Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Informatización de una PYME aplicando una metodología ágil: Un caso real.

Trabajo Fin de Grado
Grado en Ingeniería Informática

Autor: Taberner Aguas, Baldo
Tutor: Albert Albiol, Manuela
2014 / 2015

Resumen

Demasiadas PYMES españolas siguen una agravante trayectoria de desactualización tecnológica que deben subsanar para conseguir una mayor competitividad. Desgraciadamente, las PYMES españolas no pueden competir contra las grandes empresas, la falta de personal cualificado en plantilla y la falta de músculo financiero colapsa la capacidad de inversión.

En este contexto, el proyecto pretende desarrollar un sistema informático para gestionar y abordar las necesidades básicas de una PYME en concreto a partir de nuevas tecnologías y metodologías ágiles. Estas nuevas tecnologías y metodologías permiten cuantiosos beneficios para las PYMES.

El sistema a desarrollar es para una PYME dedicada a la producción y distribución de abanicos y otros complementos de moda y se quiere implementar un sistema capaz de gestionar los productos, los clientes y los pedidos de una forma fácil y eficiente, de la misma forma se quiere mostrar una imagen más corporativa de cara al cliente y al público en general.

Palabras clave: PYME, sistema, desarrollo, metodologías ágiles, tecnologías.

Abstract

Too many Spanish SMEs follow an aggravating path of technological obsolescence that must overcome to achieve greater competitiveness. Unfortunately, the most Spanish SMEs can not compete against big business, lack of qualified personnel and lack of financial muscle collapses investment capacity.

In this context, the project aims to develop a system to manage and address the basic needs of a particular SME. From new technologies and agile methodologies. These new technologies and methodologies allow substantial benefits for SMEs.

The system to be developed is for an SME engaged in the production and distribution of fans and other fashion accessories and want to implement a system to manage products, customers and orders in an easy and efficient way, in the same way wants to show a great corporate image for the customer and the public in general.

Keywords : SME, system, developing, agile methodology, new technologies.

Índice

Índice de figuras e imágenes

Figura 53: Scripts para DB

Figura 54: Respuesta de expertos en MongoDB
Fuente: stackoverflow.com/questions/27857494/the-best-solution-architecture-in-mongodb

Capítulo 1. Introducción

1.1 Motivación

Con el crecimiento exponencial que está cosechando Internet desde hace ya décadas, muchas PYMES españolas siguen una agravante trayectoria de desactualización tecnológica que deben subsanar para conseguir una mayor competitividad tanto a nivel nacional como internacional, abrirse a nuevos mercados y consolidarse en un mundo cada vez más globalizado.

Las PYMES españolas no pueden competir contra las grandes empresas, la falta de personal cualificado en plantilla y la falta de músculo financiero, colapsa la capacidad de inversión al ritmo del avance de las nuevas tecnologías, que junto con el débil conocimiento de su repercusión genera un ensanchamiento entre las compañías punteras con más recursos y las otras.

En solución al problema, existen multitud de empresas de desarrollo de software que proveen estas PYMES vendiendo moldes idénticos de sus productos y ahorrando costes, pero degradando la calidad corporativa de las mismas al no poder disfrutar de un producto que se ajuste a sus necesidades reales y pudiendo diferenciarse del resto, es decir, un producto hecho a medida.

1.2 Objetivos

El objetivo principal del proyecto es construir un ecosistema informatizado en forma de aplicación web que permita la gestión de una PYME en concreto. El producto debe estar hecho a medida y conseguir un consumo de recursos lo más ajustado posible a las necesidades reales de una pequeña empresa. Aunque el sistema pretenda ajustar el consumo de requisitos, éste también debería ajustarse a la más que probable escalabilidad que pueda sufrir en capacidad de productos como de usuarios y permitir un mantenimiento lo más cómodo posible para la administración. Para conseguirlo se dará uso de una metodología ágil, que ha sido amoldada al proyecto a partir de hibridar otras metodologías ágiles estandarizadas. Esto permite construir un producto totalmente a medida que cumpla con las necesidades del cliente y por otro lado permite reducir recursos innecesarios y que pueden llegar a ser contraproducentes.

La estructura básica del modelo de negocio de la empresa se basa en tres puntos clave: producto, cliente y pedido. A partir de los tres principales núcleos de negocio se quiere lograr una buena gestión de cada uno de ellos, así como que queden relacionados. A su vez, el cliente debería ser capaz intuitivamente de lograr aquello que busca y facilitar el contacto y la automatización de realizar pedidos y eliminar trabajo burocrático a la empresa, por lo que éste recibiría la mayor información posible y autonomía suficiente para funcionar sin necesidad de llamadas telefónicas, visitas o correos electrónicos.

También se pretende trabajar aplicando tecnologías modernas capaces de adaptarse al futuro próximo y que tengan a la vez, buena aceptación de la comunidad, así como documentación y demás utilidades.

1.3 Organización de la memoria

Capítulo 2
La memoria sigue el ciclo básico de desarrollo de software, comienza con los principios acordados de las metodologías ágiles y se definen algunos aspectos que se han considerado importantes de algunas de ellas para formar la metodología que se ha dado uso en este proyecto, que se define también, y muestra una vista rápida del seguimiento del proyecto.

Capítulo 3
En este capítulo se determinan los requerimientos del sistema. Se define la elicitación que se ha obtenido a partir de la información que ha proporcionado el cliente, mostrándose en tablas y diagramas aclarativos. Seguidamente se crea la lista de requisitos funcionales, no funcionales y de restricción.

Capítulo 4
El capítulo 4 está dedicado al diseño que se ha fijado como resolución al proyecto. Se describen las diferentes arquitecturas que se han establecido, tanto a nivel de desarrollo, como estructura de datos y la arquitectura del entorno donde trabaja el sistema.

Capítulo 5
Las distintas tecnologías que se han utilizado para la elaboración y gestión del proyecto quedan descritas en el capítulo 5 con ejemplos propios del proyecto. Siguiendo un orden en el índice según su naturaleza.

Capítulo 6
En el capitúlo 6 se describen las técnicas que se han usado para testear el proyecto en sus diferentes niveles de desarrollo. A partir de pruebas de aceptación, sistema, unitarias, integración y rendimiento, que queda definidas y ejemplarizadas.

Capítulo 7
En el capítulo 7 se comentan algunos de los problemas que han ido surgiendo conforme avanzaba el desarrollo y qué respuesta se les ha dado. Los problemas han sido en la planificación, cambios radicales, reestructuración y técnicas de desarrollo.

Capítulo 8
Finalmente el capítulo 8 da cavidad a las conclusiones finales del proyecto.

Capítulo 2. Metodología

Las metodologías ágiles se han convertido en los últimos años en los principales métodos a seguir por la gran mayoría de empresas de desarrollo de software. En 2001, diecisiete prestigiosos ingenieros, convocados por Kent Beck, crearon un manifiesto que dictaba doce principios básicos sobre las metodologías ágiles.

1. Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.

2. Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Ágiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.

3. Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.

4. Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.

5. Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.

6. El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.

7. El software funcionando es la medida principal de progreso.

8. Los procesos Ágiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.

9. La atención continua a la excelencia técnica y al buen diseño mejora la Agilidad.

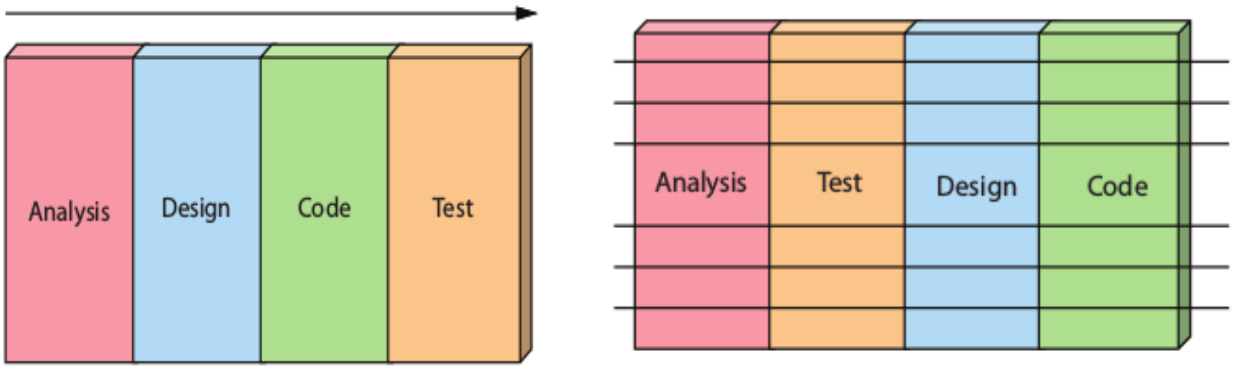
10. La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.

11. Las mejores arquitecturas, requisitos y diseños emergen de equipos auto-organizados.

12. A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.

A partir de estos principios, se definen diferentes metodologías reconocidas como ágiles basadas en el desarrollo iterativo e incremental, donde los requisitos y soluciones evolucionan mediante la colaboración de grupos auto-organizados y multidisciplinarios. La forma de dirigir el desarrollo del proyecto cambia en comparación a la forma tradicional, las etapas están fraccionadas en unidades de requisitos que avanzan en diferente tiempo, por lo que permite mayor flexibilidad a la hora de seguir el proceso de ingeniería.

Figura 1



Comparación del método tradicional respecto al método ágil

Entre esta familia de metodologías destacan Scrum, Kanban, o eXtreme Programming. Cada una con sus peculiaridades, sus pros y sus contras. Aunque lo cierto es que como dijo Corey Ladas: "*El proceso de planificación del trabajo ideal debería siempre ser proporcionado por el equipo de desarrollo con el mejor propósito para trabajar en lo siguiente, ni más ni menos.*". Por esa misma razón seguir una metodología dependerá del equipo y después del proyecto, pudiendo hibridar las metodologías para el ajuste correcto. En este proyecto se ha tomado como base la metodología Scrumban y se han adoptado otras características de otras metodologías ágiles, hasta conseguir la metodología de trabajo.

2.1 Scrumban como base

Scrum-ban unifica las bondades de Scrum con Kanban de manera que aprovecha la visualización del proceso por etapas y asignación del equipo en una pizarra blanca con sticks que representan las tareas a realizar y que son seleccionadas por el personal apropiado, hasta llegar a la etapa final. Además proporciona un límite de tareas en las que se esté trabajando, tanto en equipo como de manera personal. Así pues se crea la política de que un miembro del equipo puede hacer recursivamente como mucho dos tareas al mismo tiempo, esto es debido a que en algún momento por algún tipo de bloqueo o circunstancia debamos de flexibilizar la aceptación de otra tarea. Dependiendo del equipo de desarrollo se acota también el número de tareas en proceso a nivel global, para mantener el desarrollo del trabajo de manera constante y ordenado.

Conforme van apareciendo nuevos requisitos o nuevas tareas para solucionar fallos y demás que se han reportado en utilidades tipo Issue/Bug Tracker se recopilan en un contenedor de tareas llamado *Backlog*. Para gestionar todo el cúmulo de trabajo, se divide en *sprints* que son subcontenedores que comprenden tareas a realizar entre 1-2 semanas, ordenando las tareas por prioridad y por reajuste tanto en complejidad como en coste temporal. Al final del *sprint* se presentan las novedades a los responsables del negocio. Todo esto forma un ciclo que permite mantener la vida del software lo más saludable posible manteniendo funcionalidad desde el primer momento y aportando otras nuevas características al sistema.

2.2 Otras características adoptadas

Scrumban permite la gestión eficiente del proceso de desarrollo, pero además acepta nuevas características de otras metodologías que puede ser muy interesantes y se complementan perfectamente, como es el caso de *eXtreme Programming* (de ahora en adelante XP).

XP proporciona características más concretas y técnicas en el desarrollo de software como es la adopción de pruebas unitarias continuas (TDD, *Test driven development*) repetidas y automatizadas, incluyendo pruebas de regresión. Esto permite un sistema más consistente y con gran aceptación a nuevo código. El TDD consiste en crear el test antes que el código de manera que no se testea un código ya elaborado, sino que se testea el comportamiento que luego deberá realizar el código.

La técnica de *refactoring* también viene definida en XP y consiste en el seguimiento de una serie de normas para realizar código de calidad sin cambiar ningún aspecto de funcionalidad externa en cada método o clase en el que se realiza el *refactoring*. La eficiencia a largo plazo es tangible y permite una mayor aceptación por otros desarrolladores que pueden ser nuevos miembros del equipo de trabajo.

Gracias a los sistemas de control de código fuente (SCM) se puede desarrollar fácilmente en equipo, trabajando incluso la misma porción de código en paralelo, esto es posible gracias al registro histórico que se va almacenando dentro de un repositorio común. Si el repositorio está on-line, se permite el trabajo remoto y a la vez siempre controlado. El uso de este tipo de herramienta se cita como muy recomendable por Martin Fowler dentro de la aplicación de metodologías ágiles en integración continua.

2.3 Metodología de trabajo

Teniendo en cuenta todo lo anterior, la metodología a seguir en este proyecto es una base que tiene como mezcla las técnicas de Scrumban junto a otras técnicas de otras metodologías teniendo como objetivo el control y la flexibilidad de abordar el proyecto atendiendo también a razones externas y permitiendo escalabilidad si se requiriese.

Como el equipo de trabajo está formado por una sola persona, se agiliza y se flexibiliza aún más en su procedimiento. Las reuniones son directas con el cliente y no existen roles ni jerarquías en el equipo de desarrollo.

2.3.1 Los Sprints

Scrum entre otras cosas es útil para la planificación del proyecto por objetivos (Sprints). Los Sprints tienen una duración entorno a los 15 días aproximadamente, pero son variables y dependientes de factores externos. Esto significa que la planificación del Sprint rara vez se cumplirá en tiempo, pero sirve para gestionar y tener un plan al que intentar ceñirse.

En este proyecto se ha seguido una organización por Sprints en base a la prioridad y estabilidad en carga de trabajo por cada uno de ellos. El resumen de los Sprints con sus tareas de mayor importancia que se han ido siguiendo se relatan a continuación.

Sprint 1

El primer grupo de tareas ha consistido sobretodo en agrupar y ordenar a *grosso modo* los requerimientos que solicitaba el cliente. A partir de reuniones se ha seguido un procedimiento de elicitación y en base a ello se ha generado la especificación de requisitos del capítulo 3.

Sprint 2

Una vez llegado al entendimiento y acercamiento de ideas entre la parte técnica y el cliente se ha modelado una vista previa de la arquitectura que debería seguir el sistema. Así como las tecnologías que se identificaban como fundamentales para el mejor resultado posible en el desarrollo. En el capítulo 4 se muestra la arquitectura que se está utilizando en producción, pero en el Sprint 2 la estructura era más sencilla porque trabajaba sobre un framework de mapeo (Morphia) y modelos llenos de anotaciones que facilitaban la puesta a punto, de una manera rápida y efectiva.

Sprint 3

En el Sprint 3 comienza el desarrollo de los CRUDL's según las estructuras de datos que se habían diseñado para las entidades: Products, Collections, Colors, Materials. Según los primeros análisis establecidos, seguían un código bastante semejante, excepto Products que tenía mayor complejidad y creaba dependencia con el resto. De hecho el controlador de Products era el encargado de controlar a los demás controladores y estos a sus respectivas clases CRUDL. Desde este punto todas las entidades siguen el patrón Model-View-Controller explicado en la sección 4.1.2.

Sprint 4

Con una estructura funcional se empieza gestionar la interfaz gráfica del backend, apoyandose en Bootstrap definido en el capítulo de tecnologías. Para la gestión por interfaz de los controllers que son las únicas clases que poseen en la estructura métodos públicos se crea una clase contenedora de objetos controlador llamada Global. Esta clase se encargaba de proporcionar una comunicación conjunta entre la GUI y los controladores.

Sprint 5

Se corrigen errores de los Sprints anteriores, se mejora la usabilidad y comienza a tener forma la interfaz gráfica de la parte Frontend, también gracias a Bootstrap. Aquí el framework tendrá más peso que en la parte Backend, porque se va a cuidar en detalle los acabados al igual que un correcto código HTML. También se clona y se limita la parte estructural de Backend en Frontend y se implementa la funcionalidad de enviar e-mails, para que los clientes puedan comunicarse con la empresa.

Sprint 6

La gestión de los productos crece más en control y funcionalidad, se afinan más las propiedades y se implementan atributos como las imágenes de los mismos que servirán de muestra en Frontend aunque se gestionen desde Backend. Para ello se debe desarrollar la gestión y configuración mediante FTP, tanto en el servidor como en la aplicación Backend. Se ha usado FTPClient, definido en el capítulo 5. Se ha mejorado la gestión en Morphia porque entraban factores hasta ahora no vistos como la gestión de subdocumentos anidados con matrices de documentos en la base de datos.

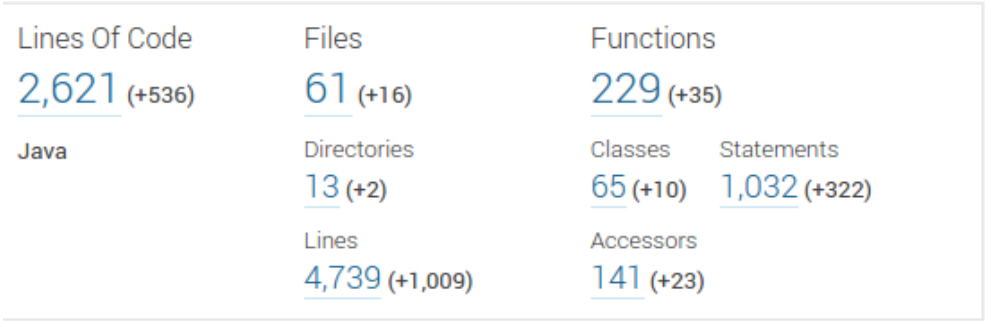
Sprint 7

El Sprint 7 ha sido utilizado para mejorar todo lo hasta aquí creado, tanto en el aspecto gráfico, como en la eficiencia del código en ambas aplicaciones web. Este Sprint ha sido muy importante para aplicar técnicas de refactorización, hacer el código más legible y reestructurar algunos métodos y clases. Además se han establecido algunos cambios de requerimientos dictados por el cliente, porque por ejemplo conjuntar los productos en colecciones era una idea primaria que ha sido descartada.

Sprint 8

Una vez las aplicaciones son más estables se comienza a crear la estructura de datos para clientes, y todas las clases pertinentes para su gestión en Backend con controlador capacitado para trabajar con CRUDL y en la parte del Frontend se ha implementado también el método de Login, con todo lo que esto conlleva en variables de sesión y demás opciones que incumben tener un rol u otro en la parte Frontend. En la Figura 2 se puede apreciar los cambios que han tenido lugar tras aplicar este Sprint

Figura 2



Cambios en Sprint 8

Sprint 9

En este Sprint el sistema ha sufrido una reestructuración. Tras haber tenido un estudio sobre las necesidades reales del sistema y como funcionan las bases de datos no relacionales, tener documentos para colores y materiales era deficiente. Como tampoco servía embeber los atributos como subdocumentos del documento Producto, porque creaba inconcordancia de datos se ha reestructurado como información interna del sistema. Es decir dentro de Java. Esto se explica en el apartado 4.2.3 de esta memoria. Como la base de datos está ya en producción, se necesita reestructurar no solo en programación sino en datos, por lo que se genera una aplicación que en base a scripts, reorganiza los datos.

Sprint 10

En el Spring 10 las tareas se han centrado exclusivamente en la parte del Frontend, mejorando la experiencia de navegación y ofreciendo al cliente registrado la posibilidad de trabajar con un carrito de la compra para poder hacer pedidos. Para ello se ha trabajado con la base de datos especializada en frontend en el apartado 4.2.2. Tras varios vayvenes se llega a un acuerdo con el cliente que permite gestionar pedidos facilmente sin perder usabilidad al usuario.

Sprint 11

Ya teniendo las aplicaciones web funcionando en este sprint se ha evolucionado para dejar de lado Collections e implementar Types y Subtypes. La agrupación de los productos tiende a gestionarse de una forma relacional y la estructura varía conforme se va creando, intentando amoldarse a la petición del Frontend sin perder la gestión en Backend. Además en Backend se integra la parte de Orders con Carts que es los mismos datos Carts con los que trabaja Frontend mediante la base de datos web-at.

Sprint 12
Tras una perfecta eficiencia lograda en el Sprint anterior en respecto a la petición del Frontend, el Backend posee una complicación desmesurada para el control y actualización de los productos según sus tipos. Se implementa otra solución para Types. En la sección 4.2.1 y en 7.4 se muestra cual era el problema y como se ha resuelto finalmente. Además se corrigen algunos errores que habían aparecido con Carts tanto en Frontend como en Backend.

Sprint 13
Se aplican mejoras gráficas, se refactoriza el código exhaustivamente de nuevo y se mejora el control de los usuarios. Por ejemplo en el registro se tiene en cuenta la validez del DNI para una persona física de nacionalidad española. Se reestructura la arquitectura en Frontend y hay entidades donde decrece la dependencia en Morphia y se construye de forma modularizada según la sección 4.1.1. Se plantea la idea de mapear o no mapear, problema explicado en el apartado 7.5.

Sprint 14
En este Sprint se comienza a investigar e implementar una solución que permita interactuar con clientes de diferente familia (PC, Smartphone, SmartTV,...) Para ello se implementa una API paralela que no está todavía en producción que permite la comunicación mediante RESTful para la obtención de datos y procesos a partir de un frontend con credenciales. Además se desarrolla en la parte Backend, siguiendo la tecnología PDFbox en el apartado 5.2.6, un grupo de clases que permite la creación de un catálogo de todos los productos que están catalogados como visibles para el cliente. De esta forma se externaliza en otro formato toda la información que es visible en Frontend para comerciales o mayoristas.

Figura 3

Cambios en Sprint 14

Sprint 15

Conforme ha ido avanzando el proyecto se han aplicado continuas mejoras gráficas y estructurales, se han implantado técnicas SEO para una mejor indexación en buscadores. Se han seguido cambiando estructuras de datos gracias al sistema de Scripts desarrollado en el Sprint 9. Se integra el uso de otras tecnologías de gestión como YouTrack explicado en 5.6.3. Se ha migrado todo el Backend eliminando la dependencia en Morphia, pues tenía mucho más trabajo que la migración Frontend.

Backlog

En el Backlog queda todavía mucho por hacer. Se quiere llevar a producción la API desarrollada en el Sprint 14, se quiere mejorar la experiencia de usuario en el Frontend, crear un motor de búsqueda eficiente que permita filtrado de características, permitir la visualización de la web multilenguaje. Control interno de estadísticas a nivel Backend y Frontend, con generación de gráficos, todo estos almacenados en una base de datos que más tarde permita una gestión inteligente de los mismos para los clientes. Aumentar la eficiencia y la recolección de listas utilizando programación funcional ya habilitada en las nuevas versiones Java. Mejorar la seguridad del Frontend. Implementar un sistema de facturación en Backend que trabaje con pedidos provenientes de la web o de otros puntos. Desarrollar un subsistema para el control de costes de producción.

2.3.2 Tareas

Conforme se empieza un Sprint se crean tantos post-its como tareas deben realizarse en una pizarra del estilo Kanban formada por cinco carriles: To do, Analysis, In process, Testing y Done. Al principio se utilizó una aplicación informática para gestionar la pizarra, pero se terminó pasando al formato físico para que el cliente pudiera seguir fácilmente las etapas por las que pasaba cada tarea. En la mayoría del proyecto se ha seguido el trabajo conforme se presenta en la **Figura 4**.

Figura 4

Pizarra Kanban

Dichas tareas estarían colocadas como post-it's dentro de la tabla Kanban en el carril “To-Do”. El proyecto diferencia varios tipos de tareas: documentación, arquitectura, producto, cliente, gestión de pedido, etc. Cada tipo está representado con un símbolo distintivo. De esta forma es fácil identificar su tipología y si el desarrollo del proyecto se está centrando más en una parte del sistema o en otra. Conforme se avance en el trascurso del proyecto se van pasando al carril “Analysis” donde tienen un análisis más exhaustivo de sus necesidades y se crean diagramas para lograr dicha tarea. Algunas se fraccionan en subtareas que vuelven a “To Do”. En caso de dudas se interactúa con el equipo o el mismo cliente, y cuando ya se tiene claro el *modus-operandi* a seguir, la tarea cae en el carril de “Process”, en este carril no deberían de haber más de dos o tres tareas consecutivamente, esta norma es debida para que el sistema no tenga demasiados frentes abiertos incompletos sin funcionalidad. Si la tarea parece finalizada pasa al carril de “Testing” donde se realizarán las pruebas pertinentes, aún habiendo pasado los test unitarios previamente en el caso de haber aplicado Test Driven Development. Dependiendo de la tarea, necesitará la supervisión del cliente. Pasado el testeo la tarea termina su ciclo en el carril “Done” donde se agruparán hasta dar finalizado el Sprint y poder así presentar las nuevas funcionalidades al cliente en su totalidad y formar el despliegue.

2.3.3 Políticas

Aunque parezca que Scrumban es la metodología que manda, realmente se encarga de la gestión en el medio-alto nivel. Scrumban permite seguir la situación del proyecto sin un conocimiento extenso del mismo. Es la ordenación de qué se quiere hacer y cuándo se quiere hacer, además de focalizar las etapas en las que está cada tarea. Aún así en la puesta en práctica aparecen diferentes objeciones y variables según el procedimiento y la naturaleza de cada tarea, de manera que se deben de crear algunas políticas a bajo nivel, cuando la tarea está en desarrollo.

- La técnica de TDD será aplicada en métodos y clases no dependientes de los datos de la base de datos, por lo que habrán métodos que no deberán de testearse con anterioridad, sólo en el carril de “Testing”.

- Si la tarea es un fallo del sistema, pasará con máxima prioridad hasta terminar en “Done”, antes que ninguna otra, y si el fallo es grave, se desplegará en el momento.

- De forma periódica (semanalmente), se deben realizar tareas de refactoring, análisis exhaustivos con aplicaciones desarrolladas para ello, como por ejemplo SonarQube, e informes sobre la evolución que va teniendo tanto sistema como código del mismo.

- Conforme se finalizan tareas se integra el código del proyecto dentro de un controlador de versiones para en caso de problemas poder retornar a un punto común y funcional, esto también ayudará como Backup del código fuente.

Capítulo 3. Especificación de requisitos

En esta sección se presenta una descripción a alto nivel del sistema. Se presentaran las principales áreas de negocio a las cuales el sistema debe dar soporte, las funciones que el sistema debe realizar, la información utilizada, las restricciones y otros factores que afecten al desarrollo del mismo.

3.1 Elicitación

Elicitación es el proceso de adquirir todo el conocimiento relevante necesario para producir un modelo de los requerimientos del dominio de un problema, para ello se han mantenido reuniones con el cliente que responden a cada uno de los puntos detallados.

3.1.1 Fuente de requisitos

- CEO de la empresa
- Consejero delegado de la empresa
- Ficha de cliente
- Factura, Pedido, Albarán
- Otros documentos

3.1.2 Stakeholders

Los diferentes interesados en el buen funcionamiento del sistema son los clientes y los administradores que se encargan de su gestión, por otra parte e indirectamente la dirección de la empresa estará satisfecha con la aplicación si los dos anteriores lo están y les es cómoda y eficiente a la hora de trabajar y ser atendidos.

Figura 5

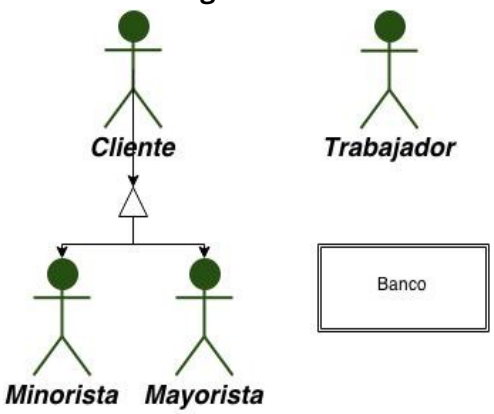
Nombre	Rol	Usuario Directo	Intereses
CEO	Preside la empresa	No	Aumentar la eficiencia de la empresa.
Administrador	Administra el sistema	Sí	Facilidad de uso, eficiencia y ahorro de tiempo.
Cliente	Comprador	Sí	Mucha facilidad de uso y seguro.

Stakeholders

3.1.3 Actores

Existe dos tipos de usuarios en el sistema, por un lado está el usuario trabajador que es el que administra la gestión de la información interna de la empresa y por otro lado se encuentran el usuario cliente que se divide en dos categorías, como minorista, o mayorista, porque funcionalmente pueden llegar a tener distinto trato. La decisión de otorgar a un cliente de una forma o de otra depende del usuario trabajador. Además existe un agente externo que es el Banco que es el capacitado en llevar cobros cuando los pagos sean vía bancaria.

Figura 6

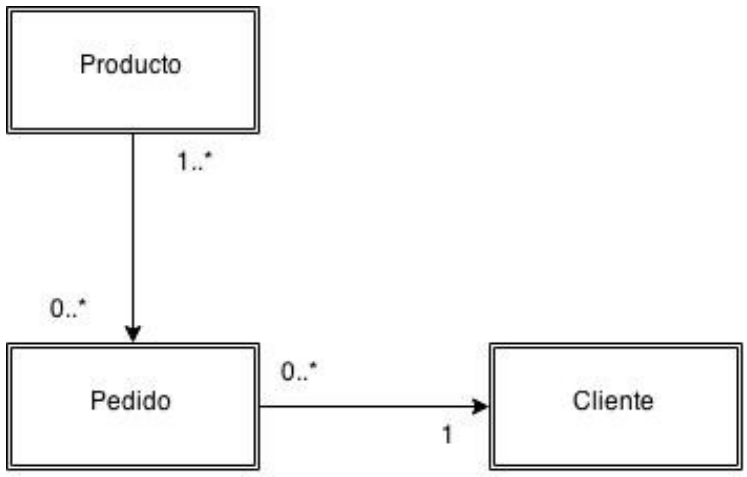


Actores

3.1.4 Modelo de dominio

El modelo de dominio en el que a *grosso* modo se basa el *software* contiene tres identidades que se comunican entre sí. Producto, pedido y cliente.

Figura 7



Modelo de dominio

3.1.5 Diagrama de contexto

A partir de la información obtenida se crea un diagrama de contexto un tanto más técnico, pero a la vez superficial y fácil de entender, que pretende representar la comunicación de cada uno de los actores que interactuan con el sistema. Queda patente que el cliente choca a través de una parte de la aplicación diferente al que interactua el trabajador, pero los dos usan las mismas fuentes de datos, con sus respectivas reestricciones a la hora de acceder a ellos.

Figura 8

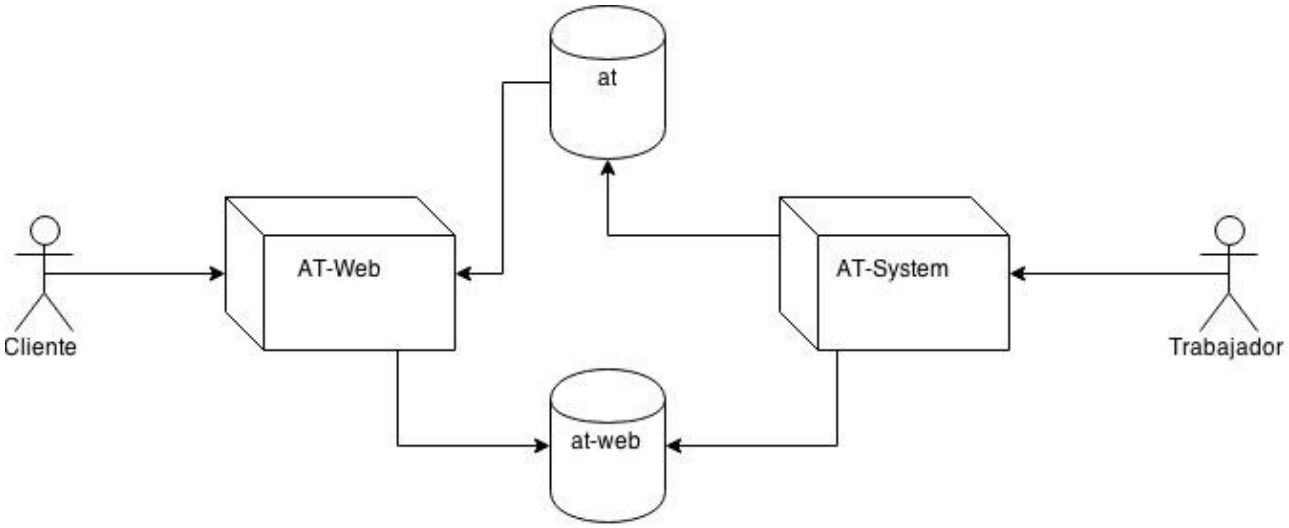


Diagrama de contexto

3.1.6 Funciones del sistema

En términos generales, el sistema deberá proporcionar soporte a las siguientes tareas de gestión de productos y clientes de la empresa:

- Mantenimiento y gestión de los clientes.
- Mantenimiento y gestión de los productos.
- Catálogo de muestra para los clientes.
- Creación de pedidos.
- Mantenimiento y gestión de los pedidos.

Mantenimiento y gestión de los clientes

Los clientes se registran atendiendo a sus datos personales básicos, así como también los de contacto y facturación. Será obligatorio su número de identidad fiscal sea DNI, CIF, IVAN. Pudiendo siempre ser editados, incluso eliminados.

Mantenimiento y gestión de productos

Los productos se registran atendiendo a sus atributos más naturales como el año de primera producción, referencia y precios. También según sus especificaciones de producto se puede definir la medida, los colores u otras cualidades del mismo. De la misma manera que los usuarios, pueden ser editados, incluso eliminados. Además el producto debería respaldarse como mínimo con una imagen de este.

Catálogo de muestra para los clientes

Los clientes que accedan al sistema tienen acceso a los productos que sean publicados como visibles, pudiendo ser ordenados por diferentes parámetros y así facilitar la búsqueda de catálogo. El cliente tendrá acceso a una gran parte de información del producto.

Creación de pedidos

Los clientes pueden hacer pedidos en base al catálogo, pudiendo seleccionar por referencia, cantidad y colores. Creando un extracto con los precios unitarios, conjuntos y globales más impuestos que forman el pedido.

Mantenimiento y gestión de los pedidos
La empresa recibirá los pedidos de los clientes que serán validados manualmente y procesados en la gestión interna del sistema hasta su correspondiente servicio y facturación.

3.1.7 Características de los usuarios

Los usuarios de este sistema están familiarizados con el uso de aplicaciones de este tipo ya que simula una tienda online, que todos conocemos. Es por ello que el sistema ha de ser gráfico. Con una interfaz sencilla e intuitiva, que no exija gran tiempo para su aprendizaje, a la vez que lo suficientemente potente para que el usuario pueda hacer aquello que necesita.

3.1.8 Restricciones

El sistema será accedido desde cualquier dispositivo que contenga un navegador web. Dicho sistema contempla dos partes una visible para el cliente y otra solamente visible para la administración de la empresa, la cual sólo será accedida dentro de la red local de la empresa. Se le exige al sistema disponibilidad online las 24 horas de los 7 días de la semana en los 365 días que tiene el año. Además deberá tener una pronta respuesta.

3.1.9 Suposiciones y dependencias

Para lograr la estabilidad del sistema se necesita de un servidor externo con buena conexión y capacitado para soportar servicios de aplicaciones web, de base de datos y transferencia de archivos. Se supone que la empresa ya está provista de un servidor de correo electrónico con el que comunicarse. A la hora de cobrar el pedido a un cliente automáticamente se necesitará el VPN de un banco especializado en cobro online.

3.2 Requisitos

En este apartado se presentan los requisitos que deberán ser satisfechos por el sistema. Todos los requisitos aquí expuestos son esenciales, es decir, no sería aceptable un sistema que no satisfaga alguno de los requisitos expuestos.

3.2.1 Requisitos funcionales

Mantenimiento y gestión de los clientes

RF001. Hay dos tipos de clientes; cliente mayorista con el que se trabaja de manera habitual y el cliente minorista que es desconocido.

RF002. El cliente mayorista tiene una cuenta que se le habrá dado de alta previamente por la administración.

RF003. La empresa debería obtener, sino los tiene ya, todos los datos necesarios del cliente mayorista para tramitar pedidos, así como datos personales / empresariales para abrirle la cuenta.

RF004. El cliente minorista no necesita registrarse para hacer un pedido. Cuanto más fácil mejor.

RF005. El cliente minorista ve el precio correspondiente al mercado minorista.

RF006. El cliente mayorista ve el precio correspondiente al mercado mayorista.

RF007. La administración puede editar y eliminar los datos del cliente.

RF008. El sistema comprueba que los datos son correctos. (p.e. DNI válido).

RF009. El sistema lista los clientes y es fácil acceder a ellos para su gestión.

Mantenimiento y gestión de productos

RF010. La administración puede dar de alta, editar y eliminar cada producto.

RF011. Cada producto tiene una referencia única.

RF012. Cada producto tiene dos precios, uno para el mercado minorista y otro para el mayorista.

RF013. Cada producto es de una familia de productos, y sólo de una.

RF014. La administración puede fácilmente subir imágenes al sistema.

RF015. Cada producto tiene una imagen principal seleccionada por la administración.

RF016. Cada familia de productos tiene en común un tipo de especificaciones.

Catálogo de muestra para los clientes

RF017. Cada producto tiene un campo que se habilita como público para el catálogo.

RF018. El cliente puede listar todos los productos.

RF019. El cliente puede filtrar productos por parámetros de los mismos, como por ejemplo, por precio, medidas, colores y otros atributos.

RF020. En el listado de los productos aparecerá su referencia y su imagen principal en tamaño reducido.

RF021. El catálogo ordena los productos por familia de productos.

RF022. El catálogo tiene una sección de novedades con los últimos productos.

Creación de pedidos

RF023. El cliente puede seleccionar la compra de un producto desde su ficha.

RF024. El cliente ingresa la cantidad y colores que desea adquirir.

RF025. El cliente rellena un breve formulario según su tipología. El minorista paga en el momento y el mayorista no.

RF026. El cliente recibe un correo de confirmación como se ha recibido correctamente.

RF027. La administración recibe por correo electrónico el pedido, al igual que se recibe en el sistema.

Mantenimiento y gestión de los pedidos

RF028. La administración debe validar los pedidos de los clientes

RF029. La administración gestiona los pedidos según estados del mismo. Validado, En Proceso, Enviado, Cobrado.

3.2.2 Requisitos no funcionales

RNF001. El sistema deberá estar disponible on-line 24 horas los 7 días de cada semana.

RNF002. El sistema responderá tan rápido como le sea posible.

RNF003. El sistema debe ser seguro ante ataques y mantener la privacidad de los datos.

RNF004. Se facilitará la indexación de buscadores para mejor posicionamiento.

RNF005. El sistema debe ser escalable y estable.

3.2.3 Requisitos de restricción

RR001. Se dispone de un servidor hospedado por terceros con buen ancho de banda.

RR002. Se dispone de una intranet de ordenadores offline

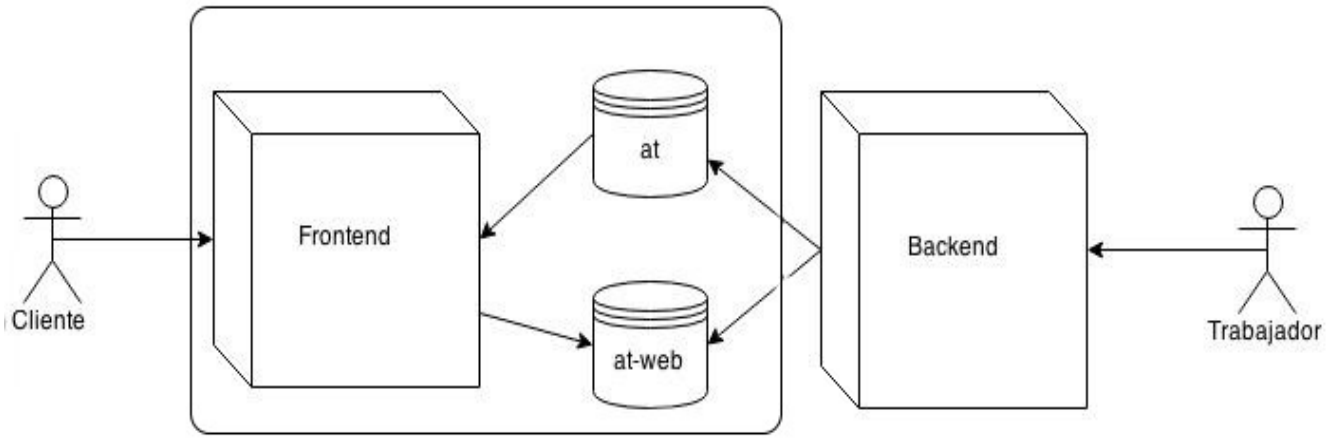
Capítulo 4. Diseño

El diseño del sistema es una de los sprints más importantes del proyecto, define la estructura del sistema, tanto en la parte de desarrollo, en la estructura de los datos o el entorno en el que se ejecuta el proyecto.

4.1 Arquitectura

Conforme a los requisitos obtenidos en el apartado anterior y tras una investigación exhaustiva se esquematiza una estructura general como en la Figura 9.

Figura 9



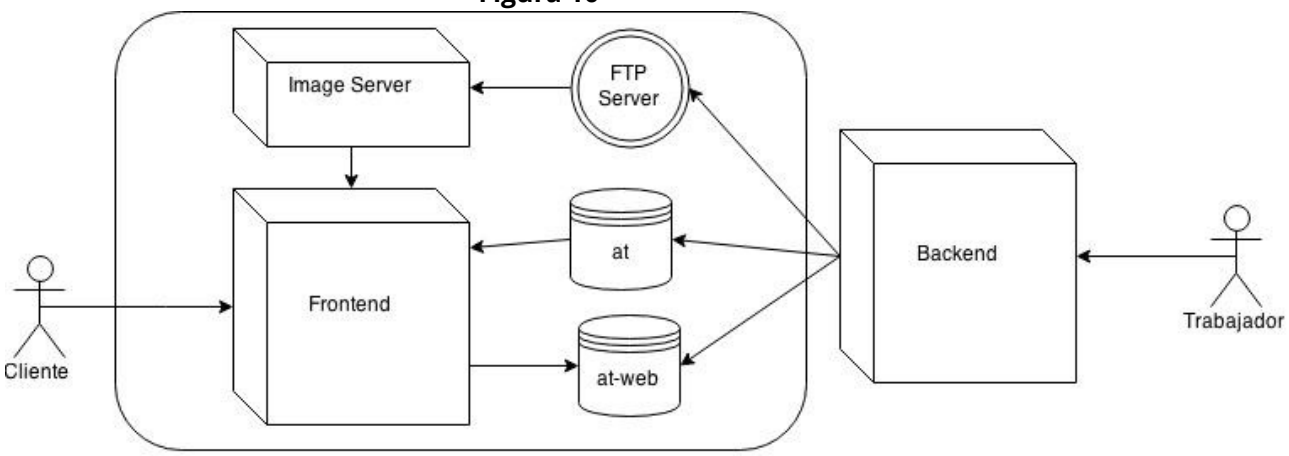
Arquitectura general

La estructura principal se organiza a partir del modelo de aplicación distribuida siguiendo una arquitectura Cliente / Servidor, permitiendo la centralización de control de los accesos, recursos y la integridad de los datos, que son controlados por el servidor de forma que un programa cliente defectuoso o no autorizado no pueda dañar el sistema.

Aunque existan dos tipos de clientes, en este esquema su funcionalidad es la misma. Como se observa, el cliente sólo interactúa contra una sola aplicación, la de frontend, vía web. Esta capa conecta con dos bases de datos, una, 'at', donde sólo tiene permisos de lectura que es objetivo del almacenamiento de toda la información de la empresa, tanto pública como privada. Por otra parte 'at-web' está creada para la información que necesite crear el Frontend. Esta parte del sistema estaría siempre a disposición on-line, sin embargo la aplicación de Backend estaría off-line en la intranet

empresarial sólo a disposición de los trabajadores de la empresa. Aportando de esta manera mucha más seguridad en la gestión de la información. Así que la única forma en la que se puede actualizar los datos importantes de la empresa es a través de la intranet.
Se necesita gestionar las imágenes de los productos, con motivo de distribuir el sistema según partes independientes, y porque las imágenes tienen una gestión diferente, pues existen varias escalas, ocupan bastante más tamaño en MegaBytes que el código, y en caso de escalar la funcionalidad de la gestión, es más fácil implementar a partir de un servidor especializado en el exclusivo trato de las imágenes. Así que añadiendo mayor funcionalidad al sistema se necesita un servidor FTP para la transferencia de estas imágenes y un servidor de imágenes.

Figura 10

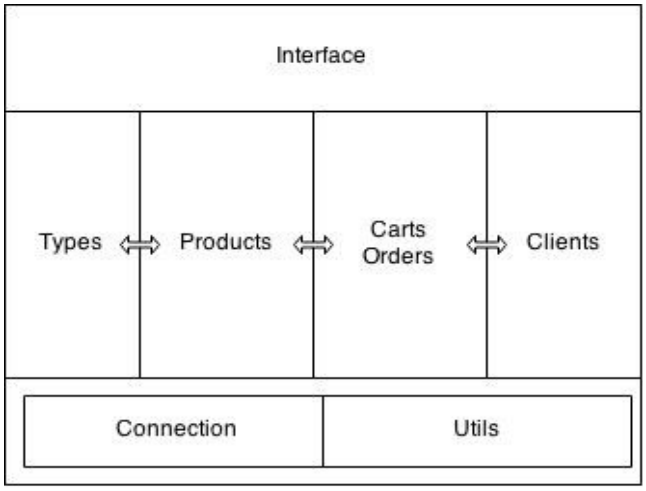


Arquitectura distribuida

4.1.1 Arquitectura modular

Ahondando dentro de la arquitectura de las aplicaciones que procesan los datos (Frontend y Backend), se separan en módulos dependiendo de la naturaleza de los datos. De manera que se puede escalar en funcionalidad fácilmente con poca dependencia entre módulos, dependerá de si necesitan interactuar con otros módulos o no.

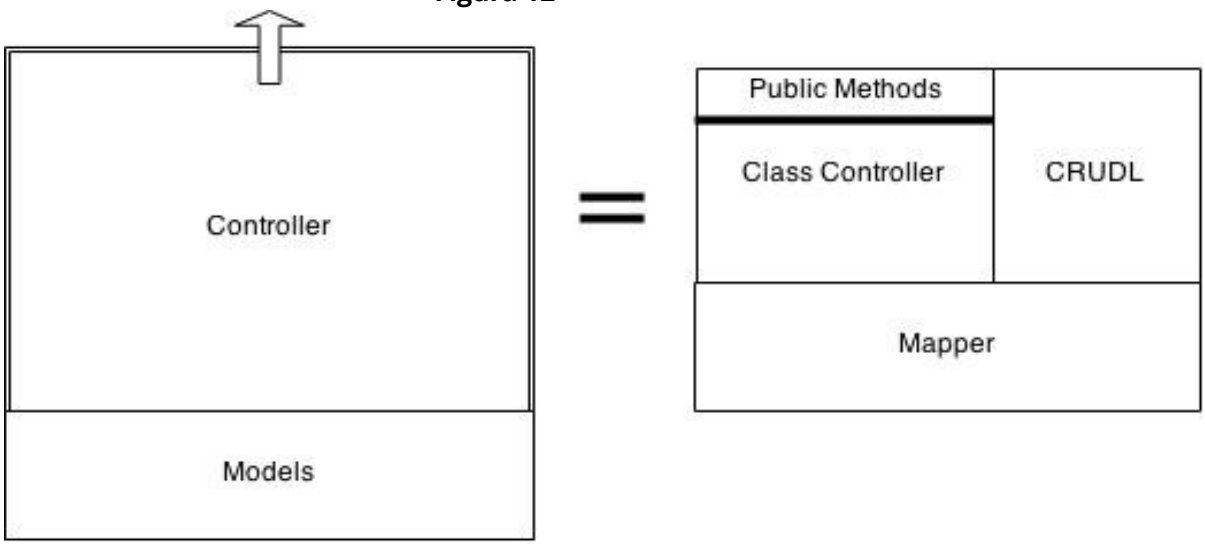
Figura 11



4.1.2 Modelo – Vista- Controlador

Por cada módulo se aplica el patrón arquitectónico MVC (Model – View – Controller) que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y el módulo encargado de gestionar los eventos y las comunicaciones. Esta forma de organizar el código facilita la reutilización del mismo y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su posterior mantenimiento.

Figura 12



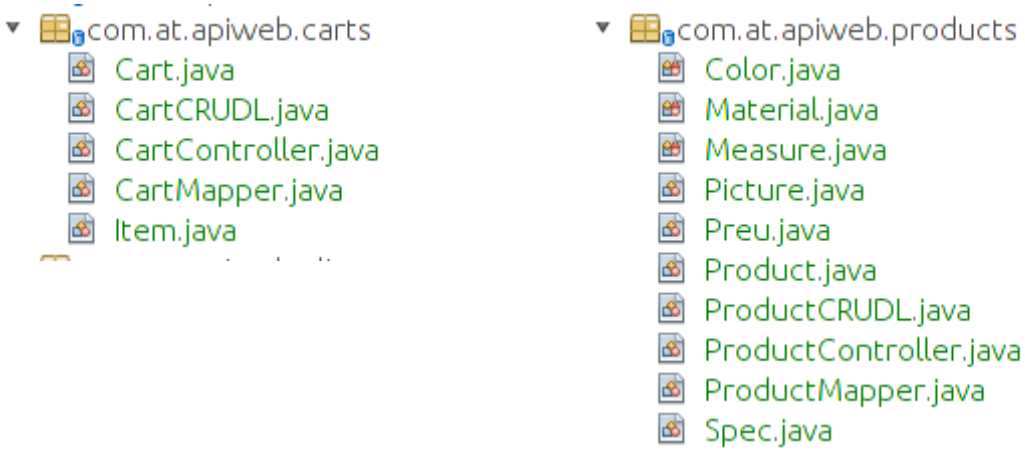
Desglose del controlador

Dentro de la capa modelos se encuentran los POJOS (Plain Old Java Object) con los atributos que se requieren para el objeto, que pueden ser diferentes en Frontend y en Backend, dependiendo de sus necesidades. Por otra parte en la capa controlador entran en comunicación diferentes clases que son separadas por tener distintos propósitos. La clase controller es la que se encarga de gestionar toda la lógica del módulo para que la vista a partir de métodos públicos pueda trabajar, independiente de lo cómo se gestionen las funcionalidades. Esta clase controlador se comunica con las bases de datos a través de la clase CRUDL de cada módulo, y permite diferente tipo de interacciones especializadas, como son crear, leer, actualizar, eliminar y listar. En respuesta a los métodos de la clase CRUDL se debe mapear el resultado como objetos de las clases de la capa de modelos, ahí entra el trabajo de la clase Mapper.

Los módulos que no necesitan una gran gestión, ni consultas complicadas se trabajan desde el framework Morphia, que obvia la clase Mapper de forma automática, ya que el mismo framework se encarga de construir las consultas y de los mapeos. El controlador y la clase CRUDL internamente funcionaría de forma diferente de acuerdo

con Morphia y los modelos deberían seguir también las anotaciones de los atributos de Morphia para que se pudiera mapear de forma automática.

Figura 13



Vista de la modularización junto a MVC

Figura 14

```
public static Product[] list(Map<String, String[]> requestParams){
    if(requestParams.containsKey("news")){
        int year = Integer.parseInt(requestParams.get("news")[0]);
        return creaArray(ProductCRUDL.listByYear(year));
    }
    if(requestParams.containsKey("type")){
        String slugType = requestParams.get("type")[0];
        return creaArray(ProductCRUDL.listByType(slugType));
    }
    return creaArray(ProductCRUDL.list());
}
```

Muestra de un método de listado por parámetros

4.2 Arquitectura de la información

La información está gestionada mediante un sistema de base de datos no relacional orientado a documentos, llamado MongoDB. Esto permite unificar datos comunes y dependientes en un solo documento donde con un sistema relacional se necesitarían varias tablas relacionadas. En el apartado sobre tecnologías se ahonda más en algunos de los beneficios que ofrece MongoDB.

Después de varias iteraciones en el desarrollo del producto, la estructura de las bases de datos ha ido mejorándose progresivamente y ajustándose al sistema y a las necesidades del cliente, buscando la eficiencia, seguridad siempre bajo el paradigma de la orientación a documentos.

Existen dos bases de datos diferentes porque se quiere independencia y seguridad, una para separar la parte interna de gestión de la empresa (at) de la parte externa que de funcionamiento a la web (at-web). Aún así el sistema perteneciente a la parte externa tendrá permisos de lectura sobre at.

A parte existe información estática en forma de clases de tipo *enum* dentro de los subsistemas que no pertenecen a la DDBB en beneficio del tiempo de respuesta y para evitar relaciones innecesarias con colecciones superfluas. En estas clases se almacenan propiedades de colores, como el código RGB, número identificatorio y traducción en otros idiomas. También se almacenan las distintas medidas estandarizadas de la tipología de los productos y el nombre de los materiales más utilizados como materia prima.

4.2.1 DB: at

Colecciones: {products, types, clients, orders}

Esta DDBB conforma el núcleo de la información principal de la organización, aquí se fundamenta la estructura del negocio, los productos, los clientes y los pedidos.
Además se permite la organización de productos en tipos y subtipos. A continuación se describen y se ejemplifican cada una de las colecciones.

Productos
Ejemplo de un documento semi-estructurado de la colección “productes”:

```
{
   "_id" : ObjectId("53fee76eafb358a62d7684c7"),
   "ref" : "0006",
   "type" : "abanicos_sin_decorar",
   "any" : 2013,
   "visible" : true,
   "spec" : {
      "mida" : "STANDARD",
      "pintat" : true,
      "calat" : false,
      "material" : "MADERA",
      "colors" : [  "NEGRO",  "AVELLANA",  "NOGAL",  "CREMA",  "PISTACHO",  "CELESTE",
         "MALVA", "VERDE",  "NATURAL",  "NARANJA",  "AMARILLO",  "AZUL",  "MARRON",
         "FUCSIA", "ROSA",  "ORO", "PLATA"
      ]
   },
   "preu" : { "web" : 4, "client" : 4,"cost" : 4},
   "pictures" : [
      {
         "direction" : "1127201412204214 70",
         "format" : ".jpg",
         "title" : "0006",
         "date" : ISODate("2014-11-27T11:20:55.957Z")
      },
      {
         "direction" : "10152014165216272",
         "format" : ".jpg",
         "title" : "0006b",
         "date" : ISODate("2014-10-15T14:52:22.956Z")
      }
   ],
   "mainpicture" : {
      "direction" : "1127201412204214 70",
      "format" : ".jpg",
      "title" : "0006",
      "date" : ISODate("2014-11-27T11:20:55.957Z")
   }
}
```

El documento es practicamente legible y entendible en su totalidad y está visiblemente desgranado en subdocumentos. En el primer nivel están los atributos más básicos como su referencia, tipología, el año de lanzamiento y si es visible como disponible, es decir si está preparado para lanzarse al público. Más tarde se agrupan características que también son plurales para todos los productos, pero que pueden estar formando conjuntos y así estar mejor organizados como es el caso de los diferentes tipos de precios disponibles, tanto de coste de producción como precio para mayorista y minorista. Una matriz de imágenes con diferentes propiedades básicas como su fecha, nombre y formato de la misma. De igual forma se duplica una de ellas como imagen principal. Esta duplicación está fundamentada para agilizar las consultas y no cargar todas las imágenes ni buscar una dentro de una maraña de imágenes que podrían ralentizar la respuesta. De esta manera llamando a un campo concreto de la colección se extrae la imagen de muestra. Además cada tipo de producto presenta unas especificaciones distintas entre familias de productos, es por ello por lo que hay un subdocumento especializado libre de estructura global para enfocarse en la descripción más especializada en la concreción de un producto.

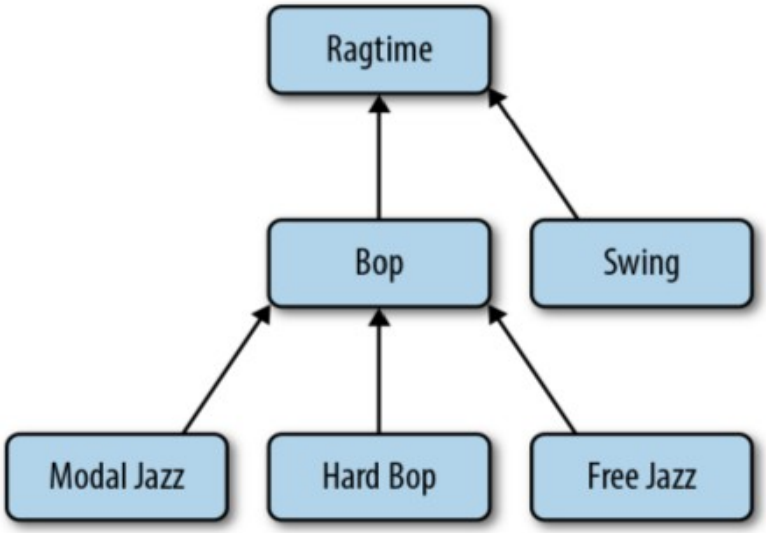
Además de ser fácilmente legible para la percepción humana gracias a la estructura JSON, sin necesidad de relaciones entre tablas, las consultas son muy directas, porque toda la información al respecto de un producto está dentro de la misma colección del producto.

Tipos
Ejemplo de un documento semi-estructurado de la colección “types”:

```
{
   "_id" : ObjectId("54cf2f1197b6cfa81f913f38"),
   "slug" : "abanicos_especiales",
   "language" : {
      "es" : "Abanicos Especiales",
      "en" : "Specials Fans"
   },
   "parent" : ObjectId("54cf2ec197b6cfa81f913f36")
}
```

Esta estructura sigue el patrón de diseño “Categoy Hierarchy” definida para MongoDB por Rick Copeland.

Figura 15



Category Hierarchy

Este documento ha necesitado bastantes reestructuraciones hasta llegar a la simplicidad del de ahora. Por otra parte el resultado además de ser sencillo, es totalmente escalable y fácilmente editable, además de ser mínimo en tamaño. Por mostrar como ha ido cambiando, a continuación se describe la estructura predecesora.

```
{
  "_id" : ObjectId("53fae88aafb3dc3494e03225"),
  "name" : "Fan",
  "nombre" : "Abanico",
  "subtypes" : [
    {
      "slug" : "especiales",
      "name" : "Specials",
      "nombre" : "Especiales",
      "products" : [
        {
          "_id" : ObjectId("53fee76eafb358a62d7684c7"),
          "ref" : "0006",
          "any" : 0,
          "visible" : true,
          "mainpicture" : {
            "direction" : "112720141220421470",
            "format" : ".jpg"
          },

          "preu" : {
            "web" : 2,
            "client" : 2,
            "cost" : 0
          }
        },
        {
          "_id" : ObjectId("53fee6bfafb358a62d7684c5"),
          "ref" : "0003",
          "any" : 0,
          "visible" : true,
          "mainpicture" : {
            "direction" : "101520141645171131",
            "format" : ".jpg"
          },
          "preu" : {
            "web" : 1,
            "client" : 1,
            "cost" : 0
          }
        }
      ]
```

```
    },
    {
      "slug" : "novia",
      "name" : "Weeding",
      "nombre" : "Novia",
      "products" : [
        {
          "_id" : ObjectId("53fee656afb358a62d7684c4"),
          "ref" : "0002",
          "any" : 0,
          "visible" : true,
          "mainpicture" : {
            "direction" : "101520141644387",
            "format" : ".jpg"
          },
          "preu" : {
            "web" : 2.299999952316284,
            "client" : 2.299999952316284,
            "cost" : 0
          }
        },
        {
          "_id" : ObjectId("53fee4fcafb358a62d7684c3"),
          "ref" : "0001",
          "any" : 0,
          "visible" : true,
          "mainpicture" : {
            "direction" : "101520141100291562",
            "format" : ".jpg"
          },
          "preu" : {
            "web" : 1.75,
            "client" : 1.75,
            "cost" : 0
          }
        }
      ]
    }
  ]
}
```

Este documento es bastante más complejo porque además de todas las propiedades de un tipo embebe todos los subtipos con sus propias propiedades. Además dentro de cada subtipo existe un array de productos que tan solo contiene la información necesaria para el listado de los productos como su referencia, la imagen principal o los diferentes precios.

El documento puede engordar conforme se añaden productos a los subtipos, por lo que se necesita concretar las proyecciones de las consultas para conseguir una máxima eficiencia y recolectar tan solo los datos que se vayan a utilizar
Por ejemplo; para conseguir solo los productos de un subtipo concreto se debería formular la query tal que así:

db.products.find({"name": "TypeName", "subtypes.name": "SubtypeName"}, {"subtypes": {$elemMatch: {"name": "SubtypeName"}}, "subtypes.products": 1 })

Mientras que para recopilar solo los tipos y los subtipos sin ningún producto, para mostrar la información necesaria para un menú de navegación, la query sería así:

db.products.find({"name": "TypeName"}, {“name”:1, "subtypes.name": 1 })

Aunque se guarde la *_id* del producto, en ningún caso es necesario establecer ninguna relación pues la información a mostrar ya está embebida en la misma colección, la identificación será simplemente un atributo más para trabajar. De todas formas hay que tener cuidado en no perder la integridad y consistencia de los datos, y eso requerirá un esfuerzo mayor en el desarrollo de las funcionalidades CRUD (Create, Read, Update, Delete) de los productos.

Las dos alternativas son eficientes en tiempo de respuesta, más rápida la segunda siempre y cuando el documento no sea demasiado extenso, aún así, en mantenimiento resulta mucho más sencillo para la primera, y las consultas son directas en las dos opciones, siendo más sencillas en la primera.

Clientes
Ejemplo de un documento semi-estructurado de la colección “clients”:

```
{
   "_id" : ObjectId("546e0ded44ae69c9c00705ea")
   "nom" : {
      "nom" : "Emilio",
      "llinage" : "Rodriguez"
   },
   "email" : "emilio@rodriguez.com",
   "password" : "rodriRodri_28",
   "telf" : ["961520521","961522314"],
   "direccio" : {
      "via" : "C/ San Emilio, 22",
      "cp" : "46975",
      "ciudad" : "Aldaya",
      "provincia" : "Valencia",
      "pais" : "España"
   },
   "blocked" : false
}
```

El documento es sencillo, con los datos básicos del cliente, se crea un subdocumento para la dirección postal, una matriz para los teléfonos, un subdocumento para la distinción de nombre y apellido y un atributo de control “blocked” para el acceso a la aplicación web.

Pedidos

Ejemplo de un documento estructurado de la colección “orders”:

```
{
   "_id" : ObjectId("54e4728b44ae808ee0b48c20"),
   "client" : {
      "nom" : {
         "nom" : "Baldo",
         "llinage" : "Taberner"
      },
      "email" : "baldo@mail.com",
      "telf" : "66426978",
      "direccio" : {
         "via" : "C/ Mayor, 21",
         "cp" : "46967",
         "ciudad" : "Aldaya",
         "provincia" : "Valencia",
         "pais" : "España"
      }
   },
   "date" : ISODate("2015-02-14T12:43:14.531Z"),
   "items" : [
      {
         "ref" : "0032",
         "price" : 4.2,
         "quantity" : 1,
         "details" : "NEGRO"
      },
      {
         "ref" : "0053",
         "price" : 4.45,
         "quantity" : 2,
         "details" : "Todos Blancos"
      }
   ],
   "total" : 12.7
}
```

Este documento define como es un pedido tal cual en la vida real; tiene una fecha que corresponde al momento en el que se crea, los datos del cliente para su envío y posible contacto vía web y telefónico, así como los productos de manera estructurada. El campo total externo a items facilita su cálculo a nivel general.

4.2.2 DB: at-web

Colecciones: {carts}

Esta base de datos actualmente solo contiene una colección, pero tiene su razón de ser. Esta base de datos es la única que otorga permisos de escritura desde la aplicación cliente, con motivo de ofrecer mayor seguridad al sistema. Por lo que en la actualidad el único movimiento de escritura que gestiona desde la parte cliente es la función del carrito de la compra, pero se pretende escalar en funcionalidad, registrar visitas y generar contenido de interés para mejorar el sistema.

Carts

Ejemplo de un documento estructurado de la colección “carts”:

```
{
   "_id" : ObjectId("5460e3ca44aecd4cfc90a8c4"),
   "lastModified" : ISODate("2015-02-14T12:43:14.531Z"),
   "items" : [
      {
         "ref" : "0032",
         "price" : 4.2,
         "quantity" : 1,
         "details" : "NEGRO"
      },
      {
         "ref" : "0053",
         "price" : 4.45,
         "quantity" : 23,
         "details" : "Todos Blancos"
      }
   ],
   "sent" : true
}
```

Esta colección es muy parecida a la de pedidos, es totalmente estructurada y también ha sufrido algunas reestructuraciones hasta llegar a la actual.

El campo *sent*, cuando está *true* indica que el pedido ha sido enviado y no es modificable hasta la confirmación de la empresa. El resto de los atributos están definidos como la colección de pedidos.

4.2.3 Datos Java

En la aplicación hay datos que estuvieron en un inicio formando parte de la base de datos, pero por mejorar su rendimiento se externalizó para formar parte de la aplicación. Son datos que no necesitan tener ninguna identificación y son atributos generalizados de los productos como son el caso de medidas, colores y materiales. De forma que quedan enlazados con la identificación del enum como String. Además al guardarse de esta forma, no necesita ningún tipo de mapeo porque ya quedan como un objeto con diferentes atributos.

Figura 16

```
public enum Color {
    NEGRO("Negro", "Black", "1", "#000000"),
    AVELLANA("Avellana", "Hazelnut", "2", "#CF7F0C"),
    NOGAL("Nogal", "Walnut", "3", "#AB6F19"),
    GRANATE("Granate", "Garnet", "4", "#9D2727"),
    MARINO("Marino", "Navy", "5", "#27379D"),
    VERDEOSC("Verde oscuro", "Dark green", "6", "#3C9937"),
    BLANCO("Blanco", "White", "7", "#FFFFFF"),
    ROJO("Rojo", "Red", "8", "#F5011E"),
    CREMA("Crema", "Cream", "9", "#F6F9C2"),
    PISTACHO("Pistacho", "Pistachio", "10", "#86CD17"),
    CELESTE("Celeste", "Sky-blue", "12", "#8AC9F9"),
    MALVA("Malva", "Mallow", "13", "#B888D3"),
    VERDE("Verde", "Green", "14", "#48B636"),
    NATURAL("Natural", "Natural", "15", "#B9A897"),
    NARANJA("Naranja", "Orange", "16", "#FF931F"),
    AMARILLO("Amarillo", "Yellow", "17", "#CAF500"),
    AZUL("Azul", "Blue", "18", "#3198C7"),
    MARRON("Marron", "Brown", "21", "#8D3731"),
    FUCSIA("Fucsia", "Fuchsia", "22", "#E853C0"),
    ROSA("Rosa", "Pink", "23", "#FAACE5"),
    ORO("Oro", "Glod", "24", "#DBB000"),
    PLATA("Plata", "Silver", "25", "#A5A5A5");
```

Enum Color

En el caso de los colores, el color en formato RGB está como atributo. El color en sí, no está definido en ninguna tonalidad o pantón en concreto, simplemente es una muestra del color.

Figura 17

```
public enum Measure {
    PERICON("Pericon", 32),
    SEMIPERICON("Semi-pericon", 27),
    STANDARD("Standard", 23),
    SEMISTANDARD("Semi-standard", 21),
    BOLSO("Bolso", 19),
    FIESTA("Fiesta", 16),
    MINI("Mini", 12);
```

Enum Measure

4.3 Arquitectura del entorno

El sistema se despliega dentro de una maquina virtual escalable que se adecua a las necesidades del momento, sigue la tecnología EC2 (Elastic Compute Cloud) y posee dirección IP estática. En la máquina corre como sistema operativo Ubuntu 14 al que se la han añadido diferentes aplicaciones:

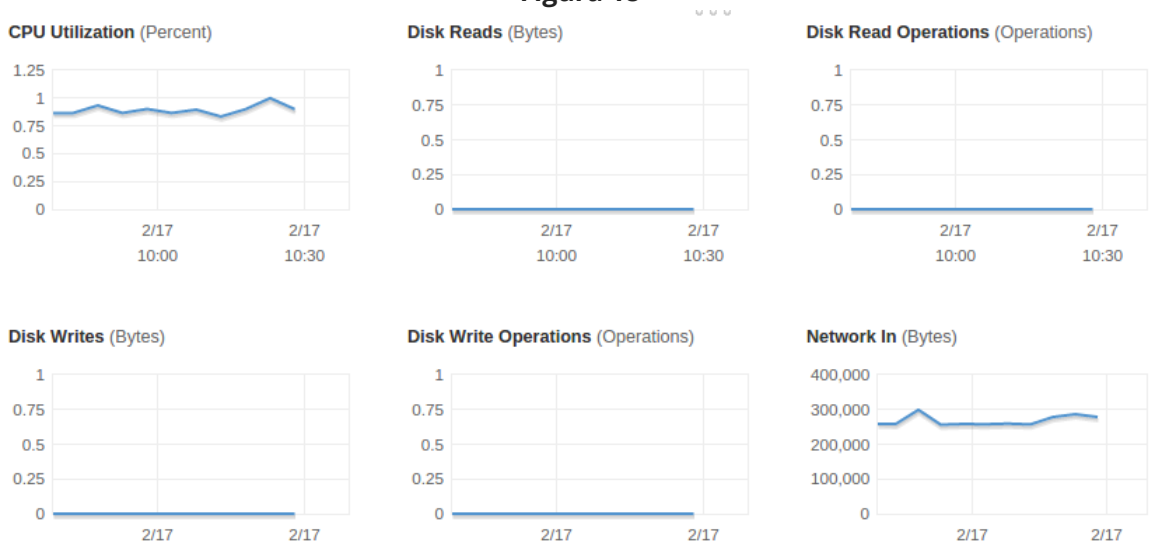
- Java Development Kit (JDK)
- SSH Server
- MongoDB Server
- Glassfish Server
- FTP Server
- Apache HTTP Server

4.3.1 Procedimiento operacional

La comunicación real con el servidor, será mediante SSH y un key name personalizado para garantizar mayor seguridad en el acceso. El despliegue de la aplicación se realiza en Glassfish que disponiendo de MongoDB con el puerto de escucha abierto, se comunica automáticamente. Lo mismo ocurre con el servidor FTP y la gestión del servidor de imágenes que se hospeda en Apache HTTP Server, de manera que es todo transparente en el despliegue.

La creación de copias de seguridad no está automatizada. Para ello se debe ingresar en el sistema por consola y trabajar con los instrumentos que destina MongoDB para los casos de importar o exportar archivos JSON (mongoimport/mongoexport) o en archivos BSON (mongodump/mongorestore).

Figura 18



Gestión del servidor

Capítulo 5. Tecnologías

Para la realización del sistema en su parte más técnica se necesitan la interacción e integración de varias tecnologías, cada una especializada en una parte concreta del proyecto. Se definen desde tecnologías extensas como bases de datos o lenguajes de programación hasta frameworks especializados en casos concretos y pasando por aplicaciones como servidores o instrumentos de gestión de proyectos software.

Figura 19

5.1 MongoDB

MongoDB es un gestor de bases de datos de código abierto y forma parte parte de la familia de las bases de datos NoSQL. En vez de guardar los datos en tablas como se hace en las base de datos relacionales, MongoDB guarda estructuras de datos en documentos. Los documentos son almacenados en BSON, que es una representación binaria de JSON, siendo a la vez fácil y elegante la percepción para el ojo humano.

Una de las bondades más importantes que posee con respecto a las bases de datos relacionales, es que no es necesario seguir un esquema. Los documentos de una misma colección – concepto similar a una tabla de una base de datos relacional -, pueden tener esquemas diferentes. Esto permite cambiar rápidamente cuando las aplicaciones evolucionan, proporcionando siempre la funcionalidad que los desarrolladores esperan de las bases de datos tradicionales, tales como índices secundarios, un lenguaje completo de búsquedas y consistencia estricta.

La libertad que proporciona MongoDB es abismal con respecto a la tecnología SQL, todo esto es bueno y malo, según como se mire. Cada vez existen más datos semiestructurados o sin ninguna estructura. Por ejemplo para la realización de este proyecto el cliente puede tener productos de diferente índole con diferentes propiedades, de manera que un documento que almacene un producto de tipo A, podrá ser totalmente distinto en su estructura al que almacene un producto del tipo B y estará hospedado en la misma colección de documentos. Se evita de esta manera campos vacíos inservibles o diferentes tablas por tipo de producto. Otra gran bondad es que permite crear subdocumentos o incluso listas de más documentos, eliminando la dependencia relacional y la creación de tablas que sólo sirven para mantener la estructura SQL, de esta forma se evita el uso desproporcionado de la cláusula JOIN que termina por sobrecargar las consultas y el tiempo de respuesta del gestor de base de datos conforme escala en registros. Véase la Figura 20.

Figura 20

```
{
    _id: 1,
    "nombre": {"pila": "Francisco", "apellido": "Valero"},
    "ciudad" : "Valencia",
    "telefono" : ["951234385","646728873","961233212"]
}
```

Personas

id	nombre	apellido	ciudad
1	Francisco	Valero	Valencia

Telefonos

id	idPersona	telefono
1	1	951234385
2	1	646728873
3	1	961233212

MongoDB Vs SQL

MongoDB ha sido creado para brindar escalabilidad, rendimiento y gran disponibilidad, escalando de una implantación de servidor único a grandes arquitecturas complejas de centros multidatos. MongoDB brinda un elevado rendimiento, tanto para lectura como para escritura, potenciando la computación en memoria (in-memory). La replicación nativa de MongoDB y la tolerancia a fallos automática ofrece fiabilidad a nivel empresarial y flexibilidad operativa.

Todos estos beneficios deben de ser controlados por el desarrollador y la arquitectura del sistema debe de ser capaz de trabajar con todas las variables que puedan existir en la base de datos. Además, por priorizar tiempos de respuesta más ágiles y estructuración de los datos, se puede caer en la fatal inconsistencia de los mismos, por lo que también debería de controlarse en la programación, pudiendo ser a veces más compleja que en SQL.

Por otro lado MongoDB facilita diferentes instrumentos para aumentar la productividad en su gestión como *mongostat* que crea un listado de estadísticas avanzado de una instancia en ejecución. M*ongotop* provee un método para dar seguimiento a la cantidad de tiempo que dura una la lectura o escritura de datos en una instancia. También provee estadísticas en el nivel de cada colección. *Mongosniff* es un instrumento de línea de comandos que provee un sniffing en la base de datos haciendo un sniffing en el tráfico de la red que va desde y hacia MongoDB. También se permiten copias de seguridad en JSON usando *mongoexport* / *mongoimport* y en BSON usando *mongodump* / *mongorestore*

5.2 Java

Java es un lenguaje de programación de propósito general, concurrente, orientado a objetos que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Capaz de correr sobre diversos sistemas operativos sin necesidad de ser recompilado y gracias a su extensa comunidad, Java es, a partir de 2012, uno de los lenguajes de programación más populares en uso, particularmente para aplicaciones de cliente-servidor de web, con unos 10 millones de usuarios reportados. Todas estas características soportan la decisión de elegirlo como base fundamental del desarrollo del proyecto.

Al poseer una comunidad activa tan grande, soporta una cantidad de tecnologías inmensa desde las más viejas hasta las más novedosas y tiene multitud de

frameworks para diferentes usos. En este punto se habla de las diferentes APIs que se usan en este proyecto para conseguir un sistema más robusto, aprovechar otras tecnologías complementarias y conseguir más seguridad y tolerancia a fallos.

5.2.1 Maven

Maven es una herramienta de software para la gestión y construcción de proyectos Java, similar en funcionalidad a Apache Ant. Tiene un modelo de configuración de construcción simple, basado en un formato XML.

Maven utiliza un Project Object Model (POM) para describir el proyecto de software a construir, sus dependencias de otros módulos y componentes externos, y el orden de construcción de los elementos. Viene con objetivos predefinidos para realizar ciertas tareas claramente definidas, como la compilación del código y su empaquetado.

Con Maven es muy fácil integrar nuevas dependencias al proyecto, actualizando su fichero POM, -existiendo solamente uno por cada programa Java- al compilar el programa, Maven se encarga de recopilar todas las dependencias que existen hospedadas en distintos servidores y las empaqueta dentro del proyecto. Esta forma de trabajar es muy útil cuando se trabaja en equipo, cuando se usan controladores de versiones, debido a que la dependencia en vez de ser un ejecutable es un registro en un documento XML. En la Figura 21 se muestra un ejemplo de un archivo POM del proyecto.

Figura 21

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.at</groupId>
    <artifactId>Frontend</artifactId>
    <version>1.1.1</version>
    <packaging>war</packaging>

    <name>Frontend</name>

    <properties>
        <endorsed.dir>${project.build.directory}/endorsed</endorsed.dir>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>

    <dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.10</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>pdfbox</groupId>
            <artifactId>pdfbox</artifactId>
            <version>0.7.3</version>
            <type>jar</type>
        </dependency>
        <dependency>
            <groupId>javax</groupId>
```

Sistema de dependencias

5.2.2 jUnit

JUnit 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.

Al estar utilizando técnicas de *refactoring* continuamente y aplicando *Test Driven Development*, jUnit es una herramienta crucial para garantizar el mantenimiento de nuestro sistema. Aún así la dependencia a métodos que trabajan con datos de la base de datos, impiden un correcto testeo que controle cada una de las funciones públicas que pueda tener el proyecto.

En la siguiente imagen se muestran los diferentes test que se le aplican al método que valida un DNI.

Figura 22

```
@Test
public void testCheckDNI() {
    System.out.println("CheckDNI");

    //null
    Assert.assertFalse(Person.checkDNI(null));

    //Incorrecto
    Assert.assertFalse(Person.checkDNI("inventado"));

    //Correcto
    Assert.assertTrue(Person.checkDNI("48586484L"));

    //Correcto con letra minuscula
    Assert.assertTrue(Person.checkDNI("48586484l"));

    //Incorrecto
    Assert.assertFalse(Person.checkDNI("48586484M"));

}
```

5.2.3 Mongo Java Driver

Mongo Java Driver es la API oficial que distribuye MongoDB para trabajar desde Java. Es fácilmente intuitiva una vez se tienen establecidos los conocimientos básicos y se crean las conexiones pertinentes. A partir de este punto las consultas que atacan a Mongo son muy parecidas a las que se usan contra la consola. Basta aprender a trabajar con los distintos objetos que provee la API

com.mongodb.DBObject;
com.mongodb.DBCollection;
com.mongodb.DBCursor;
org.bson.types.ObjectId;

Por ejemplo, en el proyecto se listan los tipos según se ha definido en la arquitectura, pudiendo tener tipo padre o no, con lo que se necesita una consulta sobre consola según lo que se requiera.
Para una consulta que devuelva los tipos de primer nivel

db.types.find({parent:{$exists : false}},{_id:1, “slud”:1, “language”:1})

Para una consulta que devuelva los subtipos de un determinado tipo

db.types.find({parent: ObjectId("54cf2ec197b6cfa81f913f36")},{_id:1, “slud”:1, “language”:1}))

La consulta en Java desde el framework que provee MongoDB es algo tal como se muestra en la Figura 23.

Figura 23

Muestra de consulta

```
    protected static DBCursor list(String idParent){
        BasicDBObject query;
        if(idParent != null){
            ObjectId oid = new ObjectId(idParent);

            query = new BasicDBObject();
                query.put("parent", oid);
        } else{
            BasicDBObject exist = new BasicDBObject();
                exist.put("$exists", false);
            query = new BasicDBObject();
                query.put("parent", exist);
        }

        BasicDBObject projection = new BasicDBObject();
            projection.put("_id",1);
            projection.put("slug",1);
            projection.put("language", 1);

        return collection.find(query, projection);
    }
}
```

5.2.4 Morphia

Morphia es una biblioteca ligera para el mapeo de objetos Java a partir de documentos BSON en MongoDB. Morphia ofrece un tipado seguro de los objetos, y trabaja sobre la API Mongo Java Driver, para que las consultas se hagan con fluidez en tiempo de ejecución y de validación.

Morphia utiliza anotaciones para que no existan más archivos XML de gestión. Y el mapeo sea totalmente automatizado y transparente al desarrollador, que con un poco de experiencia previa en otros mapeadores como JPA, se es capaz de sacarle un gran rendimiento.

En este proyecto se hace gala de Morphia en la parte del backend para desarrollar los CRUDL (Create, Read, Update, Delete, List) de una manera sencilla para cada uno de los elementos básicos que componen el sistema.

Para dar un pequeño ejemplo de su uso, cuando se define un POJO (Plain Old Java Object) se insertan algunas anotaciones para facilitar al framework el mapeo. Se establece que variable es el identificador, que datos se indexan y que otros se embeben como subdocumentos, pudiendo crear incluso listas. El siguiente código muestra uno de los POJOS que Morphia mapea en el proyecto.

Figura 24

```
@Entity(value = "productes", noClassnameStored = true)
public class Producte {
    @Id private ObjectId id;
    @Indexed private String ref;
    @Indexed private String type;
    @Indexed private int any;
    @Indexed private Boolean visible;

    @Embedded private Picture mainpicture;

    @Embedded private Preu preu;

    @Embedded private Spec spec;

    @Embedded private List<Picture> pictures;
```

Muestra de POJO con Morphia

A partir de que el mapeo esté establecido se puede trabajar fácilmente con los diferentes métodos que aporta la librería. Según el código de bajo cabe destacar la facilidad de crear o eliminar un documento en la colección, con tan solo pasarle al método el objeto en cuestión.

A la hora de crear consultas simples también es muy intuitivo, para filtrar o crear proyecciones. Sin embargo Morphia se complica conforme las consultas se complican. Es decir, para actualizar campos de subdocumentos en documentos Morphia no rinde con la misma libertad que permite el API oficial que hay por debajo, por lo que el proyecto trabaja según para qué objetivos de una forma o de otra.

Figura 25

```
@Override
public void create(Producte producte){
    datastore.save(producte);
}

@Override
public Producte read(String id){
    ObjectId objectId = new ObjectId(id);
    return datastore.get(Producte.class, objectId);
}

public Producte readByRef(String ref){
    Query<Producte> query = datastore.find(Producte.class)
        .filter("ref", ref)
        .retrievedFields(true, "ref", "_id", "visible", "mainpicture.direction", "mainpicture.format",
    return query.get();
}

@Override
public void update(String id, Producte objectUpdated){
    Producte producte = read(id);

    UpdateOperations<Producte> ops = datastore.createUpdateOperations(Producte.class)
            .set("ref", objectUpdated.getRef())
            .set("type", objectUpdated.getType())
            .set("any", objectUpdated.getAny())
            .set("visible", objectUpdated.getVisible())
            .set("spec", objectUpdated.getSpec())
            .set("preu", objectUpdated.getPreu());
    datastore.update(producte, ops);
}
```

Muestra de CRUDL básico

5.2.5 Java Server Pages

JavaServer Pages (JSP) es una tecnología que ayuda a los desarrolladores de software a crear páginas web dinámicas basadas en HTML, XML, entre otros tipos de documentos. JSP es similar a PHP, pero usa el lenguaje de programación Java.

El rendimiento de una página JSP es el mismo que tendría el servlet equivalente, ya que el código es compilado como cualquier otra clase Java, por lo que puede ser visto como una abstracción de alto nivel de los servlets Java. A su vez, la máquina virtual compila dinámicamente a código de máquina las partes de la aplicación que lo requieren. Esto hace que JSP tenga un buen desempeño y sea más eficiente que otras tecnologías web que ejecutan el código de una manera puramente interpretada.

Al final se trata de embeber el código java, usando etiquetas especiales aportadas por la API de Java Server Pages, dentro del etiquetado HTML que tiene cualquier web, siendo el formato del documento un .jsp.

Figura 26

```
  </div>
  <div id="navbar" class="navbar-collapse collapse">
    <ul class="nav navbar-nav" style="margin-top: 5px;">
      <li><a href="<%=request.getContextPath()%>/elements/productes/list.jsp?news=2015">Novedades</a></
      <% for(Type type : TypeController.list()) { %>
          <li class="dropdown">
              <a href="<%=request.getContextPath()%>/elements/productes/list.jsp?type=<%= type.getId().
              <ul class="dropdown-menu" role="menu">
                  <% for(Type subtype : TypeController.list(type.getId().toString())){ %>
                      <li><a href="<%=request.getContextPath()%>/elements/productes/list.jsp?type=<%= s
                  <% } %>
              </ul>
          </li>
      <% } %>
    </ul>
    <ul class="nav navbar-nav navbar-right">
          <% if(session.getAttribute("userid")==null){ %>
              <li><a rel="nofollow" href="<%=request.getContextPath()%>/login.jsp"><span style="font-si
          <% }else {
              String userid = session.getAttribute("userid").toString(); %>
              <li><a href="<%=request.getContextPath()%>/elements/pedidos/view.jsp"><span style="font-s
              <li><a href="<%=request.getContextPath()%>/Logout"><span style="font-size: 2em;" class="g
          <% } %>
          <li><a href="<%=request.getContextPath()%>/about.jsp"><span style="font-size: 2em;"  class="g
          <li><a href="<%=request.getContextPath()%>/contact.jsp"><span style="font-size: 2em;" class="
    </ul>
  </div><!--/.nav-collapse -->
/nav>
```

Muestra de JSF

5.2.6 PDFbox

Apache PDFBox es una herramienta Java de código abierto para trabajar con documentos PDF. Este proyecto permite la creación de nuevos documentos PDF, manipulación de documentos existentes y la capacidad de extraer el contenido de los documentos.

Sin embargo en este proyecto sólo es usado para generar nuevos proyectos PDF para catálogos o pedidos de creación automática. El uso es realmente sencillo cuando se trata de simple texto, por otra parte lo interesante aquí es la creación del catálogo con las imágenes de cada producto, que curiosamente se debe de generar como objeto *Image* antes que el objeto contenedor de toda la información de una página PDF. Salvando la dudosa explicación técnica que pueda tener, el uso del framework es sencillo e intuitivo.

Figura 27

```
url = new URL(producte.getPictureDirection());
image = ImageIO.read(url);

ximage = new PDJpeg(doc, toBufferedImage(image));


try (PDPageContentStream content = new PDPageContentStream(doc, page, true, true)) {

    content.drawImage(ximage, 20, 20);

    PDFont font = PDType1Font.HELVETICA_BOLD;
    content.beginText();
    content.setFont( font, fontSize+fontSize );
    content.moveTextPositionByAmount( 100, 700 );
    content.drawString("Ref. "+producte.getRef());
    content.setFont( font, fontSize );

    content.moveTextPositionByAmount(0, -fontSize * 1.2f);
    content.drawString("Material: "+producte.getMaterial());

    content.moveTextPositionByAmount(0, -fontSize * 1.2f);
    content.drawString("Medida "+producte.getMida());

    content.moveTextPositionByAmount(0, -fontSize * 1.2f);
    content.drawString("Precio: "+producte.getPreu());

    content.moveTextPositionByAmount(0, -fontSize * 1.2f);
    content.drawString("Colores:");
    List<String> colors = producte.getColors();
    for(String color : colors){
        content.moveTextPositionByAmount(0, -fontSize * 1.2f);
        content.drawString(color);
```

Muestra de PDFBox

5.2.7 FTPClient

La posibilidad de conectarse a un FTP con Java no es una utilidad que nos den las librerías base de Java. Es por ello que para poder acometer esta tarea y conectarnos a un FTP con Java deberemos de utilizar las librerías de Apache Commons. En concreto el componente Net.

En este proyecto se da uso de un cliente FTP para la gestión de imágenes en un servidor web, por ello se trabaja con la clase FTPClient y una vez recibido los datos de conexión necesarios para el acceso se crean los métodos para conectar y desconectar.

A partir de conectar al servidor FTP se cargan, mediante un *streaming,* tres copias de una misma imagen con diferentes escalas y se almacenan dentro de una ruta de carpetas que identifican las referencias de los productos.

Figura 28

```
public static void uploadFile(String producteREF, String fileName, String format){
    try {

        connecta();

        Boolean dirExists;

        client.setFileType(FTPClient.BINARY_FILE_TYPE);

        // Sino existis carpeta la crea
        dirExists = client.changeWorkingDirectory(producteREF);
        if (!dirExists) {
            if (!client.makeDirectory(producteREF)) {
              throw new IOException("Unable to create remote directory '" + producteREF + "'.  error='"
            }
            if (!client.changeWorkingDirectory(producteREF)) {
              throw new IOException("Unable to change into newly created remote directory '" + producte
            }
        }

        InputStream is;
        // Medium
        is = new FileInputStream(Vars.getFILE_DIRECTORY()+producteREF + "/" + fileName + Vars.getMedium
        client.storeFile(fileName + Vars.getMediumSufix() + format, is);
        is.close();

        // Thumbnail
        is = new FileInputStream(Vars.getFILE_DIRECTORY()+producteREF + "/" + fileName + Vars.getThumbn
        client.storeFile(fileName + Vars.getThumbnailSufix() + format, is);
        is.close();
```

Muestra FTPClient

5.3 Tecnología Web

El sistema no tendría sentido sino tuviera una salida web, tanto para la gestión interna del backend como del frontend, por ello es inevitable no hablar de las principales tecnologías web, que existen y que también se explotan en el proyecto, que son HTML5 y CSS3, junto a algún framework de apoyo para la eficiencia del desarrollo gráfico y de comportamiento.

5.3.1 HTML5

HTML5 (HyperText Markup Language, versión 5) es la quinta revisión importante del lenguaje básico HTML.

HTML5 establece una serie de nuevos elementos y atributos que reflejan el uso típico de los sitios web modernos. Algunos de ellos son técnicamente similares a las etiquetas * y *, pero tienen un significado semántico, como por ejemplo *<nav>* (bloque de navegación del sitio web) y *<footer>*. Otros elementos proporcionan nuevas funcionalidades a través de una interfaz estandarizada, como los elementos *<audio>* y *<video>*. Mejora el elemento *<canvas>*, capaz de renderizar elementos 3D en los navegadores más importantes (Firefox, Chrome, Opera, Safari e Internet Explorer).

El proyecto sigue el uso de las nuevas etiquetas y las buenas prácticas que dicta la W3C para la facilitación en la indexación de los buscadores y aplicación de técnicas SEO

5.3.2 CSS3

Hoja de estilo en cascada o CSS es un lenguaje usado para definir la presentación de un documento estructurado escrito en HTML. El World Wide Web Consortium (W3C) es el encargado de formular la especificación de las hojas de estilo que sirven de estándar para los agentes de usuario o navegadores.

La idea que se encuentra detrás del desarrollo de CSS es separar la estructura de un documento de su presentación, haciéndolo más comprensible, más editable y mejor organizado.

5.3.3 JavaScript

JavaScript es un lenguaje de programación interpretado. Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.

Se utiliza principalmente en su forma del lado del cliente (client-side), implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas

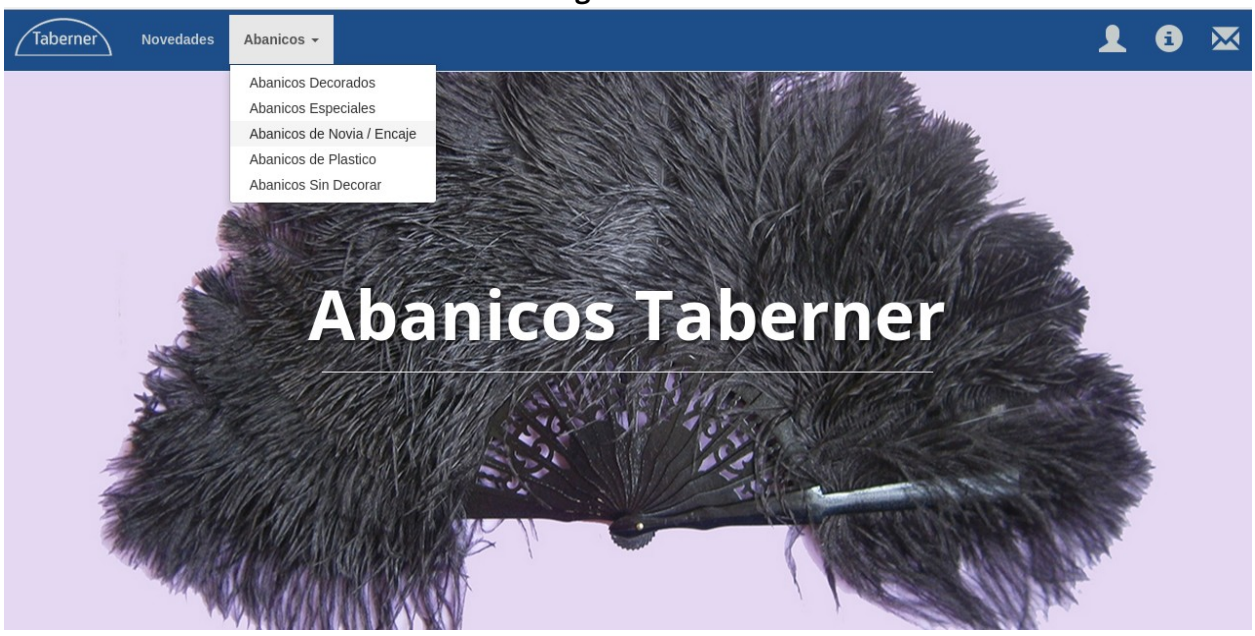
Todos los navegadores modernos interpretan el código JavaScript integrado en las páginas web. Para interactuar con una página web se provee al lenguaje JavaScript de una implementación del Document Object Model (DOM).

5.3.4 Bootstrap

Para unir todas las tecnologías web descritas en los apartados anteriores se da uso de el framework desarrollado por Twitter, Bootstap.

Bootstrap provee de diferentes utilidades en formato CSS y JavaScript, incluso una extensa colección de iconos, para que sean fácilmente utilizadas dentro de las estructuras HTML5. Además es totalmente escalable ajustando el diseño al tamaño de la pantalla del dispositivo. A continuación se presentan algunas de las imágenes del diseño de la web del proyecto

Figura 29

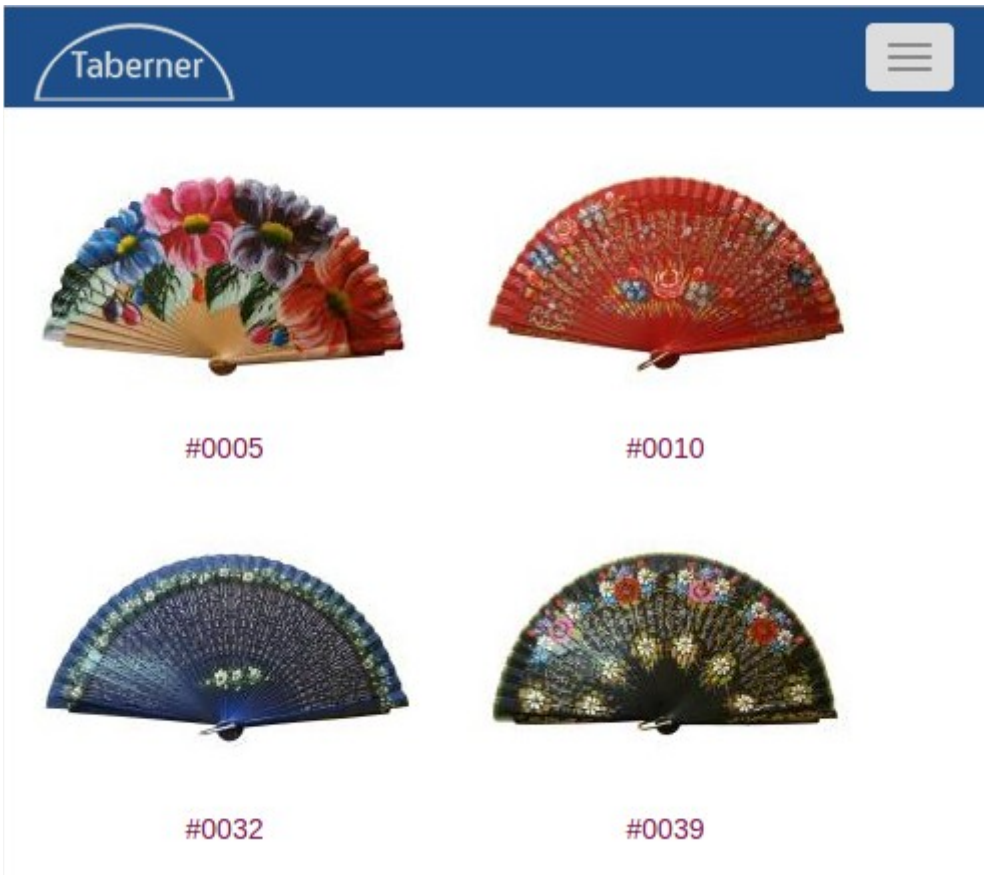


Landing con menú desplegable

Figura 30

Vista con una resolución 1920x1080

Figura 31

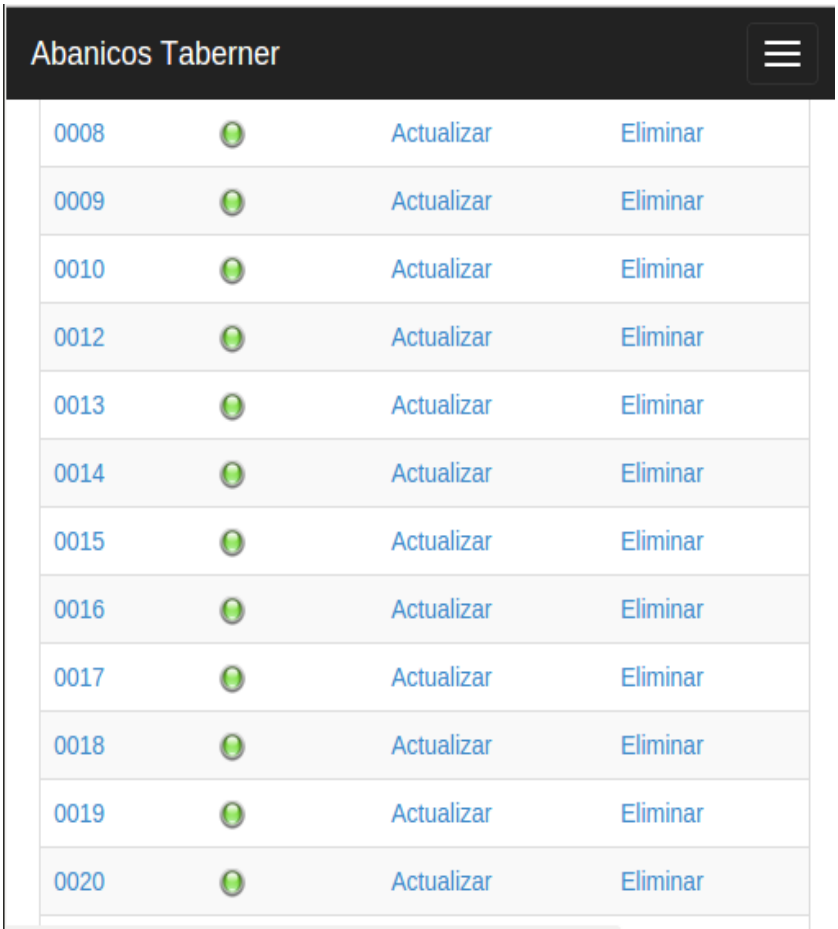


Vista con resolución básica de smartphone

Figura 32

Vista del producto y detalle de colores a partir de capas

Figura 33



Vista para smartphone del listado de referencias para la gestión de productos vía Backend. Véase la completitud del espacio en pantalla y el menú para móvil que existe en la barra superior derecha.

5.4 Glassfish

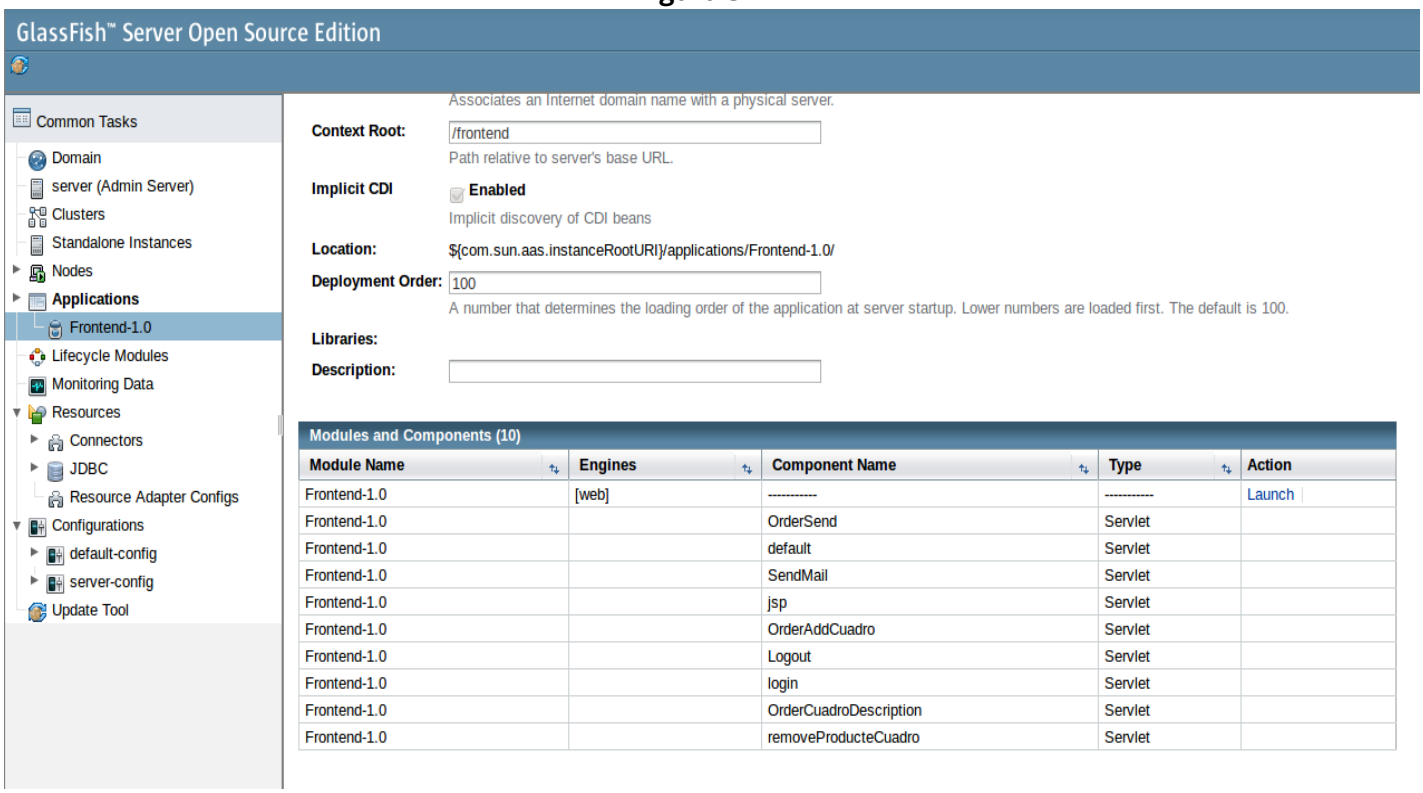
GlassFish es un servidor de aplicaciones de software libre desarrollado por Sun Microsystems, compañía adquirida por Oracle Corporation, que implementa las tecnologías definidas en la plataforma Java EE y permite ejecutar aplicaciones que siguen esta especificación. Es gratuito y de código libre.

GlassFish está basado en el código fuente donado por Sun y Oracle Corporation; este último proporcionó el módulo de persistencia TopLink.1 GlassFish tiene como base al servidor Sun Java System Application Server de Oracle Corporation, un derivado de Apache Tomcat, y que usa un componente adicional llamado Grizzly que usa Java NIO para escalabilidad y velocidad.

Con Glassfish se procesan las aplicaciones del lado del servidor, realizando conexiones bidireccionales y/o unidireccionales y síncronas o asíncronas con el cliente y generando o cediendo una respuesta a partir de un fichero .war.

Pese a todas las bondades que ofrece Glassfish, en este proyecto solo se usa como contenedor para el despliegue de la aplicación.

Figura 34



Muestra de Glassfish

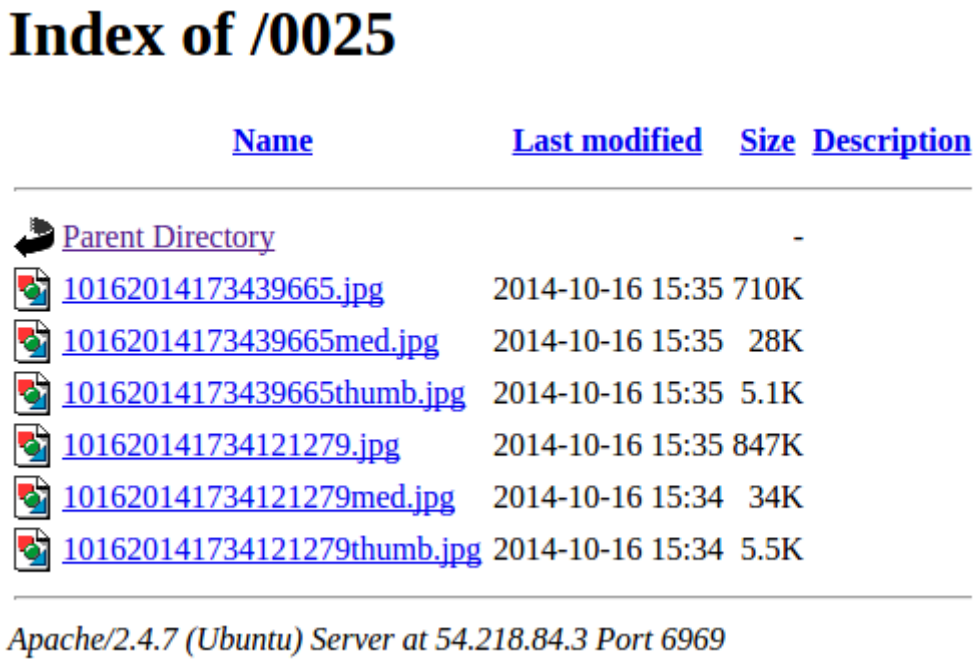
5.5 Apache HTTP Server

El servidor HTTP Apache es un servidor web HTTP de código abierto, multiplataforma, que implementa el protocolo HTTP/1.12 y la noción de sitio virtual.

Apache presenta entre otras características altamente configurables, bases de datos de autenticación y negociado de contenido, pero fue criticado por la falta de una interfaz gráfica que ayude en su configuración. En 2005 fue el servidor empleado en el 70% de los sitios web en el mundo.

El uso que tiene en el proyecto es simplemente como contenedor de imágenes.

Figura 35



Gestión de imágenes en Apache

Para mejorar su seguridad se ha implementado un fichero .htaccess que permite definir diferentes directivas de configuración para cada directorio (con sus respectivos subdirectorios) sin necesidad de editar el archivo de configuración principal de Apache. Un ejemplo del uso de las políticas de .htaccess sería el siguiente

```
<Files ~ "^\.(htaccess|htpasswd)$">
        deny from all
</Files>
Options Indexes
order deny,allow
```

5.6 Gestión

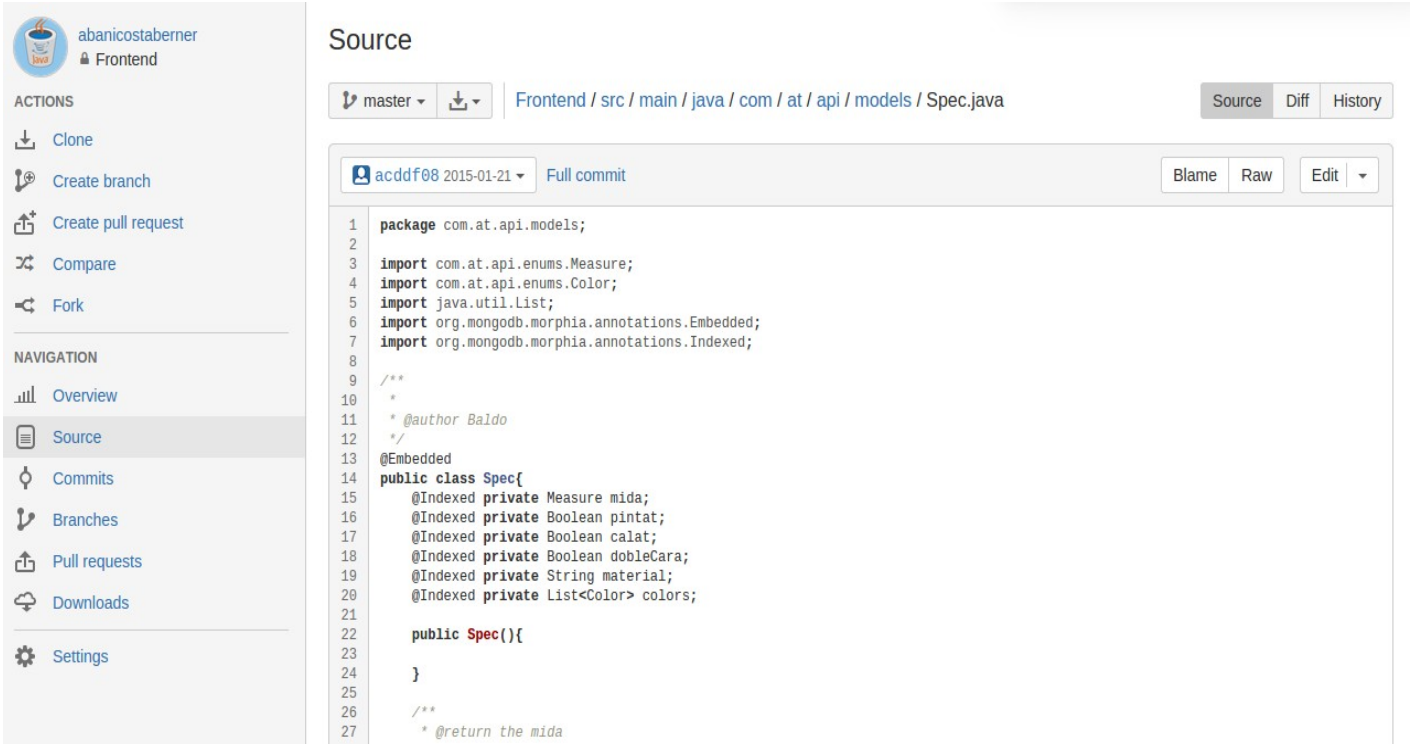
Además de desarrollar el sistema atendiendo a una serie de tecnologías, existen otras que son de vital importancia a la hora de gestionar el proyecto. En concreto se focaliza en tres tipos diferentes de software que facilitan el mantenimiento del código y que son interesantes para llevar en forma de historial la evolución del proyecto.

5.6.1 Git

Git es un controlador de versiones gratuito y de código abierto distribuido para manejar todo, desde pequeñas a grandes proyectos con rapidez y eficiencia. Fue creado pensando en la eficiencia y la confiabilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente.

Con Git se puede versionar fácilmente el código fuente, y registrar en una línea temporal los cambios que van surgiendo, pudiendo retornar a una versión anterior en caso de aparecer algún tipo de error, o para hacer una vuelta atrás y permitir integración continua. En el proyecto se da uso de un repositorio git en local por cada aplicación, y además se respalda con un servidor privado online que ofrece Atlassian en BitBucket.

Figura 36



Muestra de BitBucket

5.6.2 SonarQube

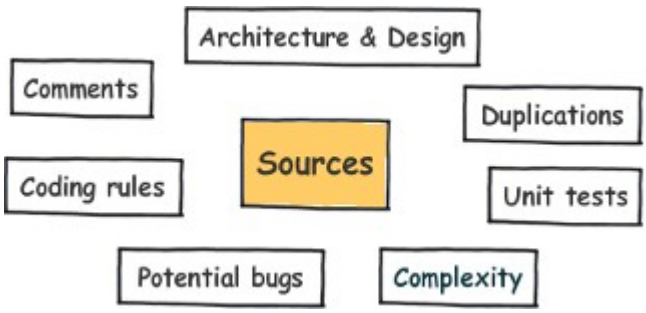
SonarQube es una plataforma para evaluar código fuente. Es software libre y usa diversas herramientas de análisis estático de código fuente como Checkstyle, PMD o FindBugs para obtener métricas que pueden ayudar a mejorar la calidad del código de un programa.

Cuando se trabaja en el mantenimiento del software se producen cambios constantemente en el sistema. El código sufre la manipulación continua debida a mejoras estructurales, refactorización, reparación de -posibles- fallos y programación de nuevas funcionalidades. El descontrol puede ser total por diferentes motivos y el cambio de una línea de código puede ocasionar graves pérdidas. Por ello es necesario mantener un control del proyecto, cumpliendo la especificación en todo momento y que una nueva codificación no influya en la estabilidad del software. Estas tareas pueden llegar a ser muy complejas y costosas tanto económicamente como temporalmente.

SonarQube ayuda a mantener un software de calidad analizando el código en busca de errores o posibles mejoras en él, cubre test unitarios y además guarda la cronología del antes y el después para que sirva como métrica comparativa, pero SonarQube no solo muestra lo que está mal, sino que también ofrece herramientas de gestión de la calidad para ayudar activamente a generar buen código: integración con IDEs, integración de Jenkins, un servidor de integración continua, y herramientas de revisión de código.

SonarQube ofrece cobertura en los 7 axiomas de calidad, es decir; aborda no sólo los errores, sino también reglas de codificación, la cobertura de pruebas, duplicaciones, documentación de la API, la complejidad y la arquitectura. Ningún software parecido actualmente aborda los siete ejes.

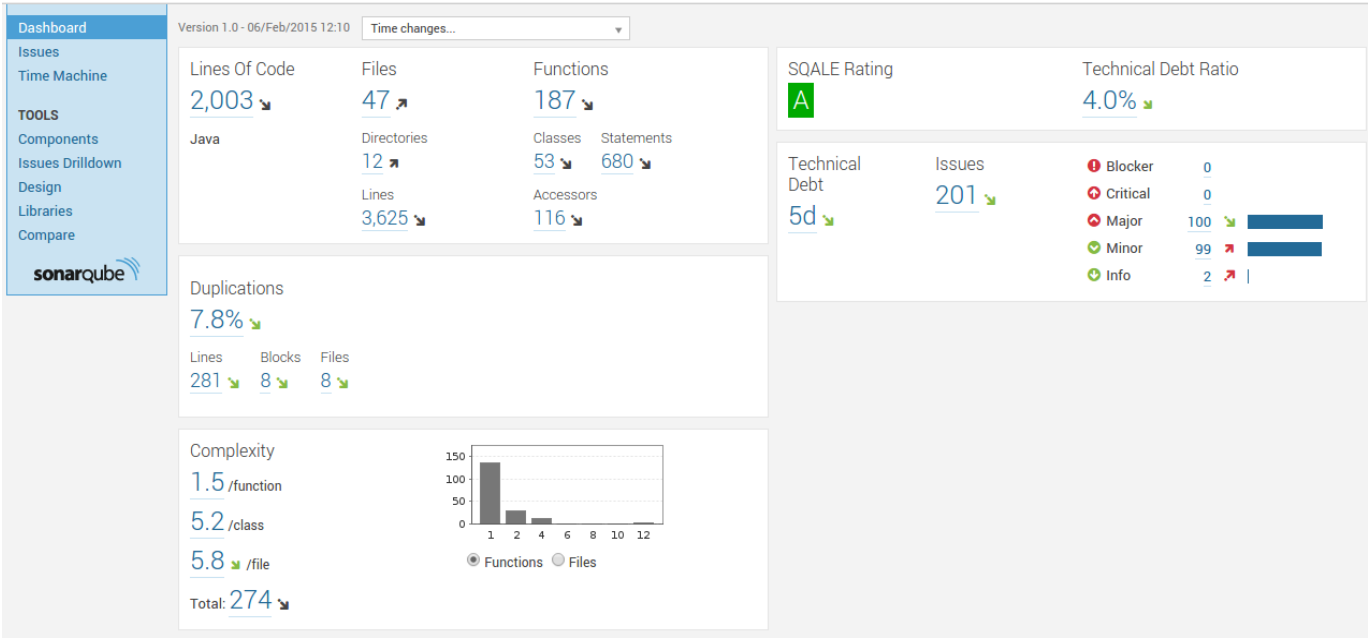
Figura 37



Los 7 axiomas de calidad

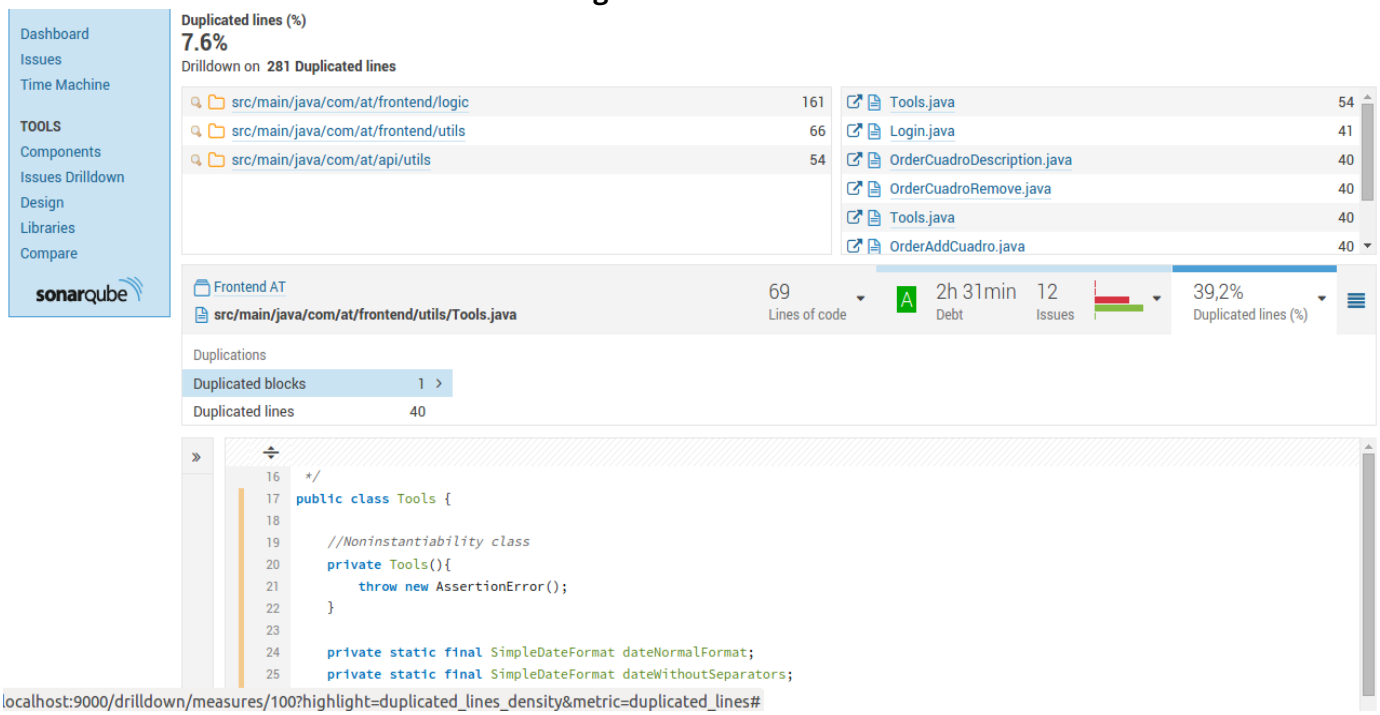
A continuación se muestran algunas capturas de pantalla de SonarQube en referencia al proyecto.

Figura 38



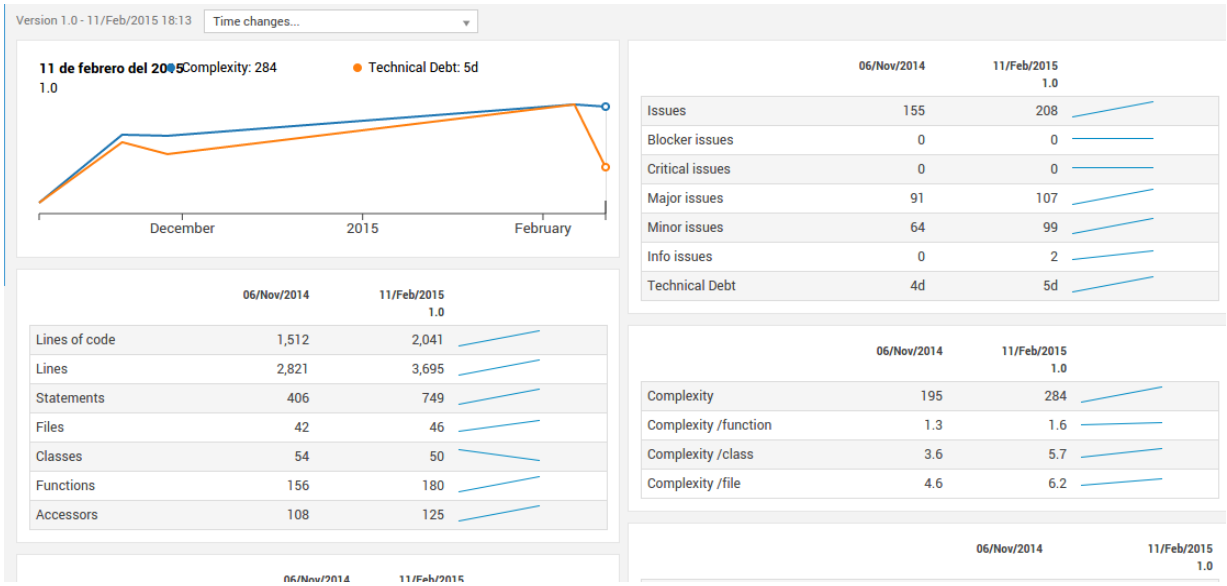
Dashboard de Frontend

Figura 39



Vista de posible duplicación de código en una clase

Figura 40



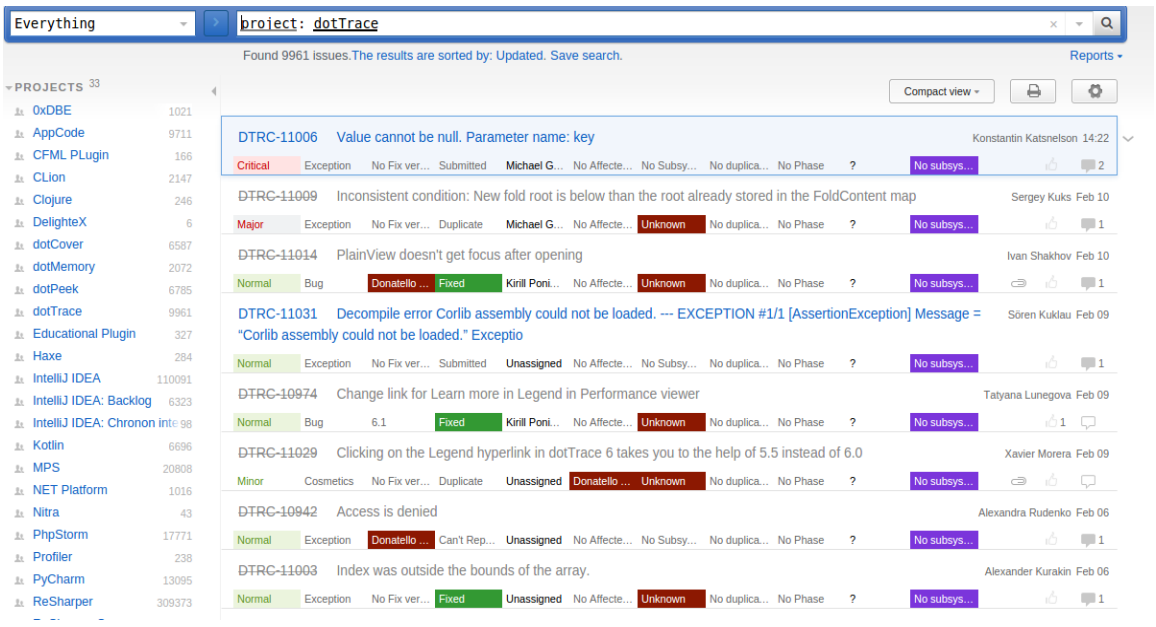
Historial gráfico de los cambios

5.6.3 YouTrack

YouTrack es un software comercial de JetBrains basado en navegador y que trabaja como un bug tracker, con lo que ayuda al seguimiento de problemas del sistema y software de gestión de proyectos.

Se centra en la gestión y búsqueda de problemas a partir de consultas con auto-realización, manipulación de los diferentes estados en los que pasa cada bug o tarea y total personalización en el conjunto de atributos de emisión y la creación de flujos de trabajo personalizados.

Figura 41



Capítulo 6. Testing

El testing consiste en establecer pruebas como factor crítico para determinar la calidad del software. Esto permite realizar una evaluación de algún aspecto en concreto del sistema: corrección, robustez, eficiencia, etcétera. De manera que es posible verificar si se está construyendo correctamente el producto y evaluar si el sistema hace lo que el cliente realmente quiere que haga. Es decir, si satisface correctamente sus necesidades.

6.1 Principios básicos

1. Testing es el proceso de ejecutar un componente software utilizando un conjunto básico de casos de prueba, con la intención de revelar defectos, y evaluar la calidad.

2. Cuando el objetivo de la prueba es detectar defectos, entonces un buen caso de pruebas será aquel con una mayor probabilidad de detectar un defecto todavía no detectado.

3. Los resultados de las pruebas deberían ser meticulosamente inspeccionados.

4. Un caso de pruebas tiene que contener los resultados esperados.

5. Los casos de prueba deberían ser definidos tanto para condiciones de entrada válidas como no válidas.

6. La probabilidad de la existencia adicional de defectos en un componente software es proporcional al número de defectos ya encontrados en ese componente.

7. Las pruebas deberían ser llevadas a cabo por un grupo que sea independiente del grupo de desarrollo.

8. Las pruebas tienen que ser repetibles y reutilizables.

9. Las pruebas deben ser planificadas.

10. Las actividades de pruebas deberían estar integradas con el resto del ciclo de vida.

11. Testing es una actividad creativa y desafiante.

Cumpliendo tal y como sea posible los once principios básicos dictados para el testing, se han llevado a cabo una serie de pruebas para la correcta validación y verificación del proyecto.

6.2 Pruebas de aceptación

Las pruebas de aceptación tienen como propósito demostrar al cliente el cumplimiento de un requisito del software en su propio entorno de explotación para determinar si lo acepta como está o no.

En el desarrollo del proyecto, al seguir metodologías de desarrollo ágiles y tener una directa y asidua comunicación con el cliente, las pruebas de aceptación se han ido demostrando conforme aparecía una nueva funcionalidad externa, o grupo de funcionalidades que estuvieran en común. Dependiendo de la importancia del cumplimiento del requisito a validar, la prueba de aceptación se llevaba a cabo de inmediato o se mantenía a la espera de crear un grupo de pruebas a modo de sprint.

La prueba de aceptación primero se demuestra ejecutando el sistema en un servidor *offline* que simula el entorno real, de manera que no se vea afectado en ningún caso el que está en producción. Se muestran al cliente las nuevas funcionalidades, o correcciones. Si fuera necesario, se forma y tutorea a la persona encargada de administrar el sistema. Una vez validado se prosigue a la integración y se vuelve a demostrar la prueba de aceptación esta vez en producción.

Cuando el cambio pertenece a la parte del sistema que se muestra al cliente de la empresa, se deben cerciorar aspectos de usabilidad y una aceptación lo más globalizada. Para ello se toman distintos posibles usuarios para que den su opinión o demuestren su capacidad de interacción con respecto a la nueva funcionalidad, nuevos cambios de imagen, etcétera. Además, cuando el producto está lanzado, la empresa acepta constantemente *feedback* directo de los clientes, por lo que las pruebas de aceptación se siguen revisando aún con el proyecto desplegado.

Si no se ha validado correctamente alguna prueba de aceptación, se analiza el caso y se encuentra una solución al problema, generalmente se puede repara o mejorar a partir de código, pero puede ocurrir que la incapacidad de cumplimiento sea estructural, incluso que sea fallo del propio requisito.

6.3 Pruebas de sistema

En el momento en el que el software, está listo para ejecutarse hay que mantener el control en la buena respuesta que tendrá cuando se ejecute en el entorno de producción. En este caso se han tenido en cuenta pruebas más concretas abajo descritas en orden de ejecución.

- Verificación de pruebas de bajo nivel.
- Conexión correcta con la Base de datos real.
- Despliegue correcto de la aplicación en el servidor.
- Comprobación de las nuevas funcionalidades.
- Verificación de otros servicios que afectan. (eMail, FTP, ...)

6.4 Pruebas unitarias

Las pruebas unitaria son una forma de comprobar el correcto funcionamiento de un módulo de código. Esto sirve para asegurar que cada uno de los módulos funcione correctamente por separado.

En el proyecto se ha intentado cubrir la mayor parte de los métodos y clases con este tipo de prueba para permitir la integración continua conforme avanza el desarrollo del producto. Sobretodo en métodos que se reutilizan en multitud de ocasiones para evaluar como se enfrenta a diferentes situaciones posibles.

Ejemplo

Figura 42

```
/**
 *
 * @author Baldo Taberner
 */
public class Lists {

    //Noninstantiability class
    private Lists(){
        throw new AssertionError();
    }

    /**
    * If List<> is null return a EmptyList
    * @param <T>
    * @param iterable
    * @return List or EmptyList
    */
    public static <T> Iterable<T> emptyIfNull(Iterable<T> iterable) {
        return iterable == null ? Collections.<T>emptyList() : iterable;
    }
```

Método a testear

Método que comprueba si una lista de cualquier tipo, es una lista con elementos. Sino tiene ninguno, en vez de devolver null, devuelve la lista como vacía, y así evita errores.

Figura 43

```
@Test
public void testEmptyIfNull() {
    System.out.println("emptyIfNull");
    List list = new ArrayList();
    Iterable expResult = list;
    Iterable result = Lists.emptyIfNull(list);
    assertNotNull(result);
    assertEquals(expResult, result);

        list.add("a");
        list.add("b");
    expResult = list;
    assertEquals(expResult, result);
}
```

Test unitario del método

6.5 Pruebas de integración

Pruebas de integración son aquellas que se realizan una vez que se han aprobado las pruebas unitarias, y consisten en realizar pruebas para verificar que un gran conjunto de partes de software funcionan juntos.

En el proyecto, entran en comunicación diversas clases que se especializan en una funcionalidad concreta, si se da por comprobado la buena funcionalidad de estas por separado. Hay momentos en los que se puede comprobar de manera unitaria que una clase controlador, que trabaja con muchas más clases por debajo, funciona correctamente. Es un indicador básico de que la integración entre clases es correcta.

Ejemplo

El objetivo es que el sistema puede a partir de una identificación del cliente leer el carrito de la compra que está gestionando, y en caso de que no exista, crear uno nuevo. Para ello se da uso de una clase controlador que gestiona el proceso.

Figura 44

```
public static Cart read(String id){
    BasicDBObject doc = CartCRUDL.read(id);

    if(doc == null){
        if(CartCRUDL.create(id))
            doc = CartCRUDL.read(id);
        else
            return null;
    }

    Cart cart = CartMapper.map(doc);
    return cart;
}
```

Método de CartController

Este método devuelve el objeto Cart a partir de un identificador, en caso de no corresponder a ningún Cart en la base de datos, devuelve un objeto null.

El método anterior internamente trabaja con dos clases más, la clase que se encarga de crear las consultas a la base de datos y la clase que mapea en objetos la respuesta de la consulta. Los métodos que entran en el proceso se presentan a continuación.

Figura 45

```
protected static boolean create(String idClient){
    try{
        ObjectId oid = new ObjectId(idClient);

        BasicDBObject obj = new BasicDBObject("_id", oid);
        collection.insert(obj);

        modified(idClient);

        return true;
    }catch (Exception e){
        System.err.println(" ERROR AL CREAR CART"+ e);
        return false;
    }
}

protected static BasicDBObject read(String idClient){
    try{
        ObjectId oid = new ObjectId(idClient);

        BasicDBObject query = new BasicDBObject();
            query.put("_id", oid);

        BasicDBObject result = (BasicDBObject)collection.findOne(query);

        return result;

    }catch (Exception e){
        return null;
    }
}
```

Métodos de CartCRUDL

Figura 46

```
protected static Cart map(BasicDBObject doc){
    Cart cart = new Cart();

    if(doc.containsField("_id"))
        cart.setId(doc.getObjectId("_id"));

    if(doc.containsField("sent"))
        cart.setSent(doc.getBoolean("sent"));

    if(doc.containsField("lastModified"))
        cart.setLastModified(doc.getDate("lastModified"));

    if(doc.containsField("items")){

        List<Item> items = new ArrayList();

        BasicDBList listDBItems = (BasicDBList)doc.get("items");

        List<BasicDBObject> listItems = Arrays.asList(listDBItems.toArray(new BasicDBObject[0]));

        for(BasicDBObject item : listItems)
            items.add(mapItem(item));

        cart.setItems(items);
    }

    return cart;
}
```

Método de CartMapper

Figura 47

```
protected static Item mapItem(BasicDBObject docItem){
    Item item = new Item();

    if(docItem.containsField("ref"))
        item.setRef(docItem.getString("ref"));

    if(docItem.containsField("price"))
        item.setPrice(docItem.getDouble("price"));

    if(docItem.containsField("quantity"))
        item.setQuantity(docItem.getInt("quantity"));

    if(docItem.containsField("details"))
        item.setDetails(docItem.getString("details"));

    return item;
}
```

Método de mapeo para elemento item dentro de Cart

A partir de este punto, se crea una prueba unitaria al método controlador de lectura, siguiendo la tecnología jUnit con las diferentes sucesos.

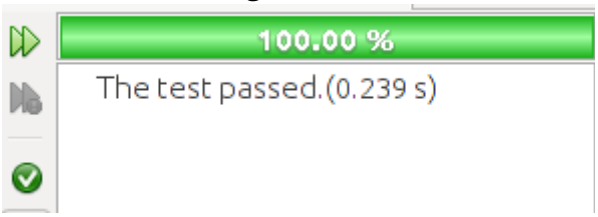
Figura 48

```
/**
 * Test of read method, of class CartController.
 */
@Test
public void testRead() {
    System.out.println("read");
    String id = "5460e3ca44aecd4cfc90a8c4"; //Correcto en DB
    Cart result = CartController.read(id);
    assertEquals(id, result.getId().toString());

    id = "5460e3ca44aecd4cfc90a800"; //Incorrecto en DB
    result = CartController.read(id);
    assertNull(result);

    id = null;
    result = CartController.read(id);
    assertNull(result);
}
```

Prueba unitaria

En respuesta final se obtiene la verificación de la prueba unitaria que hace partícipe la integración de dichas clases.

Figura 49



Resultado del test

6.6 Pruebas de rendimiento

Las pruebas de rendimiento son las pruebas que se realizan, desde una perspectiva, para determinar lo rápido que realiza una tarea un sistema en condiciones particulares de trabajo. También puede servir para validar y verificar otros atributos de la calidad del sistema, tales como la escalabilidad, fiabilidad y uso de los recursos.

Estas pruebas se han establecido a partir de tiempos de respuesta, y en base al estudio de la arquitectura del sistema. En estos casos se ha modificado estructuralmente tanto la base de datos como la forma que tienen las clases de trabajar conjuntamente. Así como la gestión y búsqueda en distintas estructuras de datos. Tanto para mejorar el rendimiento como para permitir la escalabilidad.

Por otro lado las pruebas de rendimiento con foco en el estrés del sistema no se han establecido porque no se han creído necesarias, ya que la infraestructura que soporta el proyecto está más que adaptada a las presiones de carga o al requerimiento del sistema, en casos concretos. De hecho un documento en la base de datos tiene un tamaño máximo de 16 megabytes, tamaño de sobra para albergar las características de un producto o un pedido en texto plano.

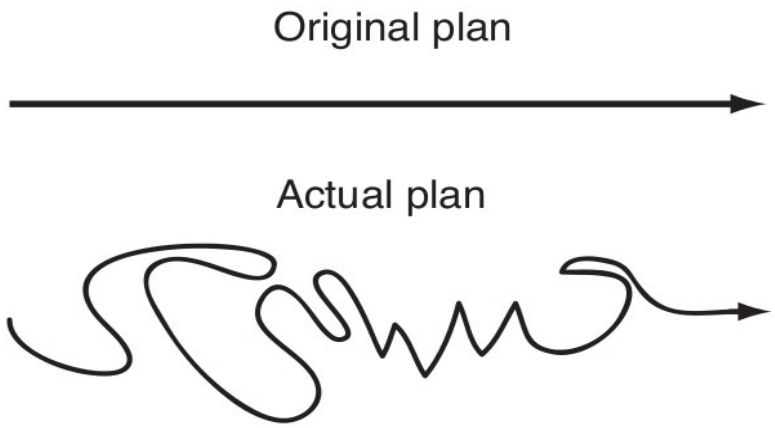
Capítulo 7. Problemas y Soluciones

Según ha ido transcurriendo el desarrollo del proyecto, en sus distintas fases han ido apareciendo problemas a los que se les ha ido dando distintas soluciones. Este apartado se centra en la descripción de algunos de ellos.

7.1 De la planificación al hecho, hay un trecho

Cuando se crea una idea en mente, esta funciona perfectamente en nuestras cabezas, hace justo lo que se quiere, ni más ni menos, pero conforme se ahonda en ella se van descubriendo algunos obstáculos o dificultades. Es por ello por lo que se necesita una buena ingeniería y una buena planificación a seguir. Se forman reuniones, se crean diagramas, se hace una investigación exhaustiva, estudiando diferentes perspectivas y se toman decisiones con fechas, presupuestos, y demás valores que se generan tras una buena planificación.

Figura 50



Planificación

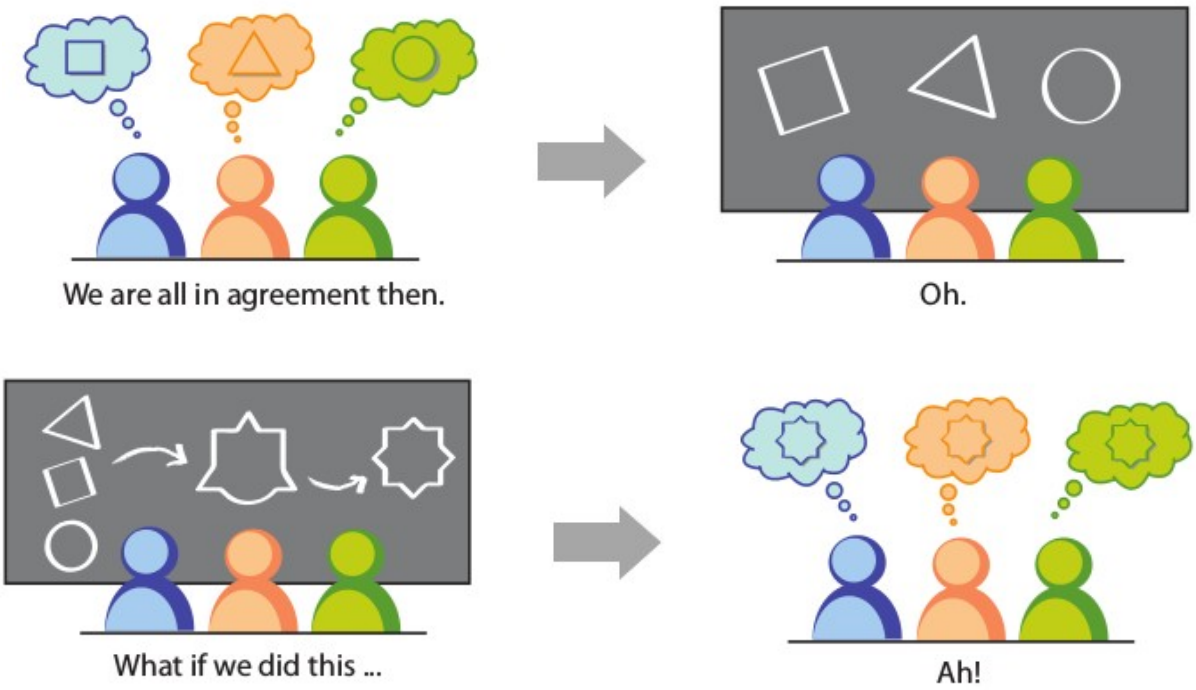
Al final, se crea un plan rectilíneo con un camino directo y realista a los objetivos que se pretenden conseguir, pero según empieza el desarrollo aparecen imprevistos continuos, reestructuraciones en las bases del sistema, cambios de requisitos que afectan totalmente al proyecto, que empieza a tambalearse y a irse en ocasiones más hacia atrás que hacia adelante. El problema no es el plan, que siempre podría ser mejor, el problema es la realidad que es impredecible y la solución que más se acerca a allanar el camino es seguir una metodología flexible, que se pueda adaptar a los vaivenes, a los cambios continuos e impredecibles. En este proyecto el uso de las

metodologías ágiles ha permitido acolchar los distintos problemas que han ido aconteciendo. La metodología basada en Scrumban ha permitido crear una planificación necesaria, un formato a seguir adecuado y controlado que junto a otras propiedades de XP; como los análisis de calidad, refactoring o el trato de fallos inmediato permite a la larga, avanzar en positivo. En caso de que se deba echar la vista atrás y deshacer parte de lo desarrollado, entra en valor el controlador de versiones Git, que ha sido crucial para retomar código funcional o de base a guía de como funcionaba anteriormente.

7.2 Donde dije digo, digo Diego

Durante el desarrollo del proyecto se han tomado diversas reuniones con el cliente, para llegar a un consenso respecto a requisitos concretos o qué debería hacer el sistema en determinadas ocasiones, las opiniones por parte de los *stakeholders* que han participado en las reuniones eran distintas. Argumentadas en diferentes razones, pero que a veces distaban mucho entre ellas. La labor del ingeniero de software ha sido crear una idea que satisfaga dentro de la realidad a cada uno de ellos.

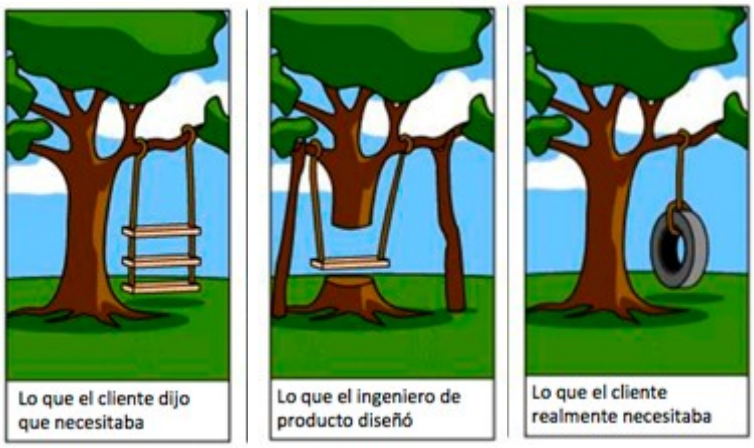
Figura 51



Mezcla de ideas

Esta forma de trabajar es la adecuada, pero el principal problema que se ha tenido es por el desconocimiento del mismo cliente. Cuando el cliente no sabe exactamente lo que quiere, porque no sabe tampoco lo que hay, ni como afrontarlo se crea un grave problema. Aquí la labor del ingeniero es ser honesto y explicar de la mejor forma posible su visión, intentar entender el modelo de negocio del cliente y poder mostrar una solución adecuada. Esta parte es realmente complicada, porque existe un desconocimiento por las dos partes y la comunicación por muy fuerte que sea, suele tener claras complicaciones, porque el cliente no sabe lo que quiere y dice querer algo que hace qué, y se termina por llegar a un acuerdo, que él mismo rechaza o acepta, pero que muy probablemente volverá a cambiar.

Figura 52



Fallo de comunicación

En este caso sucedió, entre otras situaciones, que una parte de la empresa quería dos precios por producto, uno para minoristas y otro para mayoristas, pero la parte más directiva de la empresa pretendía tener solamente uno para mayoristas y oculto y sin acceso para los minoristas. Esta situación trastoca la estructura del sistema. En definitiva, se tomó la razón de la directiva como la adecuada a seguir, simplemente por razones de negocio, pero conforme se avanzó en el desarrollo, la dirección tuvo que ir hacia atrás, y cambiar totalmente los requisitos, porque ahora quería que hubieran dos precios y se pudieran hacer pedidos tanto para minoristas como para mayoristas siguiendo diferentes procedimientos.

7.3 Mejorando arquitectura

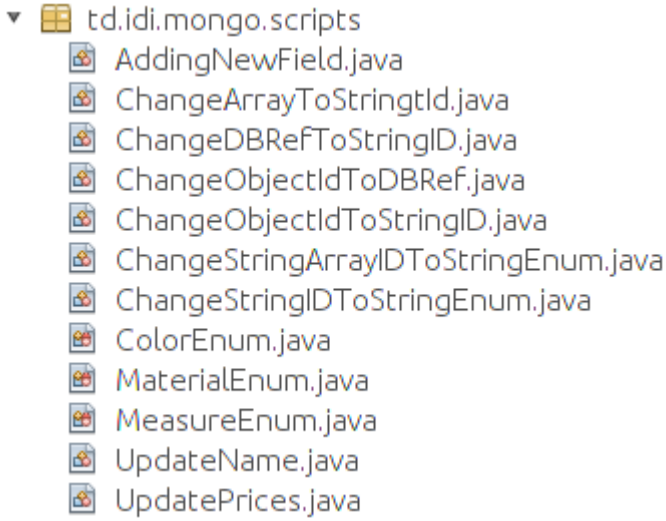
El sistema se diseñó desde un principio con una arquitectura cliente / servidor y aplicando el patrón arquitectónico modelo-vista-controlador. Con la experiencia se ha ido modulando por familias, obteniendo un código y una organización más legible. Además, al estar modulado, se ha minimizado la accesibilidad entre clases de forma que se crea una mayor seguridad en el sistema. Por ejemplo las clases CRUDL y Mapper son sólo accesibles desde la clase de su Controller y de ninguna más.

En las clases de control en su origen se aplicó el patrón Singleton. Con un poco más de investigación e intentando sacar la máxima eficiencia, en java es posible crear clases Enum que abstraen este patrón y lo mejora porque están provistas de serialización y proporcionan seguridad contra ataques de reflexión [**Joshua Bloch @ Effective Java**] Aún así, se terminó por desechar este tipo de patrón y se aplicó finalmente en estas clases el patrón *noninstantiability* con métodos estáticos porque no necesitaban tener ni siquiera una instancia.

7.4 Mejorando DB

La falta de experiencia en bases de datos noSQL ha significado reestructurar en multitud de ocasiones el esquema, incluso cuando ya estaba en producción. Se ha necesito programar scripts para los cambios de estructura de los datos.

Figura 53



Scripts para DB

Con conocimiento en diseño de bases de datos SQL, en ciertas ocasiones, se creaban relaciones entre documentos innecesarias. También se pecó de lo contrario. Se creaban subdocumentos embebidos con el problema de que los datos fueran inconsistentes.

MongoDB ofrece mucha flexibilidad con la estructura de datos, por ello han existido multitud de cambios con la intención de acercarse al origen de los propios datos y facilitar la consulta en el sistema. Por ejemplo la gestión de los colores de los productos contemplan datos propios del color y en un principio se creó una colección de documentos de colores con sus propiedades, que estaban relacionados con los productos. La respuesta a las consultas era ineficiente, sobretodo al editar los productos y se reestructuró embebiendo los colores y sus propiedades dentro de cada documento. De esta forma se aligeraba el tiempo de respuesta, pero creaba datos duplicados, y esta modificación se hizo al principio del desarrollo, cuando la base de datos comenzaba a engordar con productos reales y tuvo que hacerse una modificación de las propiedades de los colores hubo un gran problema de inconsistencia, por lo que se reestructuró de nuevo y pasaron a estar fuera de la DB y dentro del sistema, porque no eran datos naturales de la empresa, sino externos. Por lo que en la base de datos, dentro de cada producto se almacena una lista de los colores como una cadena de texto de literales y dentro del sistema pertenecen a un enum con sus propiedades.

La complicación del diseño y las posibilidades pueden ser diversas. En el caso de la creación de tipos con subtipos para la ordenación de productos tiene su complicación. Después de un estudio se eligieron diferente alternativas.

La primera opción era la típica entidad-relación, creando documentos para tipos que embebian referencia a sus subtipos, pero se generaban demasiadas consultas a la hora de listar.

La segunda opción era totalmente contraria. Anidaba dentro de cada tipo los subtipos con toda la información de estos, incluso los datos de los productos que se utilizaban solamente para el listado por subtipos. De esta forma con una sola consulta se recolectaba toda la información necesaria. El problema estaba en el tamaño que se generaba en un documento y el mantenimiento del mismo complicaba demasiado la programación.

La tercera opción era una fusión de las dos anteriores y tenia su explicación porque se realizaban dos consultas diferentes que atendían al sistema. Una para listar simplemente los subtipos de un tipo y otra para listar los productos de los subtipos. Por lo que la propuesta contemplaba dos tipos de documentos. Uno que definía el tipo y sus subtipos con información básica para listarlos en la interfaz y también con

relación a el otro tipo de documentos que formaban los subtipos con la información a mostrar de los productos que contenía. Más consultas que la segunda opción, pero mucho más eficiente y más fácil de mantener.

Como aún existían dudas al respecto, sobre qué diseño era mejor, se llevó como pregunta a expertos en MongoDB, aportando la segunda y la tercera opción. La respuesta se muestra en la Figura 54.

Figura 54

The best solution in MongoDB is the first one you show since it is a non relational database. About your questions, you don't have to worry, you can do everything you ask. You just need to know how to do it. About your questions:

"Ok I know, I can use projections, but ***Is it possible to use projections with query? Could I project the products of just one subtype?****"*

Yes, but not in the normal way. One solution you could try is to filter a subtype and then project only the products, like these:

db.products.find({"name": "TypeName", "subtypes.name": "SubtypeName"},{"subtypes.products": 1})

The problem here is that this will give you a list with all the subtypes of type "TypeName" and their list of products. So **this is wrong**.

You need to use $elemMatch in the project to retrieve the list of products you want:

db.products.find({"name": "TypeName", "subtypes.name": "SubtypeName"}, {"subtypes": {$elemMatch: {"name": "SubtypeName"}}, "subtypes.products": 1 })

Note that $elemMatch is used to limit an array field from the query results and **only returns the first element matching**.

In the use case you exposed I think this should be the solution.

Respuesta de expertos en MongoDB

De todas formas el problema estaba en mantener la integridad de los datos y se necesitaba complicar demasiado la programación y tener un control en cada actualización de los datos para que fueran actualizándose cada una de las repeticiones. Al final se optó por la solución de “Categoría jerárquica” que propone Rick Copeland en su libro de patrones para MongoDB que simplifica la definición de tipo como subtipo, dentro de una referencia al padre, y cada producto solamente contiene un tipo sin necesidad de repetir datos. Esta forma es la más sencilla y la que al final se ha mantenido como definitiva.

7.5 Mapear o no mapear

Con experiencia en programación orientada a objetos y bases de datos SQL, la necesidad de mapear en objetos los datos era fundamental para el desarrollo correcto del sistema. La especificación oficial de Java incorpora el mapeo con bases de datos SQL gracias a JPA (Java Persistence API), sin embargo no ofrece nada para el mapeo de bases de datos NoSQL, por lo que se tuvo que buscar un framework de terceros para ello y se contó con Morphia, que es fácil de configurar y eficiente en su mapeo, porque está basado en notaciones y las consultas sencillas se generan con pocas líneas de código. El problema de usar frameworks es que se delega el control en este, lo que permite abstraerse y centrarse en otras partes del producto. Morphia funciona muy bien en consultas sencillas, pero en caso de necesitar una consulta más compleja se complica la lógica y termina siendo más sencillo crear consultas con la API oficial de MongoDB.

Como el mapeo se genera de manera automática, el proceso es transparente al desarrollador, lo que impide trabajar con los datos para mapear según condiciones. Por ejemplo, si el producto contempla dos tipos de precios, según el cliente, lo correcto sería mapear el objeto con solo el precio que el cliente tiene acceso y esto se decide en la consulta y en el posterior mapeo. No tiene sentido mantener dos precios en el objeto, aunque solo quede visible uno.

En la parte del Frontend la gestión de los datos es mínima, pues el uso se centra en la lectura de estos y en ocasiones es ineficiente mantener el objeto mapeado conforme la base de datos, pues se consume memoria en *heap* que nunca será necesaria. Aún así el paradigma orientado a objetos facilita el uso de los mismos objetos, por lo que sí que es necesario un mapeo por mínimo que sea.

En solución al problema, se crea sobretodo en la parte Frontend, un mapeo no automático sin frameworks. Así se logra una mayor eficiencia y un mapeo más elaborado y conciso de los datos que se van a utilizar.

Capítulo 8. Conclusiones

En la actualidad la tecnología evoluciona a diario. El software debe adaptarse a las nuevas exigencias del mercado, y debe de estar preparado para afrontar una más que probable escalabilidad. La evolución es constante y por ello se necesita una metodología de trabajo flexible que permita un software adaptable y cambiable. El software ha de ser ajustable a la realidad del momento.

En este proyecto se han empleado las técnicas necesarias para permitir todo el vayvén tecnológico, para crear una mejora continua sin necesidad de dar grandes pasos atrás y dando funcionabilidad desde los primeros momentos del desarrollo. Es por ello, por lo que se considera extremadamente obligatorio seguir unas pautas ordenadas en el conjunto de métodos que se precisan para la gestión y el desarrollo del sistema. La creación de documentación puede llegar a ser engorrosa y en muchos casos innecesaria, pero la ausencia de esta es fatal para el buen hacer de cualquier analista o desarrollador, que aún sin en ocasiones necesitarla en la vida del proyecto, formará parte del histórico para futuros. Pues durante los primeros momentos del desarrollo la inexperiencia ha sido el principal problema, tanto en aspectos de gestión como tecnológicos y estructurales.

Finalmente el sistema desarrollado forma parte de la PYME real, que la ha adoptado con agrado y soporta una gran responsabilidad para la misma. Sin embargo el proyecto no queda inconcluso, porque los requisitos continúan cambiando, pretendiendo mayor usabilidad, portabilidad, eficiencia y más funcionalidades. Este proyecto pretende ser el núcleo de toda la información empresarial que, a modo de puzzle, se irá ampliando en nuevos frentes.

Referencias

http://www.agilemanifesto.org/
http://leansoftwareengineering.com/ksse/scrum-ban/
http://www.extremeprogramming.org/
http://martinfowler.com/articles/continuousIntegration.html
http://www.ciges.net/comparativa-mongodb-vs-mysql-conclusiones-2-parte
http://robomongo.org/
http://es.wikipedia.org/wiki/MongoDB
http://www.mongodb.com/
http://www.genbetadev.com/bases-de-datos/mongodb-que-es-como-funciona-y-cuando-podemos-usarlo-o-no
http://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n)
http://www.java.com
http://www.javahispano.org/
http://maven.apache.org/
http://es.wikipedia.org/wiki/Maven
https://platform.netbeans.org/tutorials/nbm-maven-quickstart.html
http://junit.org/
https://netbeans.org/kb/docs/java/junit-intro.html
http://es.wikipedia.org/wiki/JUnit
http://docs.mongodb.org/ecosystem/tutorial/getting-started-with-java-driver/
https://github.com/mongodb/morphia/wiki
http://es.wikipedia.org/wiki/JavaServer_Pages
http://www.jsptut.com/
http://www.tutorialspoint.com/jsp/
https://pdfbox.apache.org/
http://commons.apache.org/proper/commons-net/apidocs/org/apache/commons/net/ftp/FTPClient.html
http://www.codejava.net/java-se/networking/ftp/java-ftp-file-upload-tutorial-and-example
http://lineadecodigo.com/java/conectarse-a-un-ftp-con-java/
http://www.w3.org/TR/html5/
http://es.wikipedia.org/wiki/HTML5
http://www.w3schools.com/css/css3_intro.asp
http://www.css3.info/
http://www.maestrosdelweb.com/css-3-las-nuevas-propiedades/
http://es.wikipedia.org/wiki/JavaScript
http://www.w3schools.com/js/
http://www.desarrolloweb.com/javascript/
http://getbootstrap.com/

http://librosweb.es/bootstrap_3/
http://www.w3schools.com/bootstrap/
http://es.wikipedia.org/wiki/Servidor_web
https://glassfish.java.net
https://glassfish.java.net/es/downloads/v2.1.1-final.html
http://httpd.apache.org/
http://es.wikipedia.org/wiki/Servidor_HTTP_Apache
http://es.wikipedia.org/wiki/.htaccess
http://www.htaccesseditor.com/es.shtml
http://git-scm.com/
http://es.wikipedia.org/wiki/Git
http://www.sonarqube.org/
Baldo Taberner, Domingo Casarrubio, Enrique Carrillo – SonarQube (2014)
https://www.jetbrains.com/youtrack/
https://youtrack.jetbrains.com/issues/IDEA
http://en.wikipedia.org/wiki/YouTrack
http://es.wikipedia.org/wiki/Pruebas_de_software
http://en.wikipedia.org/wiki/Software_testing
https://www.linkedin.com/groups/Qu%C3%A9-es-Prueba-Aceptaci%C3%B3n-3636186.S.48805747
DSIC UPV - Tema 4 Pruebas. Mantenimiento y Evolución del Software
http://es.slideshare.net/abnergerardo/pruebas-de-sistemas-y-aceptacion-23663195
http://www.javiergarzas.com/2014/07/tipos-de-pruebas-10-min.htm
http://es.wikipedia.org/wiki/Prueba_unitaria
http://es.slideshare.net/ae_bm/slides-27357466
Jonathan Rasmusson - The Agile Samuray (The Pragmatic Programmers - 2010)
Rick Copeland - Mongodb applied design patterns (OReilly-2013)