SVM Logística			
Documento	Memoria Trabajo fin de Carrera	Versión	1.1
Fecha de creación	16/01/11	Fecha de última versión	

Trabajo Fin de Carrera

Memoria del Proyecto

UOC

I.T. Informática de Gestión

Curso 2010-2011

Alumno: José María González Vázquez

Consultor: Oscar Escudero Sánchez

Índice de contenido

Dedicatoria

Dedico este trabajo de fin de carrera sobre todo a mi madre. Todas las veces que me has dicho que estudie al final han servido para algo.

También quiero dedicárselo a mi novia y futura esposa por animarme siempre que estaba desanimado y apoyarme siempre que lo he necesitado. Espero corresponderte siempre igual.

También quiero dedicarle el trabajo a mi primo Antonio. Su admiración hace que quiera ser mejor persona.

Gracias a todos.

1. Introducción

1.1. Resumen

Como trabajo de fin de carrera hemos realizado el análisis, diseño e implementación de un pequeño gestor de contenidos para una empresa dedicada a la venta de productos alimenticios. Este documento es la memoria resultante.

La idea principal es la de que, el usuario de la aplicación de administración, pueda gestionar los distintos productos de la aplicación y asociarlos a diversos canales. De esta manera el usuario podrá publicar y despublicar contenidos en la Web sin necesidad de retocar el código HTML de la misma. La ventaja principal es que eliminamos la necesidad de contar con alguien especializado en la creación de contenidos Web para modificar una página porque un distribuidor haya cambiado un producto.

La aplicación incluye la gestión de productos, la gestión de canales, la gestión de proveedores y clientes (con vistas a una posible ampliación a una tienda virtual). También incluye los métodos necesarios para publicar canales y contenidos, así como los métodos para obtener los contenidos de un canal publicado.

El entorno elegido es el framework Spring. Hemos elegido este framework para el trabajo de fin de carrera ya que es uno de los más utilizados en la industria y además facilita bastante la integración de patrones de diseño como el Modelo-Vista-Controlador y el acceso a base de datos a través del patrón DAO.

Por último hemos elegido Eclipse como entorno de desarrollo ya que es una herramienta líder del sector, gratuita y de sencilla utilización.

1.2. Justificación del proyecto

En el día a día de una empresa de distribución, se hace necesario mostrar al cliente un catálogo de productos actualizados y ajustados a la realidad. Al incrementarse la cartera de productos y debido a

la limitación de tiempo del que el cliente dispone, no siempre se tiene tiempo de mostrar todo el catálogo impreso de productos. Por ello es conveniente tener un sitio Web donde el cliente pueda consultar tranquilamente y con detalle todos los productos del catálogo.

La diferencia con respecto a un proyecto Web cualquiera es la introducción del concepto de publicación de contenidos. Asociaremos productos (el tipo de contenido principal de este proyecto) a canales de publicación. Al publicar ambos, canal y contenido, el usuario final podrá verlos reflejados en el sitio Web sin que el administrador tenga que editar código HTML. Obviamente, esto hace más sencillo que cualquier persona con unos conocimientos mínimos de informática pueda crear, modificar y publicar contenidos. Esto último es de vital importancia en una empresa que no cuente con personal informático con conocimientos de creación de páginas Web.

Otra ventaja para la realización del proyecto es que podemos cambiar la presentación de nuestros contenidos de una manera sencilla al separar los datos de la presentación.

1.3. Objetivos del trabajo

En este trabajo de fin de carrera se quiere hacer la aplicación de gestión de una cartera de productos. También se quieren proporcionar las herramientas para publicar dichos productos (contenidos) en el sitio Web de una manera sencilla y sin necesidad de tener grandes conocimientos sobre la Web o código HTML.

La idea principal de funcionamiento es que el usuario sea capaz de introducir nuevos productos y publicarlos sin necesidad de tocar la página Web. Para ello se definirán canales donde se asociarán los productos y al ser publicados dichos canales mostrarán los nuevos productos sin necesidad de retocar ningún código HTML.

También se implementará la gestión de clientes, proveedores y pedidos, por si en el futuro se decide crear una aplicación de venta de productos por Internet.

Como objetivo docente del proyecto se pretende profundizar en el conocimiento de Spring. Este framework, ampliamente usado en la industria, nos simplifica las tareas más complejas en un proyecto J2EE, separando de una manera más clara el código de la aplicación de la configuración de la misma. También nos ofrece una implementación muy sencilla del Modelo-Vista-Controlador (MVC), así como una forma más estandarizada de acceder a la base de datos mediante el patrón

DAO.

Para la consecución de estos objetivos me hemos basado en el análisis, diseño e implementación de una aplicación similar. Mi experiencia profesional me ha permitido trabajar en varios gestores de contenidos como Vignette (www.vignette.com), Plone (www.plone.org) y otras aplicaciones que si bien no son gestores de contenidos puros si que tienen la capacidad de gestionar contenidos tales como Jive (www.jivesoftware.com).

Obviamente, no es el objetivo del trabajo hacer una aplicación capaz de competir con un gestor de contenidos comercial. Con este trabajo intentamos responder a una necesidad real de una PYME sin recursos como para contratar personal técnico que haga el mantenimiento de su Web.

1.4. Enfoque metodológico

Para la realización del proyecto hemos intentado seguir el enfoque que se sigue en un entorno de trabajo real. He tenido la suerte de poder analizar un caso real que tiene la necesidad de una aplicación similar.

El primer paso fue estudiar las necesidades de una empresa de distribución y la selección de las herramientas para la consecución de los objetivos. Después se utilizó un ciclo en cascada con las siguientes etapas:

- Definición funcional.
- Planificación del proyecto.
- Análisis de la aplicación.
- Diseño de la aplicación.
- Implementación del proyecto.
- Pruebas.

La implementación y mantenimiento del proyecto no está contemplada de momento.

El enfoque metodológico a la hora de escoger las herramientas de desarrollo ha sido hacer un prototipo con Spring. He hecho una pequeña aplicación donde hemos estudiado las funcionalidades que necesitaba de Spring, tales como el acceso a base de datos y el uso del Modelo-Vista-

Controlador.

La idea principal a la hora de desarrollar, más que la de tener un producto terminado al 100%, es la de explorar las funcionalidades que ofrece Spring. Este framework posee casi una decena de módulos, Para el proyecto hemos seleccionado los dos más interesantes como son el Modelo-Vista-Controlador que ofrece Spring y el acceso a base de datos Spring JDBC (que usa el patrón DAO). También utilizamos el módulo principal de Spring que es el que conecta los demás módulos.

Para estudiar la viabilidad del proyecto hemos realizado un pequeño estudio de Spring para ver que funcionalidades se adaptaban mejor al proyecto. Tras decidir la utilización del Modelo-Vista-Controlador y de Spring JDBC hemos realizado el prototipo con un controlador que gestionará la creación de un modelo y una vista. Dicho prototipo contenía también una interfaz de acceso a la base de datos.

A la hora de empezar la implementación se ha seguido el siguiente proceso.

En primer lugar hemos diseñado el esquema de la base de datos para mapear el diagrama de clases utilizando para ello MySQL.

Después, utilizando eclipse, hemos hecho el mapeo de todas las clases incluyendo las funciones para hacer el encapsulado (ver paquete *com.smv.beans*). Una vez hecho el mapeado, hemos creado las clases que mapean las tablas con los *beans* (ver paquete *com.smv.dao.mappers*). Estas clases son imprescindibles para trabajar con Spring y JDCB. Podría haberlas creado como clases internas de las implementaciones de las interfaces de acceso a la base de datos, no obstante hemos preferido separarlas para mayor claridad del código.

Una vez terminadas estas clases hemos implementado las clases DAO de acceso a los objetos de la base de datos (ver paquete *com.smv.dao*). Cada *bean* tiene una interfaz que define los distintos métodos de acceso que tiene cada *bean* y cada interfaz tiene una clase de implementación que es la que ejecuta el acceso a la base de datos a través de Spring JDBC.

Una vez finalizado el trabajo con la base de datos hemos implementado el Modelo-Vista-Controlador (ver paquete *com.smv.mvc.controller*). Por ahora hemos hecho un controlador para la inserción de nuevos productos, proveedores, canales y clientes, así como el listado de todos los anteriores más los pedidos (se supone que se añaden en la parte del cliente). Falta por terminar la edición de los productos donde también se controlará la publicación de canales y productos.

El controlador se encarga de la creación de los contenidos y también de enviar los modelos

(información que utilizarán las vistas). La implementación de las vistas se han hecho utilizando lenguaje JSP.

El Modelo-Vista-Controlador se ha implementado utilizando Spring MVC.

El proceso se ha ido revisando a medida que ha crecido el proyecto sobre todo la parte del acceso a base de datos ya que las necesidades han ido evolucionando a medida que se hacia evidente la necesidad de nuevas consultas a la base de datos.

1.5. Planificación del proyecto

El proyecto se ha realizado de Septiembre a Enero. Al no poder dedicar siempre un tiempo fijo por semana por compromisos laborales hemos supuesto que el trabajo se ha realizado de lunes a viernes, aunque hemos trabajado todos o casi todos los fines de semana.

El proyecto está sujeto a unas restricciones de tiempo que se corresponden con las fechas establecidas por el consultor. El siguiente diagrama de Gantt recoge estas restricciones de tiempo.

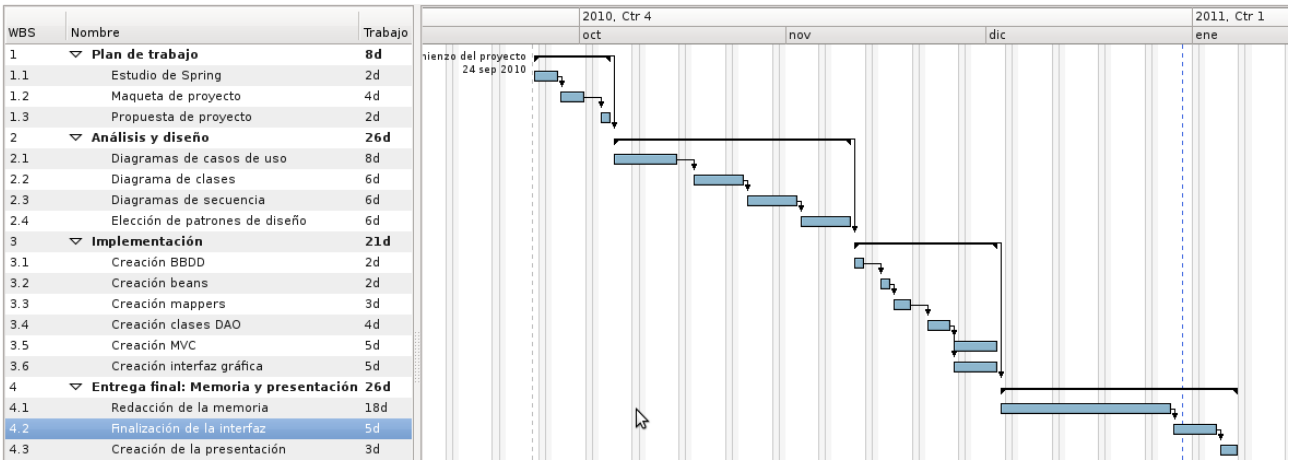

Figura 1: Diagrama de Gantt de realización del proyecto

WBS	Nombre	Inicio	Fin	Trabajo
1	**Plan de trabajo**	Sep 24	Oct 5	8d
1.1	Estudio de Spring	Sep 24	Sep 27	2d
1.2	Maqueta de proyecto	Sep 28	Oct 1	4d
1.3	Propuesta de proyecto	Oct 4	Oct 5	2d
2	**Análisis y diseño**	Oct 6	Nov 10	26d
2.1	Diagramas de casos de uso	Oct 6	Oct 15	8d
2.2	Diagrama de clases	Oct 18	Oct 25	6d
2.3	Diagramas de secuencia	Oct 26	Nov 2	6d
2.4	Elección de patrones de diseño	Nov 3	Nov 10	6d
3	**Implementación**	Nov 11	Dec 2	21d
3.1	Creación BBDD	Nov 11	Nov 12	2d
3.2	Creación *beans*	Nov 15	Nov 16	2d
3.3	Creación mappers	Nov 17	Nov 19	3d
3.4	Creación clases DAO	Nov 22	Nov 25	4d
3.5	Creación MVC	Nov 26	Dec 2	5d
3.6	Creación interfaz gráfica	Nov 26	Dec 2	5d
4	**Entrega final: Memoria y presentación**	Dec 3	Jan 7	26d
4.1	Redacción de la memoria	Dec 3	Dec 28	18d
4.2	Finalización de la interfaz	Dec 29	Jan 4	5d
4.3	Creación de la presentación	Jan 5	Jan 7	3d

Tabla 1: Tabla de fechas con los días asignados para cada tarea

1.6. Productos obtenidos

Se ha obtenido una aplicación empaquetada en un fichero war. Dicho fichero se puede desplegar directamente en un servidor tomcat. El nombre de la aplicación es SMVLogistica.war.

También se ha obtenido esta memoria como resumen del trabajo realizado.

Por último, también se incluye una presentación en Powerpoint que muestra los principales hitos del trabajo.

2. Especificación y requerimientos

2.1. Introducción

A la hora de hacer la especificación y requisitos del trabajo hemos tenido en cuenta dos factores:

- Qué necesita la aplicación.
- Hacia donde podríamos extender su uso.

A medida que hacemos la especificación y requisitos, señalaremos que es lo que está implementado y listo para utilizar, y que esta hecho pero sin interfaz de usuario. La idea general, es la gestión del contenido. La interacción con el sitio Web que se decida hacer es secundaria y por tanto, hemos intentado dejar la implementación lo más abierta posible, con vistas a las diferentes posibilidades de interacción con un sitio Web.

Como comentarios generales sobre esta fase hemos intentado hacer un esquema sencillo y fácil de entender. A mi parecer, al realizar un proyecto informático debemos jugar siempre con la idea de hacer una buena documentación que acerque la aplicación tanto a los usuarios técnicos como a los que no lo son.

Por último, hemos intentado hacer una descripción de los patrones de diseño utilizados en la aplicación, tanto directa, como indirectamente. De esta manera queremos reflejar el trabajo de investigación que hemos realizado.

2.2. Diagrama de casos de uso

2.2.1. Diagrama general de casos de uso

En este diagrama vemos la idea general del proyecto. En él vemos una parte, que detallaremos después, donde el administrador se encarga de la gestión de los productos y canales, así como de la publicación de contenidos que se reflejan en el sitio Web. También vemos una parte que sería la extensión del proyecto, donde el cliente, al acceder al sitio Web, podría realizar pedidos que se recogerían en la aplicación de administración. Esta parte esta implementada en la parte de administración.

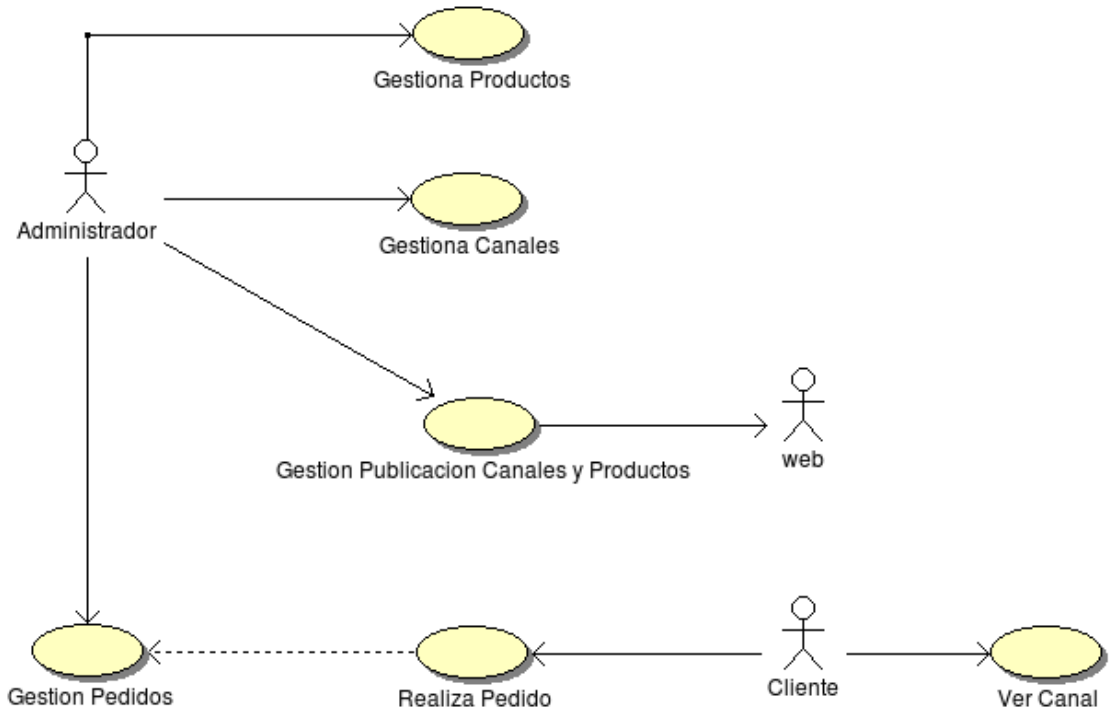

Figura 2: Diagrama general de casos de uso

2.2.2. Creación y publicación de contenidos

En este diagrama se muestra el funcionamiento principal del programa. En él, vemos que el administrador crea, modifica o elimina productos del sistema, así como también se encarga de la gestión de los canales. Los canales se encargan de contener la información. El administrador del sistema se encarga también de publicar y despublicar contenido, el cual se reflejará en la Web del sistema. Vemos que los módulos de gestión de productos y de gestión de canales solo interactuan con el sitio Web a través de la gestión de la publicación.

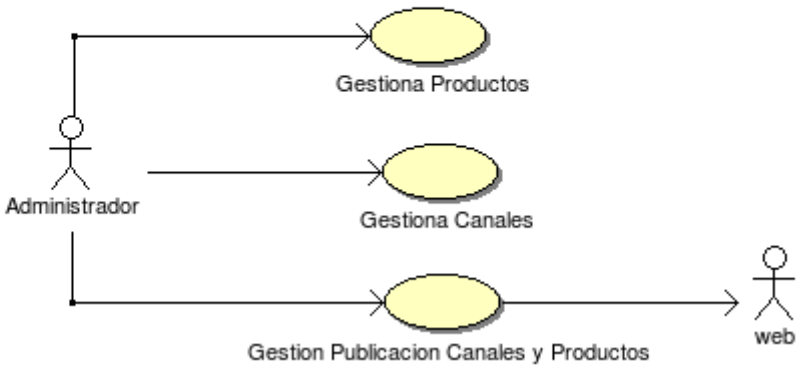

Figura 3: Diagrama de casos de uso de la creación y publicación de contenidos

2.3. Diagrama de clases y de secuencia

2.3.1. Diagrama de clases general

En este diagrama vemos las clases más relevantes del sistema. Para proporcionar mayor claridad se han omitido los métodos get y set de cada atributo de cada clase, y que se requieren para tener una buena encapsulación. También se ha omitido la visibilidad, ya que presuponemos que todos los atributos serán privados. A continuación, veremos con más detalle cada parte del diagrama.

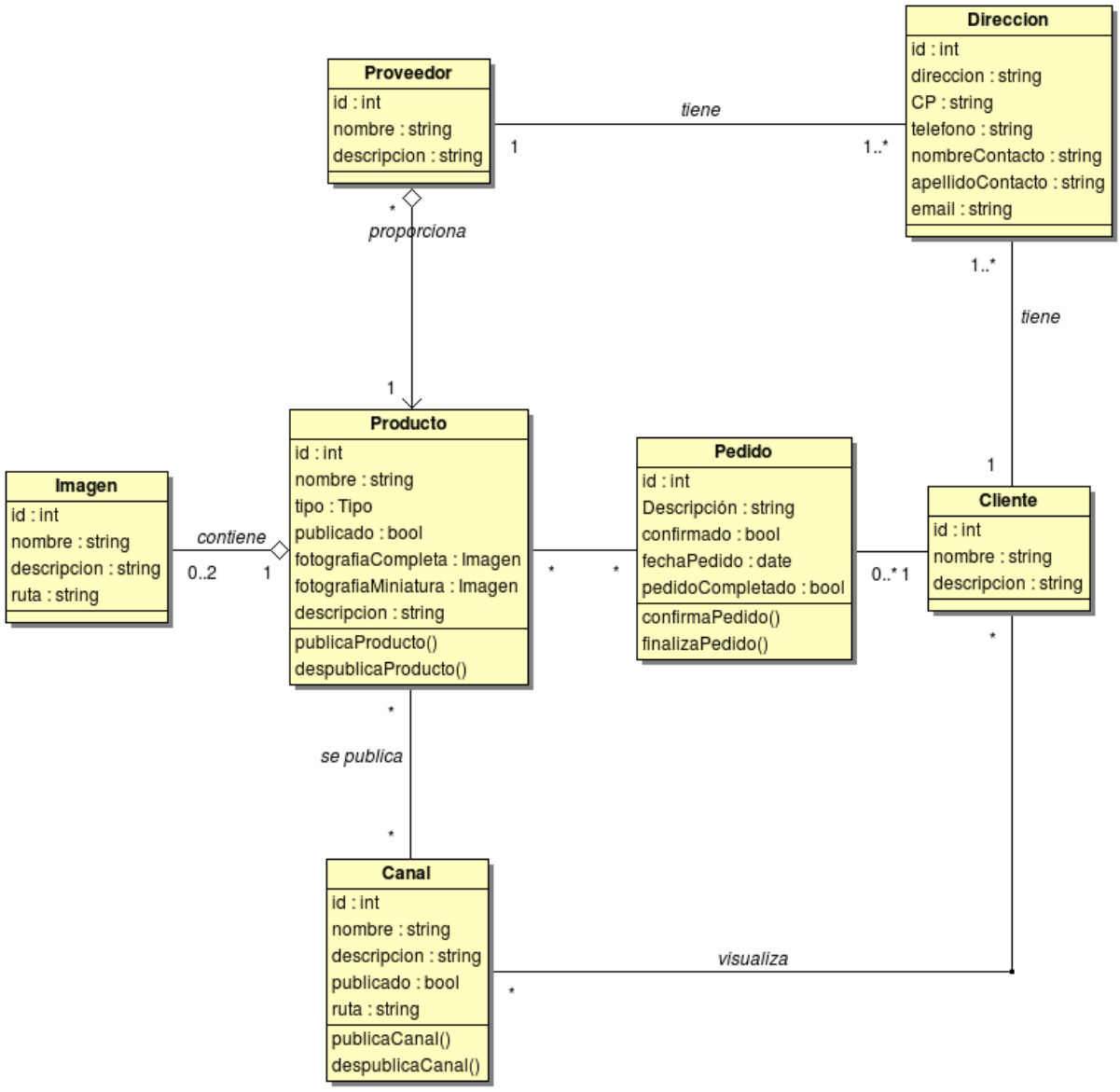

Figura 4: Diagrama de clases general.

2.3.2. Diagrama de clases de los contenidos

En este diagrama vemos la estructura básica de los contenidos que se publicarán en la Web. Se han omitido constructores y métodos get y set de los atributos. Como centro de la aplicación tenemos el Producto. A dicho producto se le asocian Canales e Imágenes. Menos importante es el Tipo, creado tan solo como elemento que puede ayudar a la ordenación en el sitio Web.

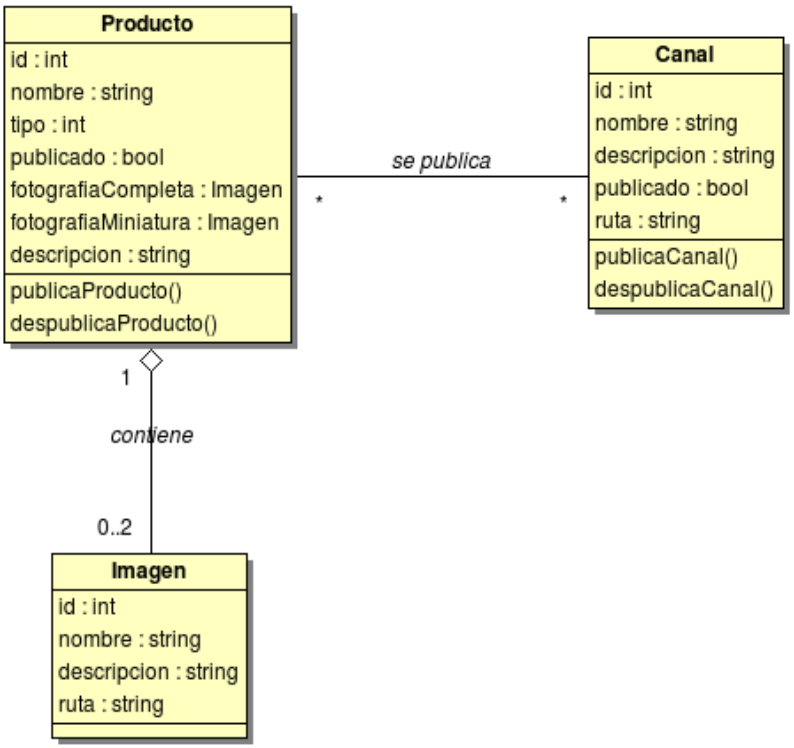

Figura 5: Diagrama de clases de los contenidos.

2.3.3. Diagrama de clientes/proveedores

En este diagrama, tenemos las clases para representar a los clientes y a los proveedores. También los pedidos que se tramiten en la Web. Vemos que los productos están asociados a los proveedores, y estos están asociados a una dirección al igual que los clientes. Aunque el sistema soporta varias direcciones, en la interfaz solo hemos puesto la opción de poner una única dirección.

También tenemos la capacidad de gestionar pedidos, aunque como tan solo nos hemos centrado en la parte de administración no hemos desarrollado esta parte de la interfaz.

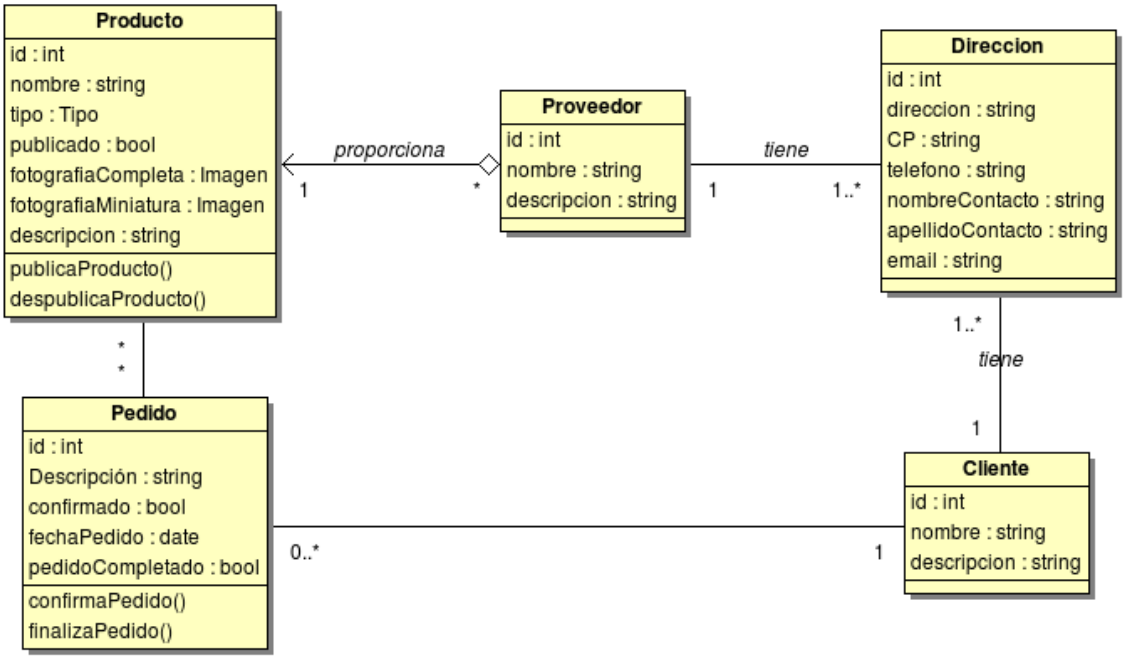

Figura 6: Diagrama de clases de clientes y proveedores

2.3.4. Diagrama de secuencia de publicación de contenidos

En este diagrama vemos como el administrador crea productos y los asigna a uno o varios canales. Para que un producto se refleje en la Web se debe publicar el mismo y el canal que lo contiene. Para despublicarlo basta con despublicar el producto.

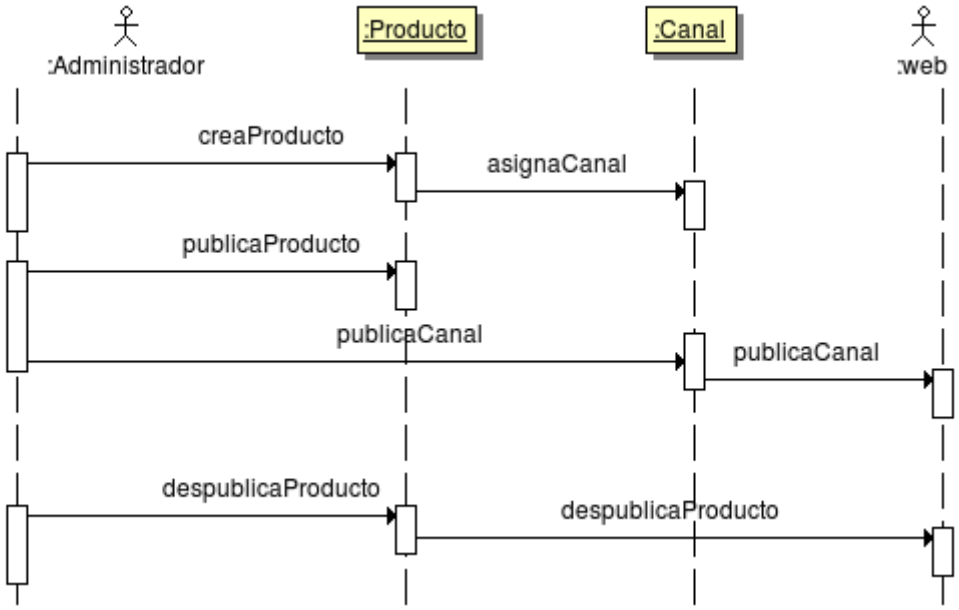

Figura 7: Diagrama de secuencia de publicación de contenidos.

2.3.5. Diagrama de secuencia de Ejecución de pedidos

En este diagrama de secuencia veremos como trataría el sistema a los pedidos que el cliente realiza.

En primer lugar, el cliente selecciona un canal que le muestra uno o varios productos. Después, el cliente selecciona uno o varios productos y los añade a su pedido. Esta parte estaría implementada en el sitio Web que publicaría los contenidos gestionados por nuestra aplicación.

Una vez que el cliente esta satisfecho, validaría su pedido y de manera asíncrona el administrador lo recoge y negocia con el proveedor. Al recibir respuesta de este, el administrador contacta con el cliente para ofrecerle un presupuesto.

Esta parte, que a priori se podría haber implementado como una tienda virtual (esa era la idea original), tuvo que ser descartada ya que no se trata con un usuario final. Los pedidos suelen ser de proporciones muy dispares y además, al ser la empresa del sector alimentario, los precios fluctúan con regularidad y de manera imprevisible, siendo muy normal que se desarrolle una negociación con cada pedido donde se añaden y quitan productos y los precios suben o bajan según el volumen de dicho pedido.

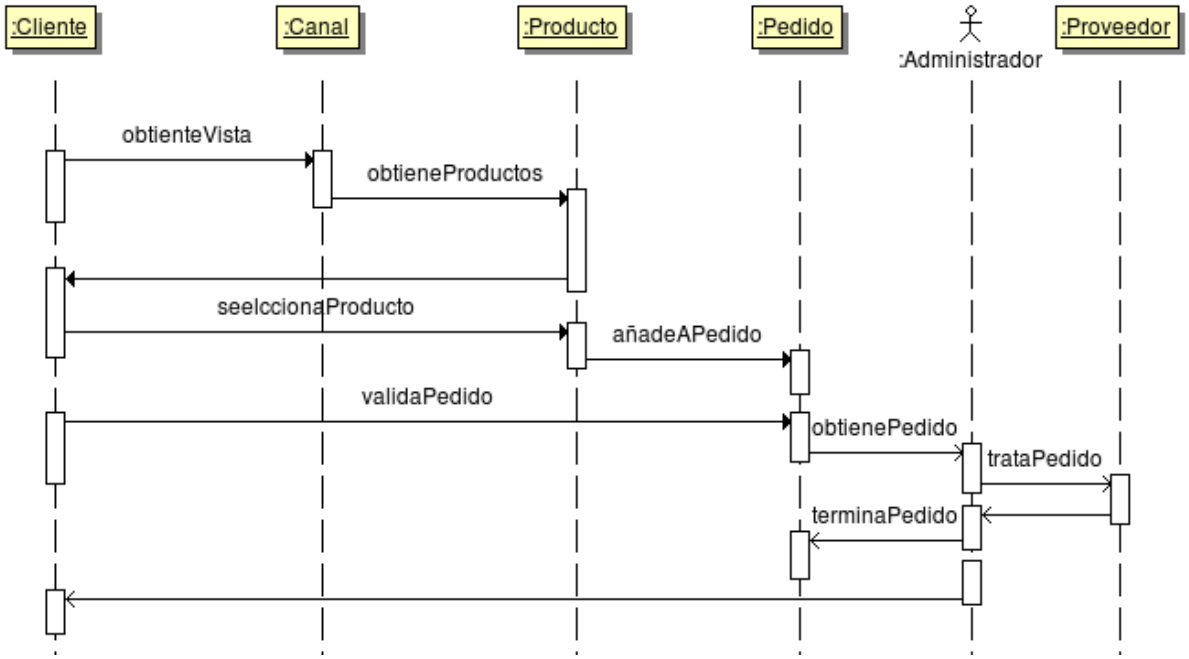

Figura 8: Diagrama de secuencia de ejecución de pedidos.

2.4. Patrones de diseño utilizados

2.4.1. Introducción

En este proyecto vamos a utilizar como framework de desarrollo Spring. Gracias a su uso vamos a utilizar de forma implícita una serie de patrones de diseño de los que vamos a destacar el **MVC** (modelo-vista-controlador), el patrón **Singleton** o el patrón **DAO**. Obviamente, Spring incluye multitud de patrones, pero estos tres son los que más influencia tendrán en el proyecto.

También comentaremos como ha sido la implementación de dichos patrones en el proyecto y cuales han sido los principales problemas que nos hemos encontrado a la hora de utilizarlos.

2.4.2. Patrón MVC (Modelo-Vista-Controlador)

Este patrón, muy utilizado hoy en día, nos va a permitir dividir la problemática de la interfaz de usuario en tres partes bien definidas. El *Modelo* va a guardar el estado de la aplicación. La *Vista* interpretará los datos del modelo y la presentará al usuario y finalmente, el *Controlador* procesará la información introducida por el usuario y, o bien actualizará el modelo, o seleccionará la vista adecuada para mostrar.

Un esquema general del **MVC** podría ser el siguiente.

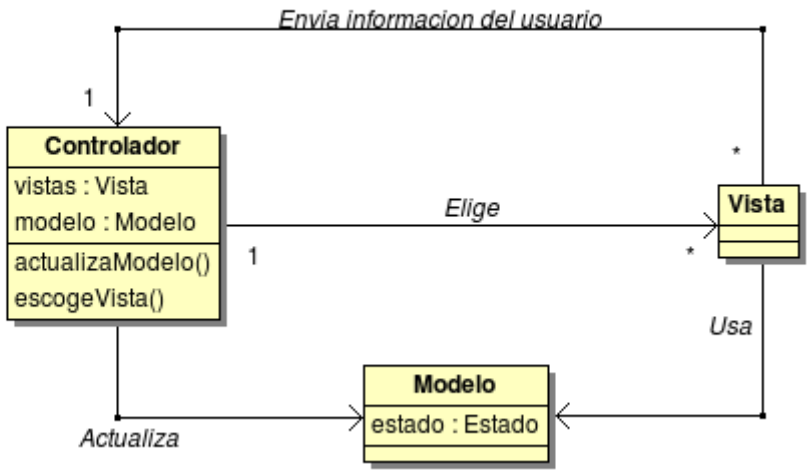

Figura 9: Diagrama del Modelo-Vista-Controlador

Así por ejemplo, el **MVC** nos ayudará a elegir que interfaz mostrar al usuario después de haber confirmado un pedido o de haber creado un nuevo producto.

Spring contiene un modulo específico para el uso del **MVC**. Gracias al cual no ha sido especialmente difícil hacer la implementación. Si que hemos visto un poco de complejidad cuando el controlador tiene que manejar demasiadas opciones. Quizá en este punto deberíamos haber hecho varios controladores en lugar de uno solo para simplificar un poco la lógica.

He intentado remediar esta complejidad creando una clase que me ayudara a gestionar los datos del modelo. Para ello hemos hecho una implementación del patrón Factory, aunque un poco retocado para adaptarlo a las necesidades del proyecto. Si volviera a hacer el proyecto de nuevo, sin duda haría una separación más fácil de entender en la lógica del controlador, así como una implementación más limpia del patrón Factory para gestionar el modelo.

He encontrado muy útil la selección de vistas. Resulta muy sencillo cambiar una vista por otra y aunque para el proyecto hemos utilizado JSP como tecnología a la hora de mostrar la vista, sería muy sencillo utilizar otro lenguaje como FTL (freemarker.sourceforge.net).

También hemos encontrado muy sencillo el paso de modelos, con varias opciones según el número y formato de los modelos a enviar a la vista.

Con respecto a otras tecnologías que implementan el **MVC**, como Struts, hemos visto que Spring es un poco menos formal. Dejando la complejidad a manos de framework.

Como comentario final quería destacar la ventaja de utilizar Spring ya que su modularidad permite que si tan solo queremos utilizar su módulo de **MVC**, no tenemos más que importar la librería correspondiente. No necesitamos engorrosos ficheros de configuración ni librerías de código pesadas que necesitan mayor tiempo para comprender su lógica.

2.4.3. Patrón Singleton

El patrón **Singleton** nos ayudara a obtener una instancia de un objeto única e igual para todo el mundo, proporcionando una entrada global para dicho objeto.

Cuando empecé el proyecto, inmediatamente pensé en este patrón a la hora de acceder a la base de datos, ya que nos permite tener una única instancia de acceso a la base de datos que nos gestionará todo el acceso, evitándonos así tener que lidiar con la apertura y cierre de conexiones a la base de datos, por cierto, una de las causas más comunes de ralentización en un aplicación Web.

Al empezar el estudio de Spring observé que el módulo Spring JDBC usa este patrón para hacernos invisible toda la lógica de acceso a la base de datos.

El patrón Singleton se usa de manera interna en el acceso a la base de datos. Spring se encarga de establecer un pool de conexiones a través del cual vamos obteniendo las conexiones necesarias a la base de datos. Este pool se crea al iniciar la aplicación y se va obteniendo la instancia original.

De esta manera Spring se encarga de crear la conexión a la base de datos, también se encarga de crear un pool de conexiones para optimizar el acceso a la base de datos y por último nos da métodos de acceso donde lo único de lo que tenemos que preocuparnos es del código SQL que queremos ejecutar.

2.4.4. Patrón DAO (Data Access Object)

Este patrón, tiene como objetivo simplificar y separar la lógica de los recursos de acceso de la base de datos. Así, el controlador de la aplicación no tiene que ocuparse del acceso a los recursos, sino tan solo llamar a las clases DAO. Tampoco el controlador necesita saber el origen de los datos. El patrón DAO utilizará una persistencia (en este caso será la de Spring, si bien podríamos reemplazarla por otras).

La idea principal de este patrón es la de crear interfaces de acceso a la base de datos. De esta manera, cuando queramos acceder a algún recurso de la base de datos no tendremos más que hacer uso de dicha interfaz.

Por supuesto, tendremos que crear la implementación de dichas interfaces, pero una vez hechas cualquier otro desarrollador, o nosotros mismos, no tendrá que preocuparse del acceso a la base de datos.

A la hora de aplicar este patrón en el proyecto hemos encontrado que Spring tiene un módulo dedicado a este propósito. Dicho módulo contiene todas las herramientas necesarias para la creación de las interfaces.

También contiene herramientas para el acceso a la base de datos y para mapear los resultados a los *beans* correspondientes.

Si bien al principio, hemos encontrado demasiado engorroso tener que crear una clase de mapeo para cada *bean*, una interfaz de acceso, también una por cada *bean*, y por último, una implementación por cada interfaz. A la hora de hacer la implementación del trabajo, hemos visto que realmente merece la pena utilizarlo. Cuando necesitábamos acceder a la base de datos para poblar un objeto, lo único que teníamos que hacer era llamar a la interfaz y ella se ocupaba de poblar los objetos con los datos almacenados en las tablas.

Hice una búsqueda para encontrar aplicaciones que facilitarán la creación de las clases para implementar el patrón DAO, pero desafortunadamente no encontré ninguna que satisficiera las necesidades del proyecto.

Por último, destacar que aunque hemos utilizado la implementación más sencilla que ofrece Spring, llamada SimpleSpringJDBC, existen más implementaciones preparadas para sistemas que necesiten un acceso mucho mas eficiente, que no es el caso de nuestra aplicación.

3. Implementación

3.1. Herramientas de desarrollo

Para la realización de este proyecto hemos utilizado las siguientes herramientas:

- **Base de datos**: Para la base de datos hemos utilizado *MySQL* (http://www.mysql.com/). En realidad, la selección de la base de datos era algo secundario, en tanto fuera una base de datos relacional. Al utilizar Spring, podemos cambiar fácilmente la base de datos modificando las propiedades de Spring. He seleccionado mysql, porque ya había trabajado anteriormente con el y además de ser gratuita, para este tipo de aplicaciones es relativamente sencilla de usar. Otra opción estudiada ha sido Postgresql, pero finalmente la herramienta de administración de MySQL me ha hecho decantarme por esta última.
- **Utilidades para la base de datos**: Para la administración de la base de datos hemos utilizado *MySQL Administrator*. Para realizar consultas a la base de datos hemos utilizado *MySQL Query Browser*.
- **Entorno de desarrollo**: Hemos utilizado *Eclipse* (http://www.eclipse.org/) para la implementación de la aplicación. No hemos utilizado ningún plugin de desarrollo de eclipse para Spring, aunque hay varios en el mercado como *MyEclipse*. Hice un pequeño estudio sobre dichos plugins, pero llegué a la conclusión de que no merecía la pena utilizar ninguno, en parte debido a que la mayoría son de pago, y que además, el grado de asistencia que dan es limitado.
- **Contenedor de aplicaciones**: Como contenedor de aplicaciones hemos utilizado *Apache Tomcat* (http://tomcat.apache.org). Es un contenedor de aplicaciones de amplio uso en el mundo profesional y gratuito. He utilizado este contenedor en casi todos los proyectos profesionales que he hecho, y a parte de ser gratuito, como ya hemos mencionado, es además muy ligero. También es muy sencilla la integración del contenedor con el entorno de desarrollo, lo que me ha permitido ejecutar el código en modo de depuración, ahorrando tiempo en la corrección de errores en tiempo de ejecución.
- **Navegador**: He utilizado *Google Chrome* (http://www.google.com/chrome?hl=es) como navegador para la implementación, aunque el código HTML es también compatible Internet Explorer y Mozilla Firefox. Ha sido particularmente útil la herramienta que ponen a

disposición de los desarrolladores para depurar el código HTML y el código Javascript.

Como características comunes a todas las herramientas, hemos favorecido siempre las que son de libre distribución. En segundo lugar hemos optado por las que tenían una comunidad de usuarios más activa, y por último, hemos intentado que el conjunto de herramientas estuviera preparado para integrarse con las demás. Así, la selección de *Eclipse* como entorno de desarrollo, vino motivada por la integración que hace de *Tomcat* y de *MySQL*.

Hemos utilizado *Ubuntu* como sistema operativo en la mayor parte del proyecto, aunque hemos utilizado Windows en algunas ocasiones.

En lo referente al editor de textos, todos los documentos han sido generados con Open Office (www.openoffice.org). Aunque hemos intentado que sea compatible con Microsoft Office.

Por último, quería señalar que gracias a que había trabajado con la mayoría de productos en el pasado, no he tenido demasiados problemas a la hora del desarrollo. Únicamente, he perdido algo de tiempo cuando intenté encontrar, sin éxito, una extensión de Eclipse para hacer la parte del acceso a la base de datos.

3.2. Manual de instalación

3.2.1. Base de datos

Para este proyecto hemos utilizado MySQL 5.1.

(http://dev.mysql.com/downloads/mysql/5.1.html#downloads)

El usuario **root** debe tener como contraseña **admin**. Sino deberemos cambiar el código de la clase de acceso a la base de datos (DbUtils.java)

Una vez instalado el programa, crearemos un esquema llamado *SMVLogistica* con el usuario **root**. Para ello ejecutaremos el siguiente código.

Create Schema SMVLogistica;

use SMVLogistica;

Después de asegurarnos que no hay ninguna tabla creada, creamos los esquemas y las relaciones.

```
ALTER TABLE `Canal` DROP PRIMARY KEY;

ALTER TABLE `Producto` DROP PRIMARY KEY;

ALTER TABLE `Tipo` DROP PRIMARY KEY;

ALTER TABLE `ProductoCanal` DROP PRIMARY KEY;

ALTER TABLE `Pedido` DROP PRIMARY KEY;

ALTER TABLE `PedidoProducto` DROP PRIMARY KEY;

ALTER TABLE `Cliente` DROP PRIMARY KEY;

ALTER TABLE `Direccion` DROP PRIMARY KEY;

ALTER TABLE `Imagen` DROP PRIMARY KEY;

ALTER TABLE `Proveedor` DROP PRIMARY KEY;

DROP TABLE `Direccion`;

DROP TABLE `ProductoCanal`;

DROP TABLE `Tipo`;

DROP TABLE `PedidoProducto`;

DROP TABLE `Proveedor`;

DROP TABLE `Canal`;

DROP TABLE `Cliente`;

DROP TABLE `Pedido`;

DROP TABLE `Imagen`;

DROP TABLE `Producto`;
```

```
CREATE TABLE `Direccion` (
        `id` INT NOT NULL AUTO_INCREMENT,
        `direccion` VARCHAR(4000) NOT NULL,
        `CP` VARCHAR(20) NOT NULL,
        `telefono` VARCHAR(20),
        `nombreContacto` VARCHAR(100),
        `apellidoContacto` VARCHAR(100),
        `email` VARCHAR(200),
        PRIMARY KEY (`id`)
) ENGINE=InnoDB;

CREATE TABLE `ProductoCanal` (
        `producto` INT NOT NULL,
        `canal` INT NOT NULL,
        PRIMARY KEY (`producto`,`canal`)
) ENGINE=InnoDB;

CREATE TABLE `Tipo` (
        `id` INT NOT NULL  AUTO_INCREMENT,
        `nombre` VARCHAR(255) NOT NULL,
        `descripcion` VARCHAR(4000) NOT NULL,
        PRIMARY KEY (`id`)
) ENGINE=InnoDB;
```

```
CREATE TABLE `PedidoProducto` (
        `pedido` INT NOT NULL,
        `producto` INT NOT NULL,
        PRIMARY KEY (`pedido`,`producto`)
) ENGINE=InnoDB;

CREATE TABLE `Proveedor` (
        `id` INT NOT NULL  AUTO_INCREMENT,
        `nombre` VARCHAR(255) NOT NULL,
        `descripcion` VARCHAR(4000),
        `direccion` INT NOT NULL,
        PRIMARY KEY (`id`)
) ENGINE=InnoDB;

CREATE TABLE `Canal` (
        `id` INT NOT NULL  AUTO_INCREMENT,
        `nombre` VARCHAR(255) NOT NULL,
        `descripcion` VARCHAR(4000) NOT NULL,
        `publicado` BIT NOT NULL,
        `ruta` VARCHAR(4000) NOT NULL,
        PRIMARY KEY (`id`)
) ENGINE=InnoDB;
```

```
CREATE TABLE `Cliente` (
        `id` INT NOT NULL  AUTO_INCREMENT,
        `nombre` VARCHAR(255) NOT NULL,
        `descripcion` VARCHAR(4000) NOT NULL,
        `direccion` INT NOT NULL,
        PRIMARY KEY (`id`)
) ENGINE=InnoDB;

CREATE TABLE `Pedido` (
        `id` INT NOT NULL  AUTO_INCREMENT,
        `descripcion` VARCHAR(4000) NOT NULL,
        `confirmado` BIT NOT NULL,
        `fechaPedido` DATE NOT NULL,
        `pedidoCompletado` BIT NOT NULL,
        `cliente` INT NOT NULL,
        PRIMARY KEY (`id`)
) ENGINE=InnoDB;

CREATE TABLE `Imagen` (
        `id` INT NOT NULL  AUTO_INCREMENT,
        `nombre` VARCHAR(400) NOT NULL,
        `descripcion` VARCHAR(400) NOT NULL,
        `ruta` VARCHAR(255),
        PRIMARY KEY (`id`)
) ENGINE=InnoDB;
```

```
CREATE TABLE `Producto` (
        `id` INT NOT NULL  AUTO_INCREMENT,
        `nombre` VARCHAR(400) NOT NULL,
        `publicado` BIT NOT NULL,
        `descripcion` VARCHAR(4000),
        `fotografiaCompleta` INT,
        `fotografiaMiniatura` INT,
        `tipo` INT,
        `proveedor` INT NOT NULL,
        PRIMARY KEY (`id`)
) ENGINE=InnoDB;
```

3.2.2. Contenedor de aplicaciones

Para este proyecto hemos seleccionado Apache Tomcat como contenedor de aplicaciones. La versión utilizada es la 6 (http://tomcat.apache.org/download-60.cgi). Una vez descargado, lo instalaremos con las opciones por defecto que nos indique el programa.

3.2.3. Instalación de la aplicación

Para instalar la aplicación, pondremos el fichero SMVLogistica.war en la carpeta *<instalación tomcat>/webapps*.

Necesitamos también las librerías de Spring. Se pueden descargar desde aquí
http://s3.amazonaws.com/dist.springframework.org/release/SPR/spring-framework-3.0.5.RELEASE.zip

Una vez descargado el fichero, lo abriremos y pondremos todos los jar dentro de la caperta lib en

<instalación tomcat>/webapps/SMVLogistica/WEB-INF/lib

Por último reiniciaremos el contenedor de aplicaciones.

3.2.4. Acceso a la aplicación

Para acceder a la aplicación abriremos un navegador e introduciremos la siguiente dirección.

http://localhost:<puertoTomcat>/SMVLogistica/admin.htm

El puerto suele ser por defecto 8080.

3.3. Guía del usuario

En la guía del usuario se explican todas las opciones implementadas en la aplicación. A continuación describimos las principales páginas que forman parte de la aplicación. Para ello nos serviremos de capturas de pantalla.

3.3.1. Menú principal de la consola de administración

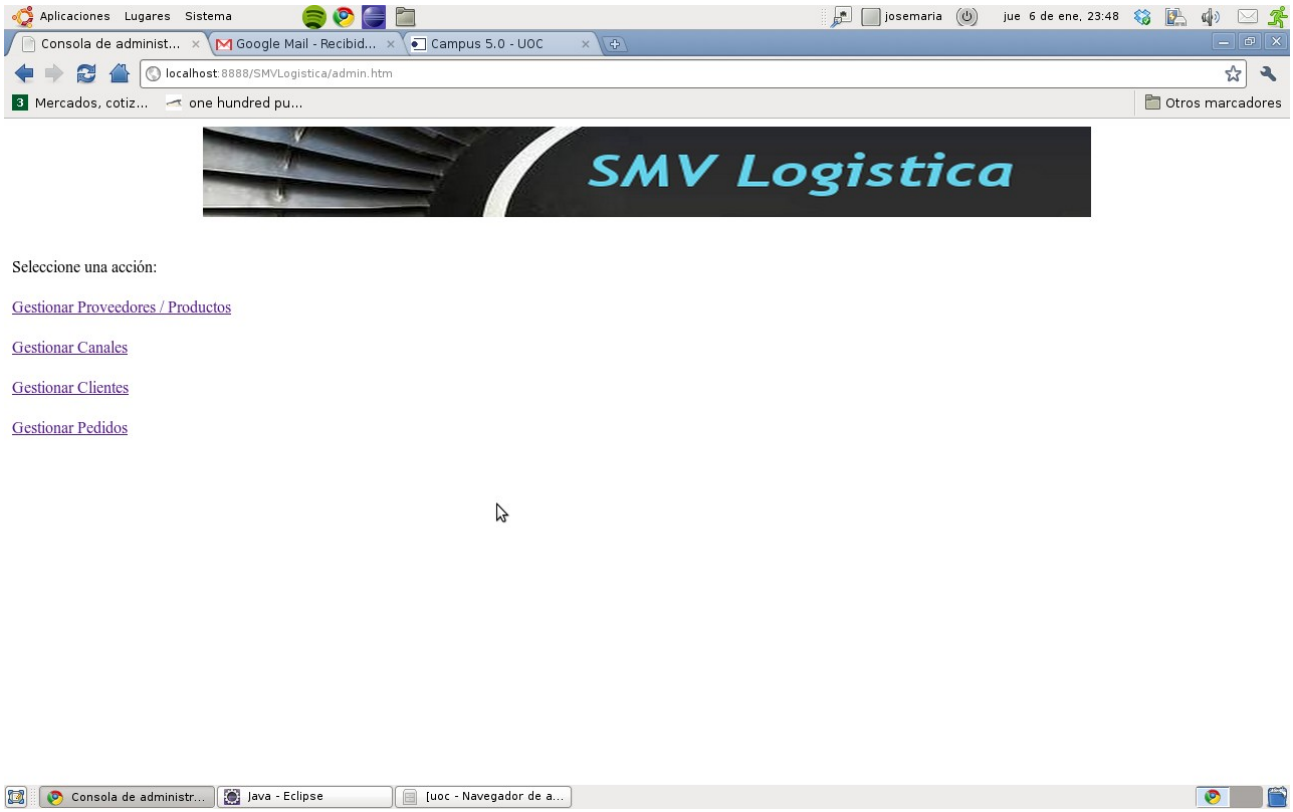

Figura 10:Menú principal de la consola de administración

Tras acceder a la URL de la consola de administración nos encontramos cuatro opciones: en la primera se accede a la gestión de productos y proveedores, en la segunda se accede a la gestión de canales, en la tercera se accede a la gestión de clientes y por último, en la cuarta opción, se accede a la gestión de pedidos.

3.3.2. Gestión de proveedores y productos

Figura 11: Menú de gestión de proveedores y productos.

En esta pantalla se muestran las opciones de añadir y visualizar tanto productos como proveedores. También existe una opción para volver al menú principal.

3.3.2.1. Añade un producto

Figura 12: Nuevo producto.

En esta pantalla se nos da la opción para añadir un producto. Introduciremos el nombre, seleccionaremos el tipo de producto (se podrán añadir, editar o borrar los tipos en esta pantalla), las fotografías completas y en miniatura del producto, la descripción y el proveedor del producto. Las imágenes todavía están en desarrollo.

3.3.3.2. Visualiza producto

Figura 13: Visualiza producto.

En esta pantalla se muestra un listado de todos los productos, mostrando el nombre, la descripción y el estado de publicación. Al seleccionar un producto, podremos modificarlo, borrarlo, añadirle canales, quitarle canales, publicarlo o despublicarlo. Si modificamos un atributo del producto se cambiará el estado de la publicación automáticamente a despublicado.

3.3.3.3. Añade proveedor

Figura 14: Añade proveedor.

En esta pantalla podemos añadir un proveedor así como sus detalles de contacto.

3.3.3.4. Visualiza proveedor

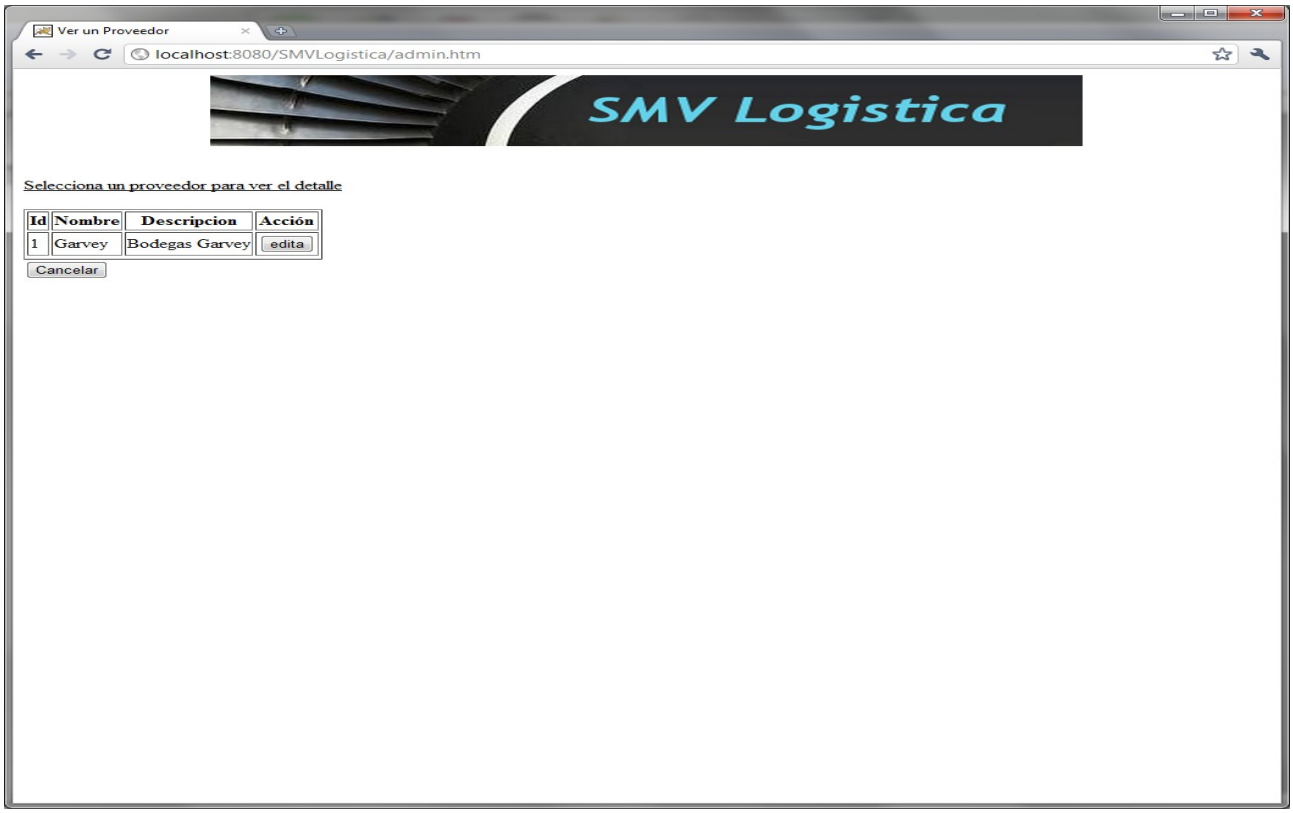

Figura 15: Visualiza proveedor.

En esta pantalla se muestra un listado de todos los proveedores. Al seleccionar un proveedor podremos modificarlo.

3.3.4. Gestión de canales

Figura 16: Gestión de canales.

En esta pantalla tendremos la opción de añadir y visualizar canales, así como la opción de volver al menú principal.

3.3.4.1. Añade canal

Figura 17: Añade canal.

En esta pantalla podremos añadir el nombre, la descripción y la ruta de un canal.

3.3.4.2. Visualiza canal

Figura 18: Visualiza canal.

En esta pantalla se muestra un listado con todos los canales, al seleccionar un canal, podremos modificarlo.

3.3.5. Gestión de clientes

Figura 19: Gestión de clientes.

En esta pantalla tendremos la opción de añadir y visualizar clientes, así como la opción de volver al menú principal.

3.3.5.1. Añade cliente

Figura 20: Añade cliente.

En esta pantalla podemos añadir un cliente así como sus detalles de contacto.

3.3.5.2. Visualiza cliente

Figura 21 Visualiza cliente.

En esta pantalla se muestra un listado de todos los clientes. Al seleccionar un cliente podremos modificarlo.

3.3.6. Gestión de pedidos

Figura 22: Gestión de pedidos.

En esta pantalla tendremos la opción de visualizar pedidos, así como la opción de volver al menú principal.

3.3.6.1. Visualiza pedido

Figura 23: Visualiza pedido.

En esta pantalla se muestra un listado de todos los pedidos. Al seleccionar un pedido podremos modificarlo, borrarlo o confirmarlo. Aunque esta opción esta contemplada en la aplicación de administración, no ha sido completamente desarrollada, ya que la parte que alimenta la aplicación con los pedidos estará en el sitio Web. He preferido dejarla abierta, aunque con casi todos los métodos de acceso y gestión para una implementación futura.

3.4. Juego de pruebas

Para realizar las pruebas, podemos ejecutar en la base de datos el contenido del fichero *ContenidoPruebas.sql*. Una vez importados, tenemos tres tipos de pruebas (todas en el paquete test del proyecto):

- Conexión a la base de datos: *TestDatasource.java*
- Creación, modificación y borrado de contenidos: *TestClient.java*
- Acceso a los contenidos publicados: *TestContenidos.java*

La primera prueba, *TestDatasource.java* está destinada a probar la conexión de la base de datos. En el caso de que la prueba fuera insatisfactoria se debe revisar la clase *DbUtils.java* que contiene la cadena de conexión a la base de datos.

La segunda prueba, *TestClient.java*, tiene por objeto probar la creación de contenidos en la base de datos. En ella, creamos un objeto de la clase tipo y lo añadimos a la base de datos. Lo modificamos, y por último lo borramos. Esta misma clase la podemos modificar fácilmente para probar el resto de clases de acceso a la base de datos.

Por último hemos creado una clase de prueba para la publicación de contenidos. *TestContenidos.java* se encarga de probar el método que devuelve todos los productos publicados asociados a un canal concreto.

La mejor manera de lanzar las pruebas es con eclipse, sino, podemos ejecutar *java test.NombreDeLaClase* en el compilado del proyecto.

Por restricciones de tiempo no hemos podido implementar clases de prueba para el entorno gráfico. De todas maneras podríamos utilizar *HttpUnit* (httpunit.sourceforge.net) para realizar estos test de manera automatizada.

4. Conclusiones

4.1. Plataforma

Spring ha demostrado ser un framework robusto y de fácil aprendizaje (teniendo una amplia base en programación Java). Estas son las principales características:

- Spring es fácilmente extensible, basta añadir las librerías necesarias para activar más funcionalidades.
- He encontrado muy útil y fácil de usar la implementación del **MVC** (Modelo-Vista-Controlador). Spring nos ofrece una manera muy sencilla de crear un controlador y suficientes herramientas para gestionar el modelo y las vistas.
- Posee una comunidad muy desarrollada y muy activa, por lo que buscar ayuda es relativamente sencillo, incluso en castellano.
- Posee un gran número de aplicaciones para ayudar al desarrollo, normalmente son de uso muy sencillo.
- Es gratuito.

El entorno de desarrollo con Eclipse, MySQL y Tomcat me ha facilitado mucho la labor al estar muy bien integrado y tener una amplia comunidad de usuarios que me ha ayudado a resolver rápidamente pequeños problemas de configuración.

4.2. Diseño

La utilización de patrones de diseño ha sido muy útil a la hora de realizar el proyecto.

El Modelo-Vista-Controlador ha simplificado la aplicación, al separar la lógica de la presentación y los datos. Salvo excepciones, las páginas jsp contienen un código mínimo. De todas maneras, al no ser un experto en Spring, hemos cometido el error de hacer un único controlador para toda la aplicación, lo que en mi opinión, lo ha complicado en exceso. Si hubiera utilizado Struts, quizá no hubiera cometido este fallo, ya que al ser más rígido, te obliga a pensar mejor la lógica.

El patrón DAO hace más sencillo el acceso a la base de datos, si bien requiere más trabajo para crear el juego de clases que dan acceso a la base de datos, aunque después simplifica mucho el

código. Me ha resultado trabajoso al principio, pero a la hora de utilizar el código de acceso a la base de datos en la aplicación, ha resultado ser extremadamente sencillo y eficaz. Merece la pena dedicar tiempo para crear las clases necesarias, ya que todo desarrollo posterior se simplifica sobremanera.

La idea principal del proyecto es la creación de un pequeño gestor de contenidos para hacer más fácil la actualización de un sitio Web. Si bien a lo largo de la implementación, he comprobado que es más complejo que la realización de una simple página Web, también he visto que es más fácil para la persona que aporta los contenidos, ya que no necesita unos conocimientos avanzados en la edición de páginas Web.

Por último, y respecto al ciclo de vida del proyecto, he tenido que replantear parte del mismo al ver que la complejidad de las transacciones en este tipo de actividad comercial, son demasiado complejas para el alcance de este trabajo. Al plantear originalmente el proyecto pensé que sería sencillo hacer una aplicación para la venta de productos, pero tras hablar con el responsable de la empresa y comprender la complejidad en las ventas de empresa a empresa, decidí dejarlo lo más abierto posible para no limitar la capacidad del sitio Web.

4.3. Posibles ampliaciones

Aunque las ampliaciones son múltiples, creo que las siguientes son las más interesantes por orden decreciente.

- **Creación de un sistema de caché**: A medida que crece el tráfico de una Web, se hace imprescindible establecer una caché que genere los trozos de código HTML para no ralentizar el servidor cargándolo con consultas pesadas a la base de datos. Spring tiene un módulo para el establecimiento de un sistema de caché. Aunque un sitio Web de una PYME de estas características no va a soportar un tráfico muy elevado, si que es cierto que al utilizar un sistema de caché vamos a evitarle al usuario el acceso a la base de datos, reduciendo así el tiempo de espera de una página.
- **Sistema de búsqueda**: Obviamente, un gestor de contenidos está justificado con un volumen de información elevado. No tiene sentido en un sitio pequeño sin contenidos que apenas varíen. También parece necesario, que al manejar un volumen elevado de

datos, se haga necesaria la indexación de los contenidos para poder buscarlos fácilmente, tanto a nivel de usuario como de administración. Para responder a estas necesidades se podría incorporar un motor de búsqueda como Lucerne, incluido en un módulo de Spring.

- **Sistema de estadísticas de acceso al sitio Web**: Es importante conocer las visitas que se producen a la página Web y sobre todo, cuales son los productos más solicitados. Para ello, podríamos implementar fácilmente un sistema como Google Analytics, que estudia todos los accesos a un sitio concreto y genera estadísticas de acceso. Aunque quizá habría que desarrollar uno a medida, ya que se ha empezado en algunos países de la Unión Europea a multar su uso por el posible uso fraudulento que se le puede dar a los datos recogidos.
- **Sistema de auditoría**: Asociada a la autenticación es importante saber quien ha creado, modificado o borrado contenidos. También es útil para poder deshacer los cambios hechos guardándolos, por ejemplo, en la base de datos (se podrían guardar el estado del objeto antes y después del cambio en formato XML). No es una ampliación demasiado importante ya que el número de administradores, en este caso concreto, no es elevado. En cuanto aumentase el número de administradores o de creadores de contendidos, sería conveniente implementarlo, ya que las posibilidades de cometer errores con los contenidos aumenta.
- **Creación de un sistema de autenticación**: Creo que sería interesante desarrollar un sistema de autenticación para poder establecer quien tiene permiso para crear, publicar, editar o eliminar contenidos. También sería interesante a la hora de autenticar a los usuarios y clientes que visiten la página Web. Spring, también incorpora un módulo para la autenticación. Esta ampliación aparece en último lugar, ya que con la seguridad de Tomcat bastaría para proteger la aplicación de administración y el número de administradores raramente pasará la media docena. Para los usuarios bastaría añadir el módulo de Spring en cuanto se decidiera como hacer los pedidos a través de la página Web.

4.3. Conclusiones finales

A nivel personal, he encontrado gratificante desarrollar este trabajo por tres razones:

- La posibilidad de poner en practica los conocimientos adquiridos a lo largo de la carrera: He tenido la oportunidad de hacer uso de muchos de los conocimientos aprendidos a lo largo de la carrera a la hora de desarrollar el trabajo. Estos son aquellos conocimientos que en el trabajo a veces aplicamos de manera inconsciente, pero que después de haberlos estudiado, comprendemos la verdadera dimensión y complejidad de todo lo que envuelve a un proyecto Web.
- La posibilidad de conocer a fondo la actividad de una empresa real y ayudar de manera directa a un familiar: Recientemente mi hermano perdió su trabajo a causa de la crisis. En ese momento decidió montar la empresa que ha inspirado este proyecto. En lugar de hacer un proyecto típico de carrito de la compra, decidí hacer este proyecto al oír a mi hermano quejarse de lo caro que era imprimir catálogos y la imposibilidad de mostrar todos los productos por restricciones de tiempo al cliente. Estos factores inspiraron este trabajo y espero que a raíz de él, se beneficie mi hermano y otras pequeñas empresas.
- La posibilidad de estudiar una tecnología actual del mercado: Como desarrollador con varios años de experiencia es relativamente fácil quedarse estancado en una tecnología que poco a poco va quedándose obsoleta. Gracias a este trabajo, he tenido la ocasión de conocer una nueva tecnología como Spring, considerada como puntera en el sector actualmente y con una amplia oferta laboral.

Por último, aunque es algo que hemos ido constatando a lo largo de estos semestres en la UOC, he comprobado, como es posible estudiar una carrera a través de Internet. En este apartado quería dar las gracias a mi consultor, Oscar Escudero Sánchez, por su ayuda y consejo y la rapidez y honestidad de sus respuestas.

5. Bibliografía

La fuente principal de bibliografía en este proyecto ha sido la Web de Spring (http://www.springsource.org/). En ella he encontrado toda la documentación referente al uso de Spring, así como ejemplos prácticos para ayudar en la implementación. También su foro ha respondido a muchas de las preguntas y problemas que he ido encontrando.

También he consultado la API de Java para despejar dudas referentes al código Java (http://download.oracle.com/javase/6/docs/api/overview-summary.html).

Todo lo relativo a patrones de diseño lo he extraído de **J2EE Design Patters** de *Crawford* y *Kaplan*. (2003 Editorial O'Reilly).