebas unitarias y de integración

El desarrollo del Módulo de Batch sigue la secuencia: creación de las entidades del modelo, capa de acceso a datos, capa de servicios y por último vistas y procesos Batch.

Siguiendo la dinámica del desarrollo se comienza creando pruebas unitarias con JUnit para las clases de las capas más bajas como son las entidades y la capa de acceso a datos. Las pruebas unitarias tendrán el objetivo de validar todas las funcionalidades que aporta cada clase. Una vez terminado el desarrollo de las capas más bajas, entidades y capa de acceso a datos se cuenta con un conjunto de pruebas JUnit que son utilizadas como Pruebas de regresión para comprobar que todo lo implementado hasta el momento funciona correctamente.

Un conjunto de Pruebas de regresión no es más que un conjunto de pruebas que se utiliza para verificar que todo lo implementado anteriormente produce errores. Es importante destacar que la ejecución de las pruebas no asegura la ausencia de errores sino que revela la presencia de estos.

Es muy importante contar con un conjunto de Pruebas de regresión ya que, cualquier pequeño cambio que se pueda producir en el desarrollo futuro, puede producir que código que antes funcionaba y parecía que no tenía relación alguna con los desarrollos siguientes, deje de funcionar.

Hasta este punto, se ha hablado de pruebas para las implementaciones de las capas más bajas. Para probar las capas superiores también se desarrollan pruebas JUnit aunque, en este caso, ya podemos hablar de Pruebas de integración más que de Pruebas unitarias.

En el desarrollo de un servicio se pueden estar utilizando varias clases de acceso a datos o incluso

funcionalidades de otros servicios. Esto evidencia que, aunque el servicio funcione correctamente, éste depende tanto de una buena implementación, como de que el resto de módulos/unidades que utiliza lo también lo estén. Es por ello que hablamos de Pruebas de integración y no de Pruebas unitarias.

Se mantendrá el uso de JUnit para la comprobación de las funcionalidades de las capas superiores ya que, al contar con el conjunto de Pruebas regresivas, si se produce un error podemos asegurar que es debido a la funcionalidad que estamos probando. Lo podemos asegurar porque lanzando las Pruebas de regresión comprobamos que el conjunto de módulos/unidades de las que depende la funcionalidad a probar están funcionando correctamente.

5 CONCLUSIÓN

En este capítulo se expone un resumen del procedimiento seguido para el desarrollo de este proyecto y se comentan algunas conclusiones al respecto. Además, se detallan diferentes líneas de trabajo que se pueden seguir para la continuación del proyecto, indicando de manera objetiva los puntos en los que se puede mejorar. Finalmente, se exponen algunas conclusiones personales.

En los apartados que se han ido tratando en este documento se ha detallado el proceso de creación de un módulo en spring Batch que cumpliera con los objetivos expuestos al comienzo de este proyecto.

En los primeros apartados se trató de introducir el contexto de la aplicación, analizándolo en detalle con el fin de explicar la solución que ha sido propuesta que se acercara al problema real. Así, se llegó al enunciado de los requisitos funcionales, y no funcionales, que supusieron la base sobre la que comenzar la implementación. Una vez enunciados los requisitos, se pasó a un análisis de la tecnología. En este análisis surgió Spring Batch, que supuso todo un descubrimiento. La idea de crear aplicaciones con módulos que funciona en segundo lugar rica en Java era algo que generaba un enorme abanico de posibilidades para solucionar el problema anteriormente expuesto. Pero para ello, fue necesario un largo proceso de documentación sobre esta tecnología, así como la generación de ejemplos sencillos que permitieran el aprendizaje.

Cuando ya se dispuso de los conocimientos suficientes de la tecnología, se comenzó a analizar cómo se debía estructurar el trabajo, llegando así, a la parte de análisis de la arquitectura. Así fue como, dividiendo el trabajo en la parte del Batch, y la del servicio y negocio, se pudo delimitar que debía realizar cada una de estas capas. Por tanto, se detallaron las funcionalidades principales de cada uno de las partes. Para la realización de la capa Batch se partía completamente de cero. Además, fue necesario completar algunas funciones en la API de acceso a datos, pues no contemplaba en su totalidad algunas funciones ahora necesarias. Esto, unido a la nueva gestión de librerías con Maven sitúa tecnológicamente a este proyecto un escalón por encima en los proyectos que he desarrollado.

Los objetivos del proyecto han sido cumplidos y por lo tanto el resultado alcanzado es satisfactorio. Pero, además de haber cumplido los objetivos del proyecto cabe destacar que:

- La solución alcanzada ha sido diseñada en función del resto de módulos de la Plataforma de HAPI, permitiendo la fácil comprensión del módulo por parte de futuros desarrolladores que vayan a realizar tareas de mantenimiento y evolutivos sobre el Módulo.
- El diseño utilizado permite incorporar nuevos procedimientos sin tener que modificar ninguna línea de código del Módulo. Esto es muy importante debido a que el crecimiento del Módulo se basará en la incorporación de nuevos procedimientos que gestionaras nuevos ficheros o nuevas etapas.

La primera vez que conocí el proyecto, me impacto la forma de gestionar datos a través de una aplicación web. Desarrollar aplicaciones de fácil acceso y con un sinfín de funcionalidades y posibilidades en un entorno web sin necesidad de instalaciones costosas o de requerimientos inasumibles para algunas máquinas. Una vez conocí la forma en la que funcionaba internamente la aplicación, se abrió un abanico de posibilidades de mejora tomando como primera opción la idea de mejorar el funcionamiento interno de la aplicación. Esto supuso un vuelco total a la hora de gestionar el tratamiento de los datos y las llamadas a la API de gestión.

El tratamiento interno de los datos ha sido una parte costosa. La implementación del módulo Batch ha significado una mejora cualitativa importante abriendo un nuevo abanico de posibilidades para futuras aplicaciones. Esto unido a la implementación on de la gestión de librerías con Maven hace mucho más fácil el futuro mantenimiento de la aplicación. Todas estas nuevas tecnologías que he aprendido, hacen que el proyecto haya sido un éxito a nivel personal y laboral dado que el conocimiento adquirido me permite desarrollar diferentes ideas de implementación en diversas aplicaciones en mi entorno de trabajo.

Por último, otro de los apartados que he querido mejorar una vez finalizado el proyecto ha sido el relativo a la herencia de este proyecto para futuros desarrolladores, dejando una documentación más adecuada para el futuro desarrollo de la aplicación, que es algo que yo eche en falta cuando escogí este proyecto

REFERENCIAS

[1] I. Scrum, «https://clickee.wordpress.com/2015/02/19/planteamiento-de-la-pregunta-problema-scrum/,» 2015. [En línea].

[2] Sopra, «Documento de gestion de proyeto Hapi,» 2017.

[3] A. Maven, «https://maven.apache.org/,» [En línea].

[4] J. D. K. (JDK). [En línea].

[5] Eclipse, «http://www.eclipse.org/,» [En línea].

[6] SVN, «https://subversion.apache.org/,» [En línea].

[7] Jenkins, «https://es.wikipedia.org/wiki/Jenkins,» [En línea].

[8] R. K. Soni, «Spring: Developing Java Applications for the Enterprise».

[9] Sonar, «Sonar. http://www.sonarsource.org/,» [En línea].

[10] SSH, «http://web.mit.edu/rhel-doc/3/rhel-rg-es-3/ch-ssh.html,» [En línea].

[11] F. T. P. (FTP), «https://es.wikipedia.org/wiki/File_Transfer_Protocol/,» [En línea].

[12] E. t. f. Windows, «https://mobaxterm.mobatek.net/,» [En línea].

[13] X. Monitor, «http://xymon.sourceforge.net/,» [En línea].

[14] w. H. Q. Center, «https://es.wikipedia.org/wiki/HP_Quality_Center,» [En línea].

[15] B. o. Sistemas, «manual_mantis.doc,» 2015.

[16] Sopra, *Documentación Interna Proyecto*, Barcelona, 2017.

[17] f. Spring, «https://spring.io/.,» [En línea].

[18] Dave Syer, Wayne Lund, Lucas Ward, «https://docs.spring.io/spring-batch/1.1.x/spring-batch-docs/reference/html-single/,» Spring Batch - Reference Documentation. [En línea].

[19] A. Tomcat, «URL: http://tomcat.apache.org/.,» [En línea].

[20] «https://www.qlik.com/us/products/qlikview,» [En línea].

[21] Hibernate, «https://docs.jboss.org/hibernate/orm/3.5/reference/en/html/,» [En línea].

[22] Atomikos, «https://www.atomikos.com/Main/ProductsOverview,» [En línea].

[23] Log4j, «https://logging.apache.org/log4j/2.x/,» [En línea].

[24] IText, «https://itextpdf.com/,» [En línea].

[25] JasperReports, «http://www.abartiateam.com/jasperreports,» [En línea].

[26] w. Junit, «https://es.wikipedia.org/wiki/JUnit,» [En línea].

[27] A. Tiles, «https://tiles.apache.org/framework/index.html,» [En línea].

[28] i. AJAX, «https://www.w3schools.com/js/js_ajax_intro.asp,» [En línea].

[29] Tasklet, «https://docs.spring.io/spring-batch/trunk/apidocs/org/springframework/batch/core/step/tasklet/TaskletStep.html,» [En línea].

[30] W. g. EDT, «https://planhammer.io/project/,» [En línea].

[31] M. y. convenciones, «https://sonarqubehispano.org/pages/,» [En línea].

[32] w. Cron, «https://es.wikipedia.org/wiki/Cron_(Unix),» [En línea].

GLOSARIO

ARS: Agencia régional de sanidad

HAPI: Aromización e Intercambio de Información

FTP: Protcolo de Transferencia de Archivos

J2EE: Java Plataform, Entreprise Edition

MCD: Modelo conceptual de Datos

XML: eXtensible Markup Language

CSS: Cascading Stylesheets

HTML: HyperText Markup Language

SQL: Structured Query Language