TRABAJO DE FIN DE CARRERA

TÍTULO DEL TFC: Prototipo de aplicación para la gestión de facturación

TITULACIÓN: Enginyeria Técnica de Telecomunicaciones, especialidad en Telemática

AUTORES: Luis Alcalde Solanilla
Ivan Fontanals Pérez

DIRECTOR: Roc Mesaguer Pallarès

FECHA: 15 de octubre de 2009

TÍTULO: Prototipo de aplicación para la gestión de facturación.

AUTORES: Luis Alcalde Solanilla
Ivan Fontanals Pérez

DIRECTOR: Roc Mesaguer Pallarès

FECHA: 15 de octubre de 2009

Resumen

El presente Trabajo de Fin de Carrera pretende hacer el seguimiento de la construcción de una aplicación destinada a un entorno web y que cubra las necesidades de un gestor de facturación.

El proyecto tiene su punto de partida en la definición de la estructura teórica del aplicativo. Para lograr dicho cometido se realiza un estudio de requisitos tanto a nivel funcional como no funcional y se imponen una serie de hitos y objetivos finales.

La segunda fase tiene un punto de vista formativo. En este periodo se adquieren conocimientos de bases de datos y en especial del modelo entidad-relación. Por otra parte, se realiza una familiarización con diferentes tecnologías que se usarán en el proyecto, con Struts e Hibernate como principales protagonistas. Una vez finalizado el aprendizaje y conociendo que rol ocupará cada tecnología se decide la arquitectura definitiva del sistema y se diseña la base de datos de la aplicación.

Durante el periodo de implementación, los diferentes módulos que contiene el aplicativo serán construidos y unificados. Será necesario también validar el correcto desarrollo de los módulos con pruebas y verificaciones de requerimientos.

El resultado final es una aplicación que da vida a un gestor de facturación y que funciona siguiendo una arquitectura web con un servidor y uno o varios clientes que pueden hallarse en una intranet o en Internet. En resumen, este proyecto permitirá seguir la evolución de la construcción del sistema, así como familiarizarse con distintas herramientas usadas en el desarrollo de aplicaciones web como Java, Struts, Hibernate, Spring o SSL.

TITLE: Prototype application of a billing manager

AUTHORS: Luis Alcalde Solanilla
Ivan Fontanals Pérez

DIRECTOR: Roc Mesaguer Pallarès

DATE: 15th October 2009

Overview

This project (Trabajo de Fin de Carrera) pretends to monitories the construction of a billings manager application for a web environment.

The work begins with the definition of the theoretical structure of the application. The target will be achieved by doing an analysis of the requirements with a functional study as well as a non-functional one. Also different milestones and final objectives will be decided at this point.

The second phase has a formative purpose. In this period data base concepts will be learned specially those related with the Entity Relationship model. Furthermore, it is going to be made a learning of the different technologies involved in the project, with Struts and Hibernate as main protagonists. Before finishing this period and once it is known the role played by the technologies, the final architecture will be stipulated and the database will be designed.

During the implementation period, the different modules of the program will be built and merged. Additionally it is necessary to validate the correct development of the modules with several tests and a verification of the requirements.

The final result is a billing manager application involved in a web architecture with server and hosts, situated in a intranet o the Internet. To sum up, this project will permit to follow the evolution of the construction of the system as well as do a familiarization with the different kind of tools used in the development of web application as Java, Struts, Hibernate, Spring or SSL.

ÍNDICE

CAPÍTULO 1. INTRODUCCIÓN

El presente Trabajo de Fin de Carrera tiene como principal propósito recoger los requisitos de una aplicación, diseñarla, llevar a cabo su implementación y finalmente comprobar que cumple con las especificaciones iniciales.

Se toma como referencia una aplicación de gestión de facturación de una empresa, que es un buen punto de partida para trabajar en los objetivos iniciales. Es necesario tener presente que se presentará en un entorno web, dónde la información ha de ser consultada y manipulada de forma intuitiva y los datos deben ser tratados de la forma más eficiente posible para no malgastar los recursos.

El documento actual se estructura para separar en distintos niveles el proceso de diseño y construcción de la aplicación e intentar profundizar en cada uno de ellos.

En el primer capítulo se realizará una descripción del proyecto y se expondrán las motivaciones y objetivos de éste. En el segundo capítulo se hará una descripción del sistema a desarrollar, se analizarán los requisitos funcionales y no funcionales del mismo y se detallarán los diferentes casos de uso. El tercer capítulo tiene un objetivo más teórico y será el encargado de introducir el escenario bajo el cual se construirá la aplicación y las herramientas que serán necesarias para desarrollarlo y realizar las pruebas.

Una vez introducido todo el marco teórico, profundizaremos en el diseño del aplicativo y se realizará un minucioso detalle de la implementación, entrando en detalle de las tecnologías desplegadas. Desde el punto de vista de desarrollo finalizaremos con las pruebas de la aplicación y se verificará que se haya cumplido los objetivos propuestos.

El último capítulo de la memoria se reserva a las conclusiones finales, haciendo hincapié en los resultados obtenidos, realizando balance de las herramientas usadas y definiendo diferentes puntos de ampliación y mejora del aplicativo final.

En los anexos podremos encontrar aquella información que complemente el escrito tal como diagramas y extractos de la implementación.

CAPÍTULO 2. DESCRIPCIÓN DEL PROYECTO

2.1 Razón y oportunidad del proyecto

Las aplicaciones web son herramientas que han cogido mucho protagonismo en los últimos años. En un escenario mundial cada vez más global y con empresas cada vez menos centralizadas, la necesidad hacer llegar la información de forma simple a fuentes cada vez menos estáticas se vuelve una necesidad.

Haciendo uso de una red como Internet se consigue que con un mero navegador cualquier persona pueda acceder a unos datos almacenados en cualquier parte del mundo. Además este tipo de aplicaciones son válidas incluso en una Intranet porque siguen proporicionando un escenario simple.

Aplicaciones basadas en SAP, Siebel o una infinidad de lenguajes Host pierden por sus propias limitaciones terreno a la hora de facilitar un escenario más dinámico y escalable, en contraposición lenguajes como Java ganan terreno a diario.

Además hay que englobar esta evolución a un escenario marcado por las limitaciones económicas y se puede comprobar la creciente tendencia al uso de software libre para reducir el precio de los proyectos. En este sentido cabe remarcar que todas las tecnologías usadas en este seguirán esta tendencia.

Por tanto, la construcción de una aplicación de gestión de datos que siga las premisas mencionadas anteriormente, puede ser un buen punto de partida para empresas que requieran una herramienta para controlar su facturación.

2.2 Descripción general del proyecto

El módulo de facturación, es un servicio que puede ser válido para una infinidad de empresas y será usado por sus empleados con el fin de tener un control estricto del producto que pasa por sus manos.

Esta herramienta deberá ser accesible tanto desde una intranet como desde Internet. Los ususarios que accederán a ella se definirán según unos roles estipulados por la empresa, que tendrán además asociados unos permisos para realizar acciones.

Se define un aplicativo estándar, eso significa que quien necesite hacer uso de esta herramienta se deberá informar de que funcionalidades ofrece. Por su diseño estará pensado para pequeñas y medianas empresas.

Para hacerse con una idea más cercana a este proyecto, especificamos en este apartado las principales funcionalidades y más adelante entraremos en el detalle.

- Gestión de artículos
- Gestión de clientes
- Gestión de proveedores
- Gestión de empresa, empleados y roles.
- Mantenimiento de datos maestros de la aplicación.

2.3 Objetivos

El principal objetivo de este proyecto es el desarrollo de una herramienta para la gestión de facturación. El proyecto se inicia en la fase de diseño y toma de requisitos, prosigue con la construcción de una aplicación web y finaliza con la verificación de objetivos.

A continuación se detallan algunos de los hitos propuestos para dar dimensión al proyecto.

- **Diseño de la herramienta.** Cómo punto de partida y una vez decidido cúal es el propósito del proyecto se pretende realizar un diseño teórico de la herramienta. Se definen unos requisitos que serán los fundamentos de la solución.
- **Diseño de la base de datos.** Teniendo en mente la aplicación de facturación es necesario realizar un modelo de esta para la creación de una base de datos que de opción a albergar la información. En este sentido nos hemos decantado por un modelo entidad relación.
- **Familiarización con las tecnologías.** Definido el diseño y la base de datos hay que realizar una primera toma de contacto con las herramientas que se usarán en la implementación del código. En este sentido, será necesario realizar un período de aprendizaje para adquirir los conocimientos suficientes para evolucionar el proyecto. El framework Struts, Spring o Hibernate son los protagonistas de esta fase.
- **Implementación.** La construcción de las diferentes funcionalidades del aplicativo (Gestión de artículos, gestión de clientes…). Con tal de obtener un sistem ágil se hará un uso eficiente de los recursos, se pretende realizar una correcta implantación de las tecnologías y al mismo tiempo se ofrecerá una interfície gráfica bien estructurada que facilite al máximo el trabajo del usuario.
- **Pruebas y comprobación de resultados.** Para finalizar se debe verificar la consecución de objetivos.

CAPÍTULO 3. ANÁLISIS Y ESPECIFICACIONES

En este capítulo se realizará un análisis teórico del sistema a desarrollar. Se hará una primera descripción del funcionamiento del sistema y a continuación se detallarán tanto los requisitos funcionales del sistema como los no funcionales.

3.1 Descripción del funcionamiento del sistema

La herramienta que se plantea contruir en este proyecto pretende cubrir la necesidad del control de gestión de producto y facturación para empresas que tengan un rol de intermediario, ya sea entre empresas o entre una empresa y un cliente final.

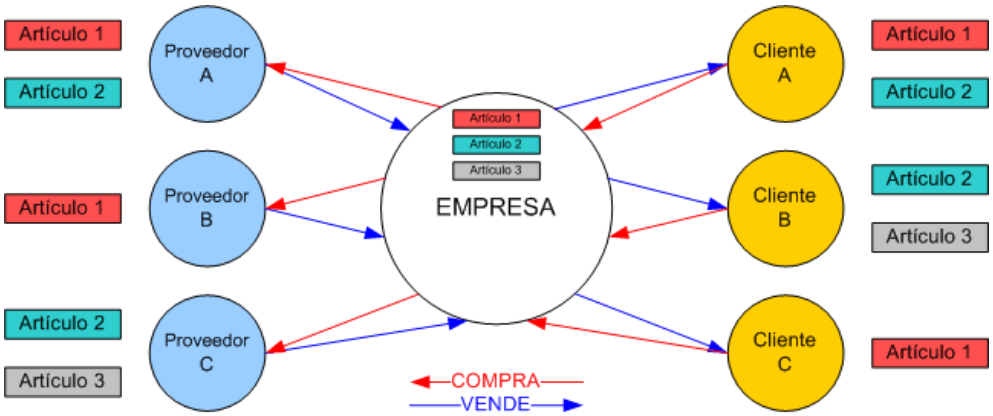


Fig 3.1 Ciclo de funcionamiento del sistema

En la figura 3.1 podemos observar el ciclo que sigue la compra-venta de producto. La empresa para la cual se diseña la aplicación se nutrirá de diferentes empresas proveedoras, las cuales le suministrarán diferentes artículos. La empresa almacenará el producto y este se verá reflejado en la aplicación. El cliente será el encargado de comprar los artículos.

Así pues, la herramienta se crea con la finalidad de poder gestionar por una parte los artículos entrantes. Por otra tener controlados los artículos, pudiendo dar de alta nuevos artículos y cambiarles el precio. Por último se permitirá también gestionar la venta de producto a los clientes.

Al mismo tiempo será necesario un módulo para gestionar todos los datos maestros, ya sea para definir formas de envío, de pago, datos de direcciones...

Por último, se tendrá en cuenta el rol del empleado para conocer que acciones puede hacer sobre el proceso. Un encargado de compras no debería tener

acceso al módulo de ventas. Por eso se implementará un sistema de control de empleados, cargos y permisos para la aplicación.

Hecha esta breve descripción del sistema análizamos más en detalle los requisitos.

Existen dos tipos de requisitos.

- **Funcionales**. Describen el comportamiento que ha de adoptar el sistema. Que acciones ha de desarrollar, como las debe llevar a cabo y que resultado se debe dar para cada una de ellas.
- **No funcionales**. Especifican cómo se debe comportar el sistema en términos de rendimiento, costo, escalabilidad… Son por tanto requisitos que no entran en la funcionalidad, pero son imprescindibles para construir una aplicación efectiva.

3.2 Análisis de requisitos funcionales

El sistema estará compuesto por diferentes módulos. Para analizar los requisitos funcionales se detallará cada uno de estos módulos mediante la explicación de lo que ofrecen, acompañados de sus casos de usos y las especificaciones de estos últimos.

- Gestión de Artículos
- Gestión de Clientes
- Gestión de Proveedores
- Gestión de Empresa
- Gestión de Mantenimientos

3.2.1 Gestión de artículos

El sistema debe permitir la gestión de artículos. Se entiende como artículo el producto que se obtiene de un proveedor y se debe vender a un cliente. Mediante una interficie amigable se deberá poder consultar, crear, modificar, inactivar, activar y detallar los artículos.

Se detallan artículos para poder diferenciar si un artículo proviene de diferentes proveedores y en que cantidad y precio.

En esta sección también se podrá controlar las famílias bajo las cuales se agruparan los artículos y los porcentajes de IVA que se aplican a estos últimos.

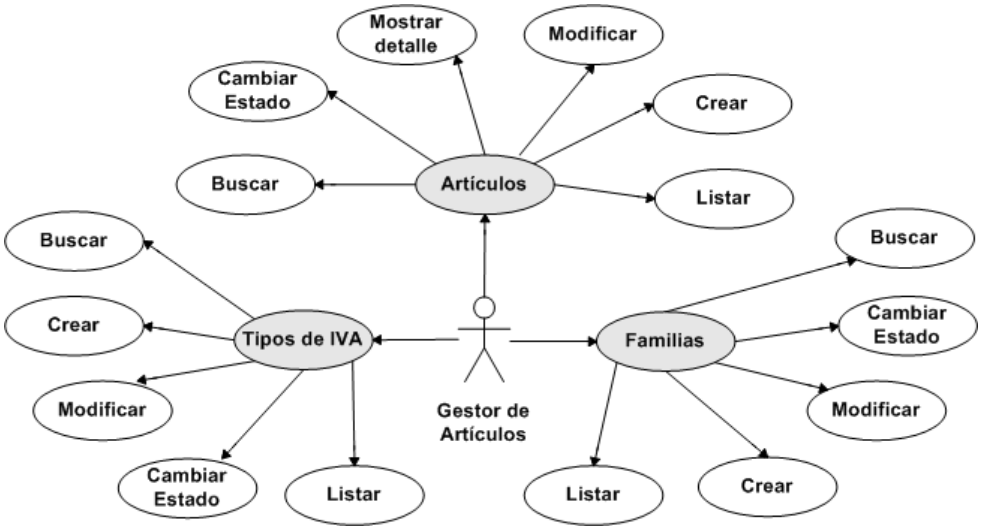


Fig 3.2 Diagrama de casos de uso Gestión de Artículos

3.2.2 Gestión de clientes

El sistema debe permitir la gestión de clientes. Se entiende como cliente el destinatario del artículo.

Mediante una interficie amigable se deberá poder listar, crear, modificar, activar e inactivar clientes. Al mismo tiempo se debe permitir crear presupuestos asociados al cliente. Estos también se podrán listar y modificar. De cada presupuesto se podrá generar una factura relativa a la venta del producto a un cliente. Las facturas se podrán tanto generar, consultar como listar. Una vez listada la factura podrá ser imprimida por el usuario.

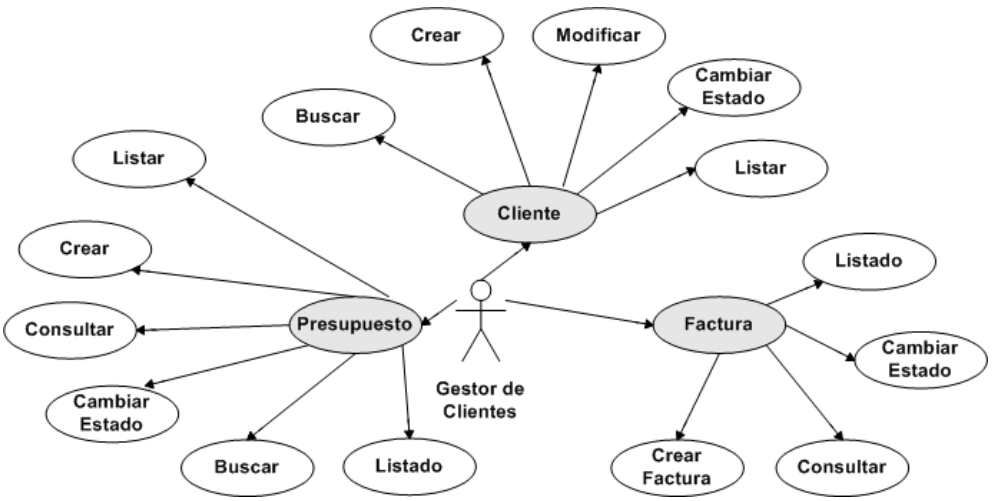


Fig 3.3 Diagrama de casos de uso Gestión de Clientes

3.2.3 Gestión de proveedores

El sistema debe permitir la gestión de proveedores. Se entiende como proveedor quien nos proporciona los artículos.

Mediante una interficie amigable se deberá poder listar, crear, modificar, activar e inactivar proveedores. Cada vez que se quiera obtener un artículo se generará un pedido. Este podrá ser consultado y su estado cambiará cuando se rechace o se genere la factura. Todos los pedidos podrán ser consultados en un listado. Por cada pedido se deberá generar una factura cuando se realice la compra del producto. La factura podrá ser generada, consultada y listada para su impresión.

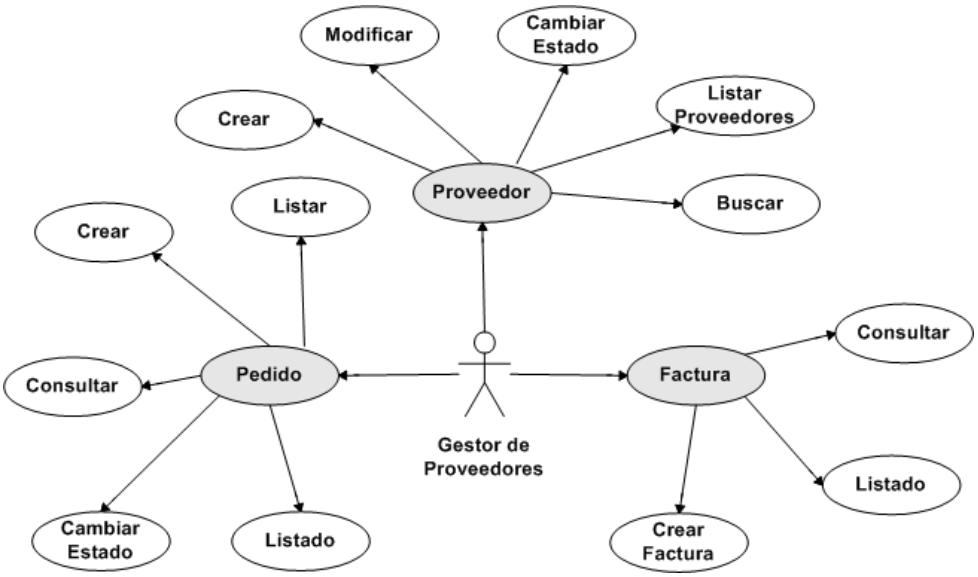


Fig 3.4 Diagrama de casos de uso Gestión de Proveedores

3.2.4 Gestión de empresa

La gestión de empresa se centra en los empleados de esta y que son los usuarios potenciales de la aplicación.

Mediante una interficie amigable se podrá consultar y modificar los datos de empresa. Exisistirá también una relación de empleados y otra de cargos. Ambas se podrán listar, crear, modificar, activar, inactivar. Si un empleado se inactiva perderá todos los permisos de la aplicación.

A cada usuario se le pueden asignar individualmente los permisos de la aplicación. En el apartado de Perfiles lo veremos más en detalle.

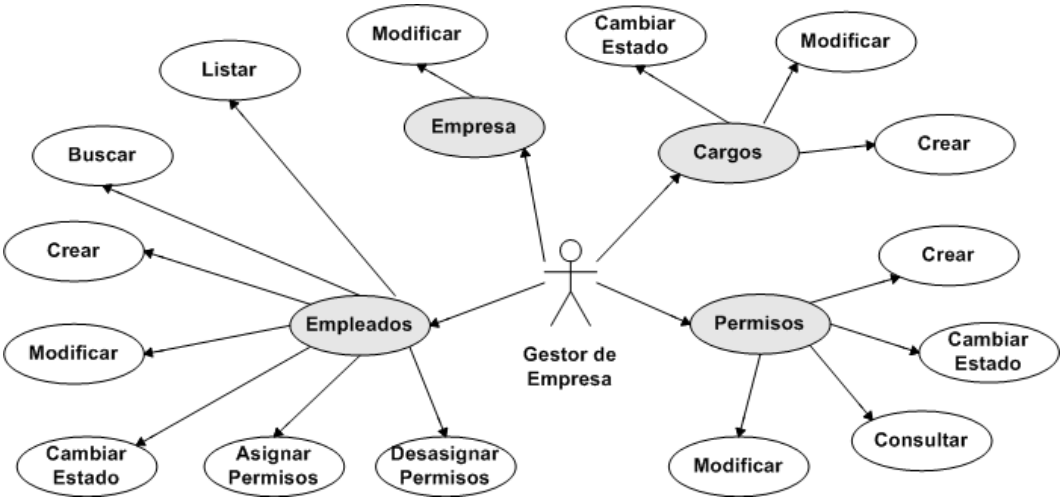


Fig 3.5 Diagrama de casos de uso Gestión de Empresa

3.2.5 Gestión de mantenimientos

La gestión de mantenimientos concentra diferentes tipos de datos maestros de la aplicación.

- Poblaciones
- Provincias
- Países
- Formas de envío
- Formas de pago

Por cada uno de ellos se podrán listar los registros, así como buscarlos, crearlos, modificarlos, activarlos y desactivarlos.

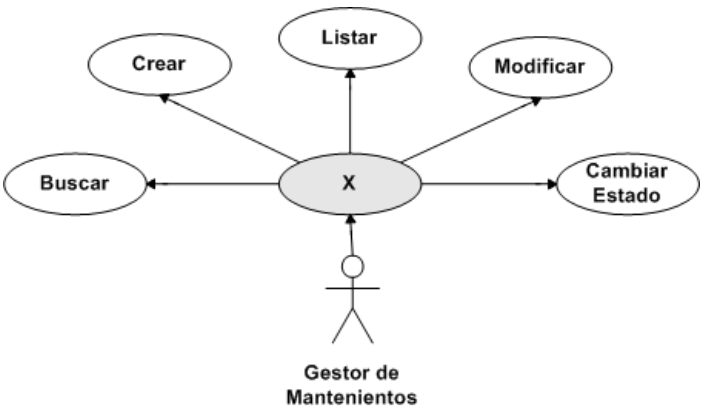


Fig 3.6 Diagrama de casos de uso Gestión de Mantenimientos

3.2.6 Otros casos de uso

Existen otros casos de uso que debe contemplar el sistema. Por una parte se realizará un acceso a la aplicación que permita autenticar al usuario. Por otra se permitirà una desconexión controlada para que los datos almacenados en sesión puedan ser eliminados, dando un plus de seguridad al programa.

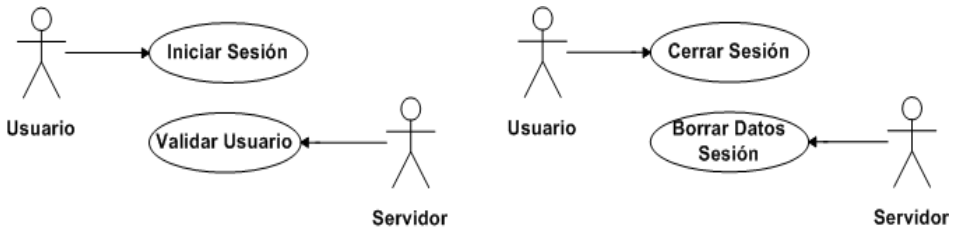


Fig 3.7 Diagrama de casos de uso registro (izq.) y desconexión (der.)

3.2.7 Perfiles

En la mayoría de los casos de uso analizados hasta el momento se hace referencia a un gestor. Es un término teórico que se ha definido para poder realizar un diseño más modular del aplicativo, pero no tiene relación directa de los permisos que los usuarios tienen sobre el programa. Los permisos serán individuales por usuario y se establecerán a nivel de acción, de forma de que puede darse el caso de que un usuario pueda consultar artículos pero no crearlos. Esta forma de definir autorizaciones permite definir de forma detallada el comportamiento de un usuario.

Habrá un usuario excepcional. Para poder realizar tareas de mantenimiento de forma sencilla se establece que cualquier empleado con cargo de administrador no tendrá limitaciones a nivel de autorizaciones.

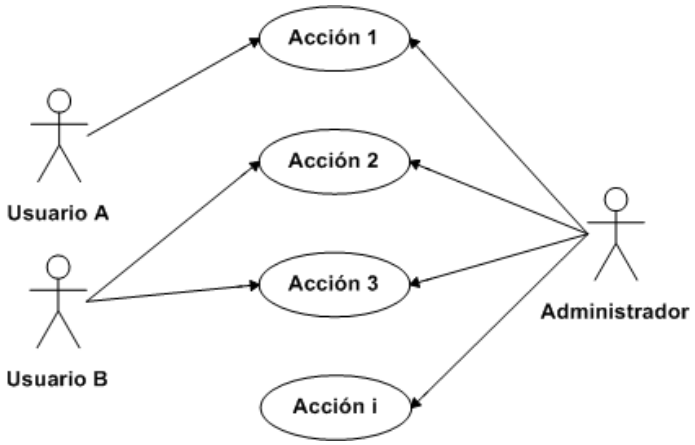


Fig 3.8 Diagrama de casos de uso de posibles perfiles y de administrador

3.2.8 Especificación de los casos de uso

En el apartado de requisitos funcionales se ha definido el modelo de casos de uso de cada módulo gestor. Ahora veremos desde un punto de vista genérico algunos de los diferentes casos de uso. En los anexos podremos hallar el resto.

3.2.8.1 Acceso a la aplicación

- Caso de uso: Acceso a la aplicación.
- Actores: Gestor y Servidor.
- Propósito: El gestor accede al sistema y puede utilizar la aplicación
- Resumen: El usuario puede acceder según el rol a los diferentes apartados de la aplicación.
- Tipo: Primario y esencial.

Tabla 3.1 Curso tipo del acontecimiento

Acciones del Gestor	Respuesta del Sistema
1. Este caso de uso empieza cuando un usuario quiere utilizar la aplicación	
	2. El sistema comprueba que los datos introducidos por el usuario sean correcto y a que parte tiene acceso
	3. El sistema muestra la pantalla principal

Existe un curso alternativo. Error en el caso que los datos no sean los correctos, el sistema devuelve un error de usuario / contraseña.

3.2.8.2 Desconexión de la aplicación

- Caso de uso: Desconexión de la aplicación
- Actores: Gestor y Servidor.
- Propósito: El gestor quiere cerrar la sesión de la aplicación.
- Resumen: El usuario quiere salir de la aplicación y se desea eliminar cualquier rastro en de los datos de sesión.
- Tipo: Primario y esencial.

Tabla 3.2 Curso tipo del acontecimiento

Acciones del Gestor	Respuesta del Sistema

1. Este caso de uso empieza cuando un usuario selecciona el botón de cerrar sessión	
	2. El sistema elimina los datos de la sesión del usuario.
	3. El sistema navega a la pantalla de registro

3.2.8.3 Crear

- Caso de uso: Crear
- Actores: Gestor y Servidor.
- Propósito: El gestor quiere crear una nuevo concepto / objeto.
- Resumen: El usuario puede dar de alta, con validación de la información incluída.
- Tipo: Primario.

Tabla 3.3 Curso tipo del acontecimiento

Acciones del Gestor	Respuesta del Sistema
1. Este caso de uso empieza cuando un usuario selecciona el botón de “Nuevo” de cualquier concepto	
	2. El sistema muestra la pantalla de concepto solicitado por el usuario con los campos vacíos
3. El usuario introduce los datos que son necesarios, para dar de alta un concepto	
	4. El sistema comprueba que los datos introducidos por el usuario sean correcto.
	5. El sistema hace un insert en la base de datos con la información que ha introducido el usuario
	6. El sistema vuelve a mostrar la lista de todos los conceptos que hay en la B.D.

Existe un curso alternativo. Error en el caso que los datos no sean los correctos, la aplicación devuelve un error informando el campo incorrecto.

3.2.8.4 Inactivar estado

- Caso de uso: Inactivar estado
- Actores: Gestor y Servidor.
- Propósito: El gestor quiere modificar el estado de cualquier concepto.

- Resumen: El usuario puede dar de baja cualquier concepto de cualquier tipo de mantenimiento, para que no pueda ser utilizado en la aplicación.
- Tipo: Primario.

Tabla 3.5 Curso tipo del acontecimiento

Acciones del Gestor	Respuesta del Sistema
1. Este caso de uso empieza cuando un usuario clicka en el botón de “Modificar” de cualquier concepto	
	2. El sistema muestra la pantalla con el concepto solicitado por el usuario con la información que hay en la bases de datos
3. El usuario clicka en el botón de “Inactivar” el concepto	
	4. El sistema hace un update en la base de datos con el estado = 99
	5. El sistema vuelve a mostrar la lista de todos los conceptos que hay en la B.D.

3.2.8.5 Buscar

- Caso de uso: Buscar concepto
- Actores: Gestor y Servidor.
- Propósito: El gestor busca la información de cualquier concepto.
- Resumen: El gestor puede buscar cualquier concepto a partir de introducir unos datos mínimos, para que el servidor muestre una lista con los datos que cumplan esos requisitos.
- Tipo: Secundario.

Tabla 3.7 Curso tipo del acontecimiento

Acciones del Gestor	Respuesta del Sistema
1. Este caso de uso empieza cuando un usuario selecciona el botón de “Buscar” de cualquier concepto	
	2. El sistema muestra la pantalla con los datos que el usuario puede introducir para acotar la busqueda
3. El gestor introduce los datos que conoce del concepto	
	4. El sistema a partir de los datos introducidos por el gestor hace una SELECT para buscar todos los conceptos que cumplan con los

	requisitos
	5. El sistema muestra una lista con todos los conceptos
6. El gestor puede seleccionar el concepto por el cual a realizado la búsqueda	

3.2.8.6 Listado de concepto

- Caso de uso: Listado de concepto
- Actores: Gestor y Servidor.
- Propósito: El gestor prepara un listado de la factura.
- Resumen: Una factura puede ser listada para su impresión.
- Tipo: Primario.

Tabla 3.9 Curso tipo del acontecimiento

Acciones del Gestor	Respuesta del Sistema
1. Este caso de uso empieza cuando un usuario selecciona el botón de “Listado” de cualquier factura	
	2. El sistema muestra la pantalla con los datos del listado.

3.3 Análisis de requisitos no funcionales

Tal y como se ha adelantado, estos són implícitos al sistema pero no son funcionalidades concretas de este. A continuación los detallamos.

3.3.1 Calidad

Para el correcto funcionamiento de la aplicación es necesario que esta se mueva en unos márgenes de calidad adecuados. Analizamos algunos de los más importantes.

- **Integridad y seguridad.** Será necesario asegurar que personas ajenas al programa no puedan acceder a él. Para aquellos usuarios que tengan privilegios de acceso habrá que corroborar que solo tengan cobertura en aquellas partes del aplicativo donde los permisos tengan vigencia. Además los datos del programa deberán ir cifrados para dar seguridad al protocolo http.
- **Flexibilidad y portabilidad.** Como estamos hablando de una aplicación genérica debe ser fácilmente adaptable a nuevos escenarios. Por tanto el

software debe ser modificable y ampliable. Además debe ser portable para poder aprovechar partes de la infraestructura del destinatario, como por ejemplo la base de datos.

- **Eficiencia y fiabilidad.** Habrá que diseñar e implementar el aplicativo de forma que no use excesivos recursos y estos además presenten los menos fallos posibles.
- **Interoperabilidad.** Será posible unir el programa a otros, por ejemplo mediante Web Services, sin grandes cambios en el sistema.

3.3.2 Carga

Se establece como requisito no funcional lograr unos parámetros de carga razonables. Por una parte se decide que pueden conectarse un mínimo 20 usuarios a la vez. Eso significa, que dichos usuarios podrán hacer uso del programa sin que el rendimiento se vea notablemente afectado. Otro aspecto a tener en cuenta será el tiempo de respuesta. Si por cada acción que realiza el usuario debe esperar un tiempo demasiado grande, la aplicación dejará de ser ágil i por tanto útil. Se define un tiempo máximo de 10 segundos desde que se hace una petición hasta que recibimos la página de respuesta. Por último, se pretende evitar que el número de errores en el aplicativo sea descontrolado. Como en el caso anterior, si no se limita, acabaríamos teniendo un programa ineficiente e inútil. El rango de errores que se establece como válido es un 1% del grueso de acciones realizadas en Franet.

3.3.3 Coste

Se presenta el coste como un factor importante del desarrollo. Se desea construir una aplicación cuyo coste sea bajo, tanto a la hora de implementarla, como a la hora de mantenerla. En este sentido será necesario encontrar tecnologías de sofware libre para lograr este propósito.

3.3.4 Requerimientos tecnológicos

Se debe tener en consideración que la aplicación debe ser accesible desde cualquier sistema operativo. El sistema usará el protocolo http para conectar los clientes y el sevidor. Por tanto, será necesario una máquina donde situar el servidor y un navegador web en aquellos equipos donde se situen los clientes. El navegador podrá funcionar con cualquier resolución igual o superior a 800 x 600 píxels.

Para lograr el hito de independizar el aplicativo de plataformas o propietarios se hará uso del lenguaje UML (Unified Modeling Language) en el diseño y lenguajes como Java, frameworks como Struts, o soluciones como JavaScript, CSS, etc. en la implementación.

3.3.5 Interfícies

Las pantallas que podrá consultar el usuario deberán ser intuitivas y sencillas de usar. La información deberá ser presentada de forma clara y ordenada. La lógica de navegación deberá ser simple y por tanto los botones tendrán que tener comportamientos poco complejos. En resumen se buscan interfícies amigables y simples.

3.4 Especificación de las pantallas del sistema

En la siguiente imagen (fig. 3.9) podemos observar el diagrama de transición de pantallas simplificado de la aplicación.

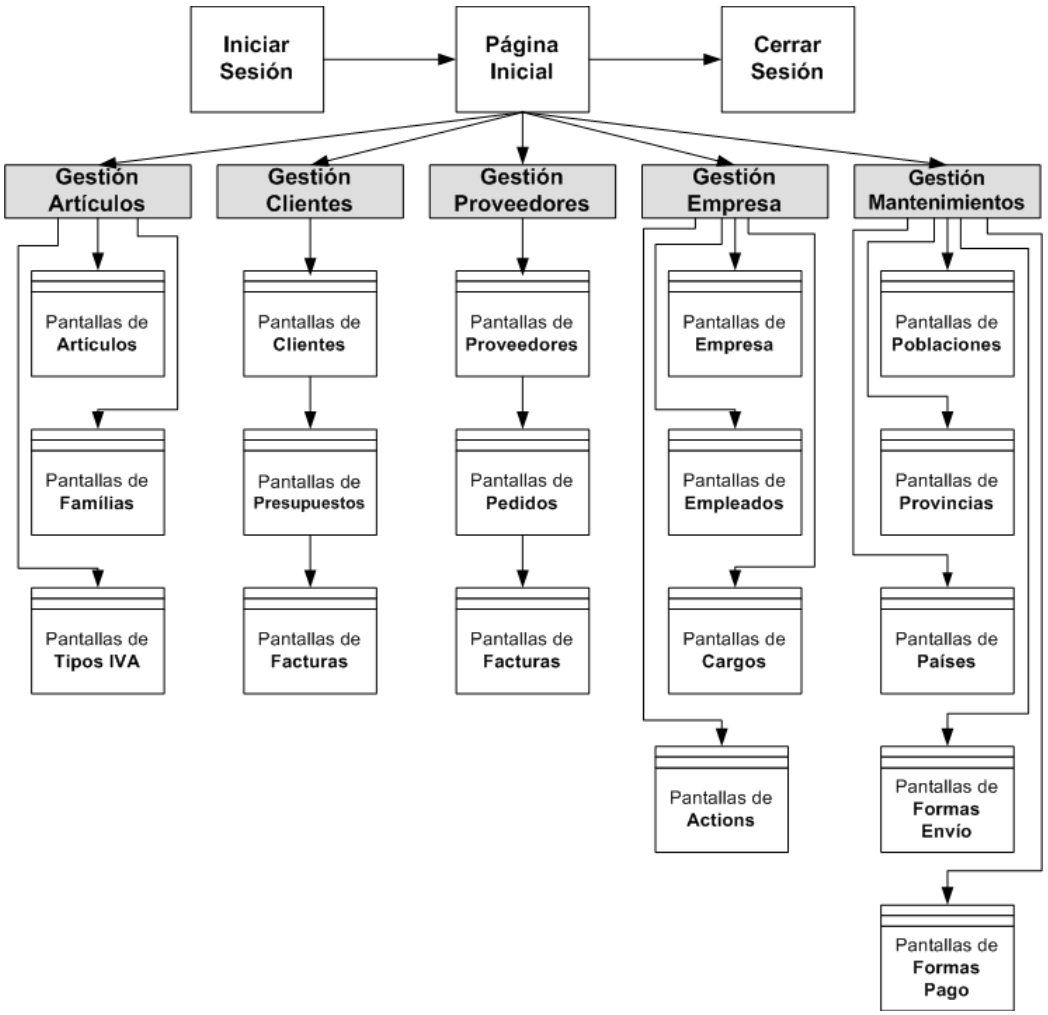


Fig 3.9 Diagrama de pantallas del sistema

Las pantallas no tendrán una estructura estática. Dependiendo de en qué punto de la aplicación estemos se tendrán menos o más secciones, desde un mínimo de una a un máximo de seis. A modo de ejemplo, la pantalla de iniciar sesión tendrá tres porciones (cabecera, cuerpo y pie). A continuación se especifican las diferentes divisiones mediante las cuales se realizarán las combinaciones.

- Cabecera: Imagen con el logo escogido por la empresa.
- Menú: Mediante el cual podremos acceder a los diferentes módulos de gestión.
- Submenú: Para acceder a los diferentes grupos de pantallas dentro de un módulo de gestión.
- Título: Referencia al punto en la aplicación donde se situará el usuario.
- Cuerpo: Contenido que consultará, modificará el usuario.
- Pie: Imagen para distinguir el fin de página.

En la figura 3.10 encontramos dos ejemplos de pantallas. En amarillo se colorean las secciones que no son coincidentes.

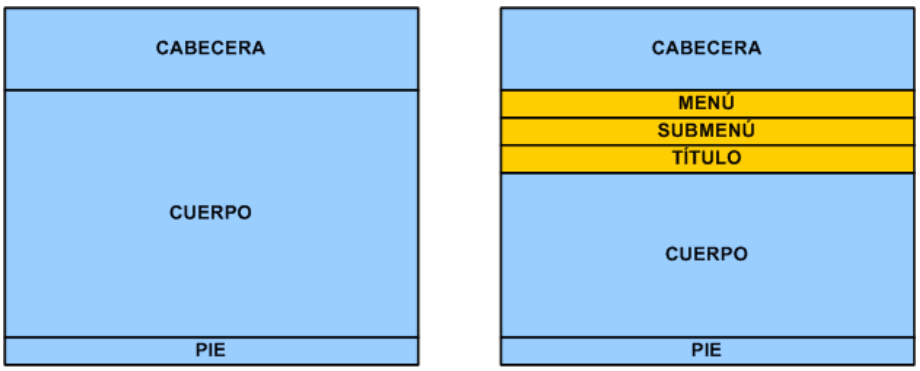


Fig 3.10 Ejemplos de estructura de pantallas

La sección cuerpo estará compuesta por texto, formularios, listados, imágenes, botones… A continuación podemos observar el detalle de dos pantallas, la primera de un listado (Tabla 3.10) y la segunda con un listado y formulario (Tabla 3.11).

Tabla 3.10 Vista listado proveedor

Elemento	Tipo	Comentarios
Listado Pedidos		
Pedidos Proveedor	Título	
ID Pedido	Texto No Editable	

Fecha Vigente	Texto No Editable	Fecha de Creación o Modificación.
Estado	Texto No Editable	
Botones		
Nuevo	Botón	Da acceso a la vista Nuevo Pedido
Consulta	Botón	Da acceso a la vista Consulta Pedido
Generar Factura	Botón	Creará la factura del pedido seleccionado.
Borrar	Botón	Llamará al Action que se ocupará de cambiarle el estado al albarán a anulado.

Tabla 3.11 Vista nuevo detalle del pedido

Elemento	Tipo	Comentarios
Formulario Nuevo Artículo Nuevo Pedido		
Nuevo Pedido	Título	
Añadir Artículo	Título	
ID Artículo	Texto Editable	
Cantidad	Texto Editable	
Botones Nuevo Artículo		
Añadir	Botón	Al seleccionar apareceremos en la misma vista pero habrá un nuevo registro en el listado.
Listado Artículos Nuevo Pedido		
ID Artículo	Texto No Editable	
Cantidad	Texto No Editable	
Botones Listado Artículos Nuevo Pedido		
Eliminar Artículo	Botón	Eliminará del listado el artículo seleccionado.
Formulario Nuevo Pedido		
Observaciones	Texto Editable	
Botones Nuevo Pedido		
Guardar	Botón	Generará un nuevo pedido. Iremos a la vista Listado Pedidos donde habrá el nuevo registro.

En el anexo podremos encontrar los detalles de las pantallas principales del aplicativo.

CAPÍTULO 4. ESCENARIO Y HERRAMIENTAS DE DESARROLLO

Una vez recogidas las especificaciones es necesario definir el escenario bajo el cual funcionará la aplicación. Se aprovechará también para conocer las herramientas mediante las cuales será posible la implementación y en que parte del escenario esta implementación cobrará sentido.

4.1 Arquitectura

El escenario que se pinta en la siguiente imagen (fig. 4.1) es el ejemplo de dos usuarios que se conectarán a la herramienta mediante un explorador desde cualquier portátil o PC conectado a la red. Podemos observar que la aplicación funcionará en un servidor Apache Tomcat y por tanto este servidor recibirá peticiones de los clientes bajo una estructura cliente-servidor típica.

Esta comunicación entre cliente y servidor dispondrá de seguridad mediante el protocolo Secure Sockets Layer (SSL). El servidor estará conectado a su vez a una base de datos que contendrá la información que visualizarán y manipularán los usuarios.

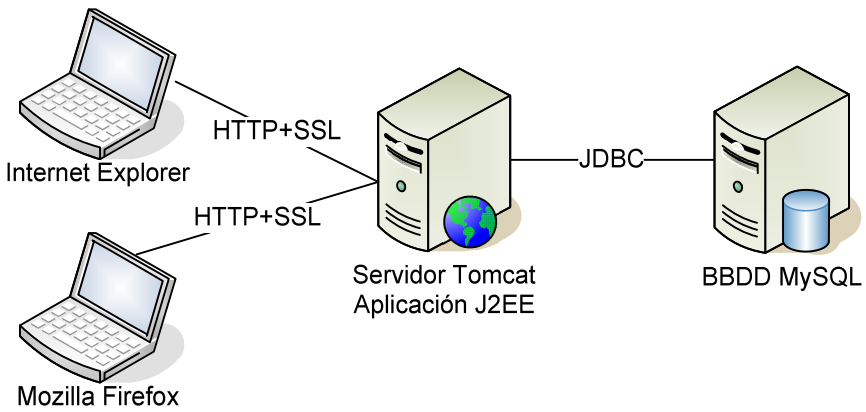


Fig 4.1 Arquitectura Franet

Se opta por una arquitectura cliente-servidor porque permite la intercomunicación con múltiples clientes, se favorece la autenticación del servidor y es la forma más simple de controlar el sistema.

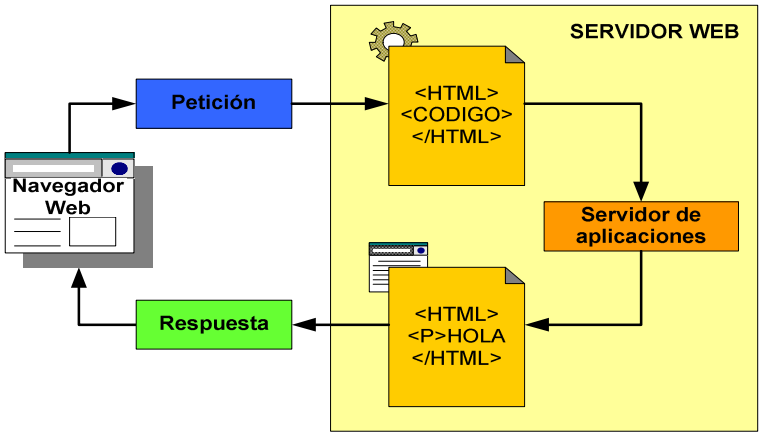


Fig 4.2 Flujo entre Cliente y Servidor

En el explorador se visualizarán páginas de lenguaje HTML que contendrán partes estáticas y dinámicas. Tendrán contenido estático aquellas secciones que siempre sean idénticas y no puedan ser modificables, en contraposición cobrará principal protagonismo el contenido dinámico, que es aquel que el usuario podrá modificar y consultar dependiendo información que proceda de la base de datos.

4.2 Apache Tomcat

El primer pilar de la aplicación web que analizamos es el servidor. Será el encargado de tratar las peticiones del cliente y generar respuestas en consecuencia gracias a una estructura de servlets. En este proyecto el servidor elegido es Apache Tomcat, por ser un servidor libre e implementado en Java lo que permite que pueda funcionar en cualquier sistema operativo que disponga de una máquina virtual Java.

La configuración de este servidor se centra en dos ficheros xml, uno es server.xml y web.xml. El primer fichero tiene como objetivos proporcionar configuración general del servidor mientras que el segundo está más enfocado a la aplicación o aplicaciones que correrán en el servidor, donde se encontrará por ejemplo el servlet para tratar JSPs.

La versión utilizada en este proyecto es Tomcat 6, en el cual viene integrados el soporte para servlets 2.5 y el soporte para JSP 2.1

4.3 Aplicación Java

En el servidor encontraremos el código Java referente a la aplicación de facturación. Se definirá bajo una estructura de aplicación web y se ha implementado haciendo uso de las librerías J2EE.

4.3.1 Módulo web en J2EE

Java 2 Platform Enterprise Edition (J2EE) una plataforma de programación orientada al desarrollo y ejecución de aplicaciones en Lenguaje Java y que estén centralizadas en servidor.

El código de la aplicación se ha estructurado como un módulo web ya que esta forma de trabajar facilita el desarrollo y conceptualmente diferencia los recursos que deben ser compilados de los que no. Entre recursos que no deberán ser compilados encontramos las imágenes, los archivos CSS, los javascript descentralizados, etc. Entre los que sí, contenidos en la carpeta WEB-INF, hallaremos las clases de código Java, librerías, contenedores de etiquetas, etc. Los archivos JSPs serán compilados a posteriori, mientras el programa esté en ejecución.

La figura 4.3 muestra la estructura de carpetas mediante la cual se ha organizado la aplicación.

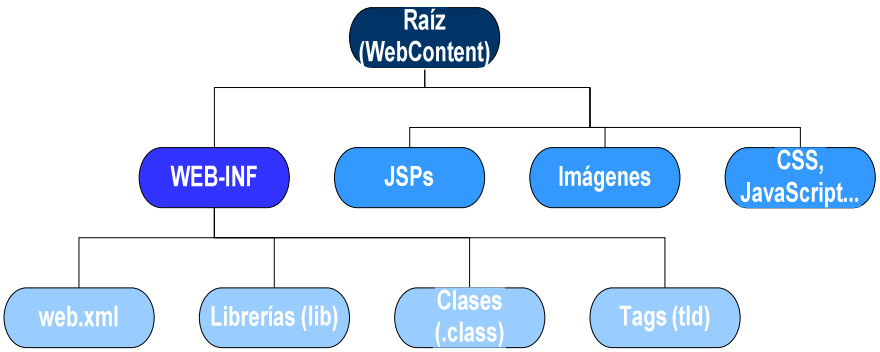


Fig 4.3 Arquitectura carpetas aplicación web Franet

En la carpeta WEB-INF también hallaremos el web.xml. Este es el archivo mediante el cual se definirá el deploy de la aplicación, es decir será el archivo base del aplicativo y en el cual se tendrán que definir una serie de parámetros que analizaremos en el capítulo de implementación.

4.3.2 Struts, Hibernate, Spring, itext…

Se han utilizado un seguido de tecnologías, válidas para escenarios web, para la construcción de la aplicación. En el siguiente capítulo (Capítulo 5. Diseño e Implementación) conoceremos que nos proporcionan y cómo se han llevado a término.

4.3.3 Eclipse

Para desarrollar una aplicación compleja es muy recomendable de hacer uso de alguno de los programas de ayuda a los desarrolladores. Eclipse es una plataforma con módulos base preparados para desarrollar en Java y otros extra encaminados a aplicaciones Web. Su uso es muy recomendable y además es una herramienta libre lo que no añade coste al proyecto. Por último remarcar que gracias a Eclipse y una buena estructuración de los recursos de la aplicación se puede realizar debug y analizar en detalle el código.

4.3.4 Ant

El módulo web que hemos introducido en el apartado anterior se debe situar en la carpeta Webapps del servidor Tomcat. Se puede trasladar como sistema de carpetas o como si fuera un paquete Web. Para esta segunda opción necesitaremos estructurar y englobar la aplicación dentro de un archivo de extensión .war.

Para realizar un empaquetamiento war, solución adoptada en este proyecto, será necesario el uso de la tecnología Ant, que es una herramienta de Apache. Ant hace uso de un archivo de configuración llamado build.xml donde se configurarán todas las instrucciones para construir el war, desde la distribución coherente de los ficheros hasta la compilación de las clases Java.

Una vez se inicia el servidor Tomcat desplegará el paquete y creará el sistema de archivos según su metodología para desplegar la aplicación.

4.3.5 Java SE Development Kit (JDK)

Java SE Development Kit es un software que requieren las máquinas donde se desarrolle o se comile código en Java. En él se incluyen librerías básicas de Java y la máquina virtual. Tanto el servidor Apache, Eclipse y Ant necesitarán de esta plataforma para poder construir y hacer funcionar la aplicación.

4.4 Navegadores

Una de las principales ventajas de las aplicaciones Web es que su uso se facilita mucho gracias a que con un simple navegador Web podemos interactuar con ellas. El navegador en la comunicación será el cliente y será el encargado de lanzar las peticiones al servidor en función de las acciones que tome el usuario y que se transportarán mediante el protocolo HTTP.

Entre los navegadores más populares utilizados en la actualidad, encontramos Internet Explorer, Mozilla Firefox, Opera, Chrome y cualquiera es válido para

acceder a la aplicación construida en este proyecto. Además el navegador puede estar funcionando en cualquier sistema operativo.

Otros beneficios que aportan las aplicaciones Web son la no necesidad de instalar software extra puesto que cualquier ordenador dispone de un navegador y el hecho de que el usuario ya estará familiarizado con él, por lo que el periodo de adaptación será menor.

4.5 Secure Sockets Layer (SSL)

La seguridad en aplicaciones Web es un punto importante a tener en cuenta. Para evitar que la información de clientes, proveedores, empleados viaje de una forma segura es una interesante opción configurar el protocolo SSL.

Aunque se conciba como protocolo único, en realidad está formado por cuatro protocolos, cada uno con un propósito. En el modelo TCP/IP situaremos el protocolo SSL entre la capa de aplicación y la de transporte, lo que permite que cualquier aplicación pueda hacer uso de él y HTTP no será una excepción.

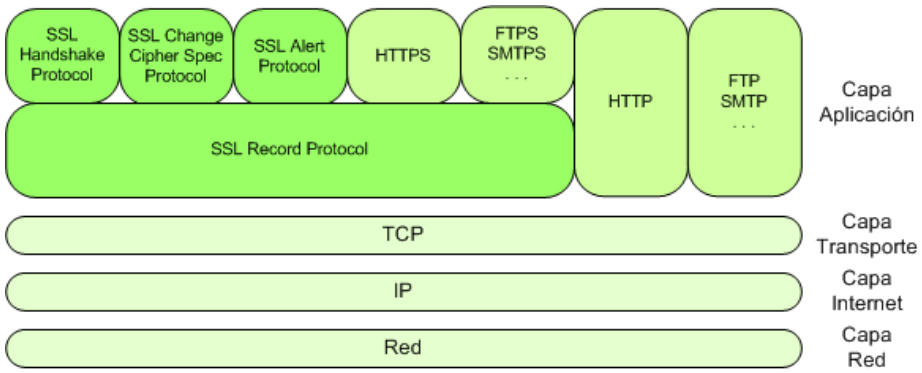


Fig 4.4 Protocolo SSL

Es un protocolo que trabaja de forma transparente para el usuario y según la aplicación que esté en uso se establecerá un puerto de conexión. La IANA asigna el puerto 443 para la versión segura de http. Podremos comprobar que nos conectamos a un puerto seguro porque en el la ventana del navegador cuando aparecerá una URL iniciada con https y un icono con un candado (fig. 4.5).

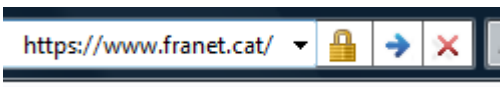


Fig 4.5 Seguridad habilitada en el navegador

El objetivo último de SSL es proporcionar una transmisión segura entre cliente y servidor proporcionando:

- **Autenticación.** Mediante certificados al establecer una sesión. El cliente podrá asegurarse que se conecta al servidor correcto.
- **Confidencialidad.** No se podrá obtener información en claro. Datos cifrados durante la sesión.
- **Integridad.** Gracias a la comprobación de un mensaje hash. Los datos no podrán ser alterados entre ambas máquinas.

El protocolo que proporciona seguridad e integridad es el SSL Record Protocol. Los protocolos encargados de establecer y mantener la sesión serán el SSL Handshake Protocol, el SSL ChangeCipherSpec Protocol y el SSL Alert Protocol. En el anexo se entra en el detalle de dichos protocolos.

4.5.1 Keytool

Con tal de implementar el SSL es necesario hacer uso de una herramienta Java incluída en Windows destinada a gestionar un almacén de claves (*keystore*). El almacén de claves es una base de datos protegida que contienen claves y certificados para un usuario o grupo de usuarios. El acceso al mismo está protegido por un sistema de contraseñas. La herramienta keytool permite:

- Crear pares de claves (pública y privada).
- Emitir solicitudes de certificados (que se envían al CA apropiado).
- Importar replicas de certificados (obtenidos del CA contactado).
- Designar claves públicas de otros como de confianza.

La utilización de esta herramienta en nuestro proyecto tiene el fin de dotarnos de un certificado autofirmado para llegar a implementar el protocolo SSL en nuestro servidor web.

4.6 Base de datos MySQL

Para la creación y administración de la base de datos, se han utilizado las herramientas propias del MySQL, como son MySQL Administrador y MySQL Query Browser. MySQL es el más indicado para aplicaciones que requieren mucha lectura y poca escritura y que no necesiten de características muy avanzadas, como es el caso de las aplicaciones web.

Las principales características de MySQL son:

- Multiplataforma. Funciona en Windows, Linux, Mac OSX, OpenBSD, Solaris, SunOS.
- Su diseño multihilo le permite soportar una gran carga de forma muy eficiente.
- Gran velocidad en la ejecución de operaciones.
- Motores de almacenamiento independientes (MyISAM para lecturas rápidas, InnoDB para transacciones e integridad referencial).
- Ligera, poco consumo de recursos del equipo.

4.7 Java Database Connectivity (JDBC)

Esta es la herramienta usada para comunicar la aplicación Java con la base de datos. Dispone de un seguido de clases y métodos destinados a la realización de operaciones sobre la base de datos. Para llevar a cabo este cometido es necesario disponer de un plugin/driver, el cual coordinará la conexión. Una de las características del JDBC es que usa el lenguaje SQL como lenguaje de comunicación lo que provoca que no importe bajo que tecnología física reside la base de datos.

Se ha optado por implementar esta tecnología bajo un modelo de dos capas. Esto significa que la aplicación Java y el plugin deberán estar en la máquina del cliente, mientras que el servidor de base de datos podrá estar en cualquier otra localización. La siguiente imagen (4.6) esquematiza esta estructura.

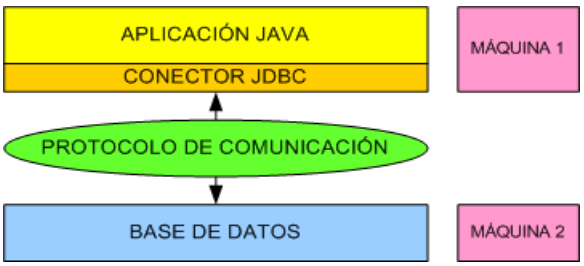


Fig 4.6 Modelo JDBC de dos capas

La información de conexión a base de datos, para este proyecto, está contenida el fichero jdbc.properties. La información que requiere el conector es:

- Clase del plugin/driver
- URL de la bases de datos
- Usuario y contraseña de la BBDD

4.8 Web Application Testing (WAPT)

El WAPT es un sencillo programa que permite simular cargas en una aplicación web o página web. Su funcionamiento consiste en generar un usuario modelo a partir de las URL recogidas durante el acceso de un hipotético usuario. Cuando el circuito del usuario modelo se ha definido es momento de decidir cuantos usuarios accederán al mismo tiempo, durante cuanto tiempo, etc. Una vez lanzada la carga se analizan los resultados obtenidos mediante informes descriptivos y gráficos que proporciona el mismo programa.

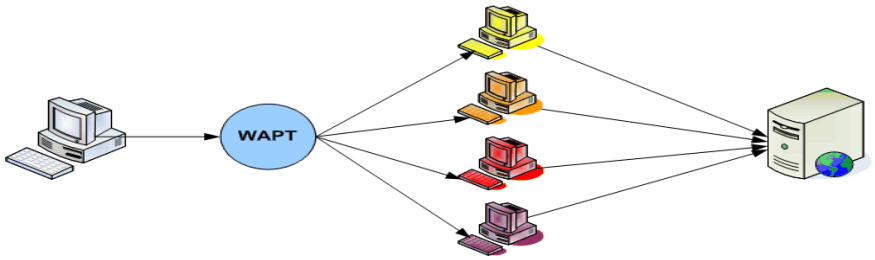


Fig 4.7 Simulación de múltiples conexiones con WAPT

4.9 TMnetSim

Para comprobar el rendimiento de una aplicación web en una WAN (World Area Network) es necesario o bien probarlo en diferentes escenarios reales, o bien simular con la ayuda de un programa intermedio, entre cliente y servidor, las condiciones de la red. Esta segunda opción se ha usado en este proyecto por tres motivos principales, la facilidad de montar un escenario sin necesidad de buscar una ip privada para el servidor, la posibilidad de decidir las condiciones del escenario simulado y la posibilidad de comunicarlo con el WAPT.

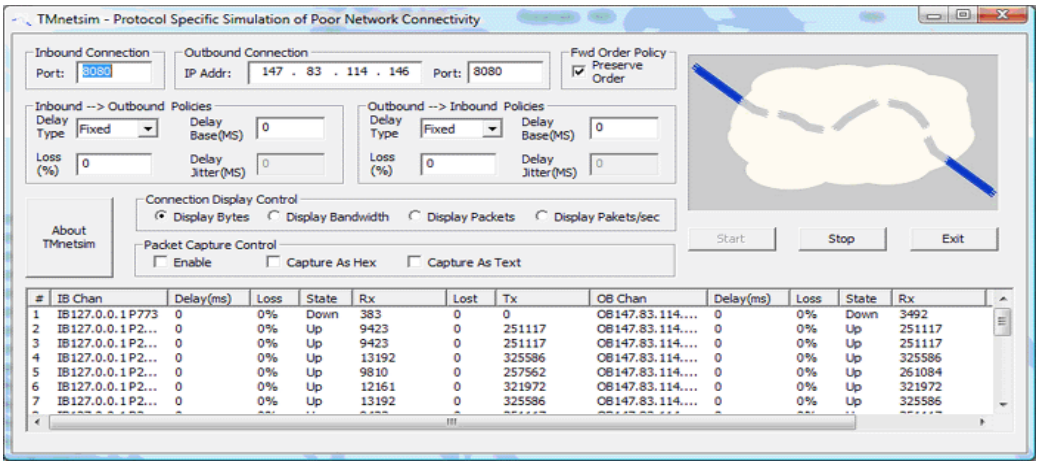


Fig 4.8 Simulación de una WAN con el programa TMnetSim

CAPÍTULO 5. DISEÑO E IMPLEMENTACIÓN

La construcción de una aplicación Web requiere seguir una estricta metodología. Para la comprensión de cómo se ha diseñado e implementado nuestra aplicación, desglosaremos esta sección en cinco apartados. Entraremos en el detalle de cada una de las tres capas del modelo vista controlador, haciendo hincapié previamente en los archivos de configuración básicos y un seguido de buenas prácticas.

5.1 Modelo Vista Controlador (MVC)

Las aplicaciones Web pueden ser implementadas de muchas maneras. No se usaría la misma estrategia para desarrollar una aplicación de dos pantallas que una de cien, ni sería conveniente construir de la misma forma una aplicación con una base de datos pequeña que con una grande. Es por ello que una interesante forma de trabajar es adaptar la aplicación a un patrón que se adapte a nuestras necesidades.

Para aplicaciones de pequeña envergadura y siempre teniendo en cuenta que queremos un comportamiento dinámico, lo más útil sería usar un patrón donde las pantallas reflejen directamente el modelo de base de datos. Sin embargo, si queremos muchas pantallas, tenemos una lógica de negocio abundante y la base de datos no es simple, conviene estructurar la aplicación de otra manera. En caso contrario corremos el riesgo de montar un sistema muy complejo e imposible de mantener. En este sentido, el patrón modelo vista controlador empleado en el proyecto es una buena manera de modularizar la aplicación, separando los datos de la aplicación, de la interfaz del usuario y de la lógica de negocio. Esta forma de trabajar provoca que la configuración sea más extensa pero a nivel conceptual es muy intuitiva y permite diferenciar bien los componentes y actuar sobre ellos de una manera más precisa. Analizamos a continuación un poco más los tres grandes bloques que define este patrón.

- **Modelo.** Es el módulo que representa los datos del programa. Se encarga de manipular estos datos de forma coherente y ofreciéndolos a la aplicación a medida que los va requiriendo. Esta capa no debe tener conocimiento ninguno ni del controlador ni de las vistas y en el capítulo de implementación conoceremos como logramos que las capas interactúen y los datos se vuelvan dinámicos.
- **Vista.** Es la representación visual de los datos. Serán en sí las páginas que el usuario contemple y es la capa mediante este interactuará con la aplicación. Aunque a menudo los datos que se representen son el reflejo de la base de datos se hará uso de unos objetos llamados bean para establecer la relación pasando siempre por la capa controlador. Entraremos más en detalle más adelante.

- **Controlador.** Es el tercero de los módulos. el que proporciona significado a las órdenes del usuario y el que en definitiva crea la relación entre modelo y vista. Esta capa será la encargada de decidir cuando se debe modificar la base de datos y cuando se deben obtener datos de ella para que los disponga la vista. Además se encarga de la lógica de redireccionamiento de pantallas.

En la siguiente imagen (5.1) podemos observar el ciclo que realiza una aplicación basada en el modelo vista controlador.

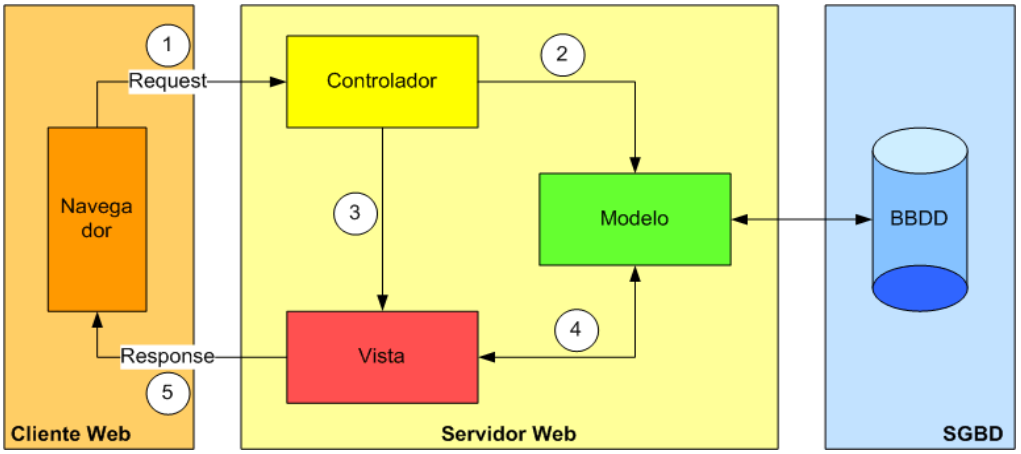


Fig 5.1 Arquitectura Modelo Vista Controlador (MVC)

En primer lugar el usuario debe realizar algún tipo de petición (*request*), es decir alguna acción en el navegador. El controlador será el responsable de capturar esa petición y de realizar diferentes operaciones. Estas operaciones muy posiblemente necesiten obtener y modificar información de la base de datos por lo que se hará uso de la capa modelo. Una vez el controlador lleve a cabo sus acciones deberá escoger la pantalla a la que se debe redirigir la petición. Será necesaria la capa vista para ofrecer al usuario los datos de una forma estructurada. El resultado será devuelto (*response*) al cliente para que el usuario pueda de nuevo iniciar el ciclo.

Las principales ventajas que proporciona trabajar con un patrón como este son la separación de los datos de la base de datos de su representación gráfica y la simpleza a la hora de añadir y modificar nuevas vistas para que se adapten según a las necesidades del modelo. Otras características que ofrece este patrón son la independencia entre las tres capas, la escalabilidad que proporciona al poder actuar en cualquiera de las tres capas y la separación de funciones que facilita el mantenimiento. En contra de este patrón se debe destacar que a veces no es simple separar las capas y definir que tecnología se usa para cada propósito. Además trabajar con este método provoca que exista un número más elevado de archivos.

5.2 Struts Framework

Struts es un Framework libre de The Apache Sofware Fundation que ofrece distintas herramientas para facilitar aplicaciones web basadas en el modelo vista controlador. Se ha optado por usar la versión Struts 1.3.10.

Las principales ventajas que ofrece son:

- Reduce el desarrollo.
- Provee una configuración centralizada mediante archivos xml.
- Proporciona diferentes recursos para facilitar la construcción de una aplicación web (formularios, validaciones…) gracias al uso de bibliotecas(taglibs).
- Es un mecanismo para capturar las peticiones (request) del tipo URI.
- Es un mecanismo para enviar las respuestas (response).
- Permite interactuar con otros frameworks y herramientas (p.ej. itext)

En realidad, Struts está más enfocado a tratar la capa vista y la capa controlador, dejando espacio a otras tecnologías para tratar la capa modelo.

En los distintos apartados de este capítulo se analizará en detalle los diferentes aspectos de Struts, se conocerán las distintas tecnologías complementarias y se verá como interactúa todo en conjunto.

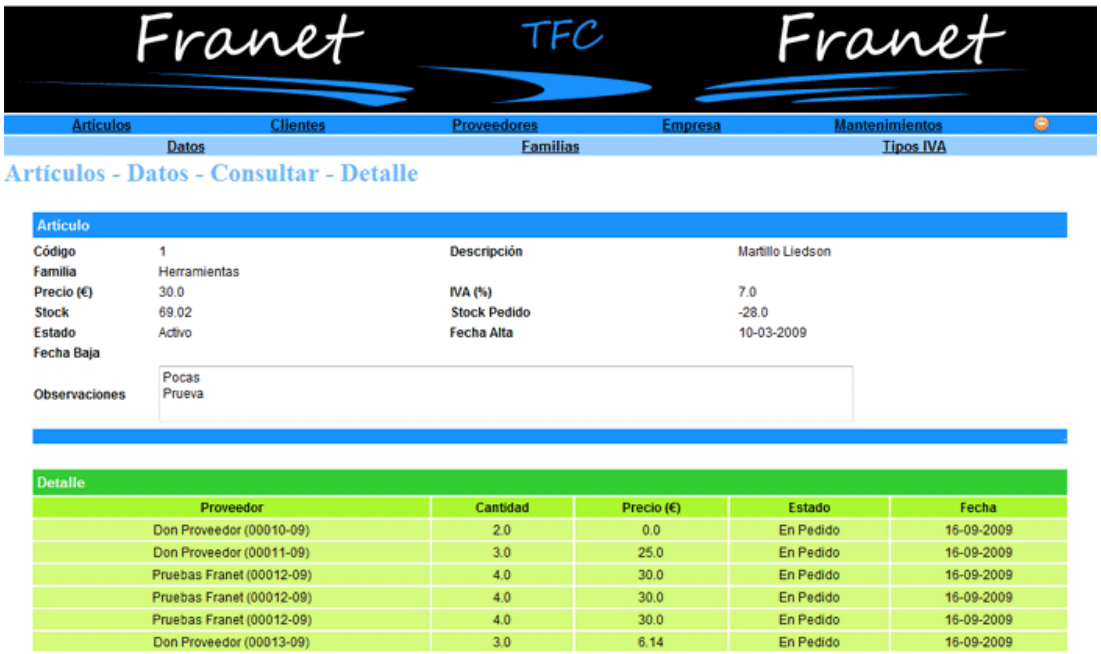


Fig 5.2 Pantalla de la aplicación Franet, implementada con Struts.

5.3 Configuración

Los ficheros de configuración son la base de la construcción. En este punto veremos que archivos requiere Franet para implementar el modelo vista controlador.

El primero se trata del web.xml, que podemos consultar en el anexo, y que es el pilar de cualquier aplicación Web. En él se definen todos los Servlets de los que se hará uso. En nuestro caso han sido incluidos el ActionServlet, controlador de Struts, como el ContextLoaderServlet, para Spring. En el mismo archivo también podemos hallar datos de inicialización de logs, archivos de configuración auxiliares y las definiciones de las bibliotecas de etiquetas (taglibs) de Struts, que conoceremos más a fondo el apartado de la capa de Vista.

El segundo de los archivos que incluye el framework de Struts es el struts-config.xml. Este archivo se referencia en el web.xml como parámetro del ActionServlet y nos permite especificar diferentes componentes que detallaremos en la capa controlador: Action y ActionForm. Las definiciones de plugins para la inicialización de herramientas complementarias, como Tiles y Validator, también las podemos encontrar en este archivo. En nuestro caso particular hemos optado por dividir el struts-config.xml original en diferentes archivos (struts-config-clientes.xml, struts-config-proveedores.xml, etc.) para facilitar su consulta y modificación.

5.4 Buenas prácticas

Como ya hemos empezado a ver, el framework de Struts nos define unas bases a la hora de implementar aplicaciones. Sin embargo, debido a las dimensiones que pueden alcanzar estas es conveniente seguir una metotodología más estricta para facilitar la construcción y mantenimiento. Algunas de las que se han tenido en consideración en este proyecto son específicas para este tipo de aplicaciones, otras son válidas para la programación en general y especialmente para el lenguaje Java. A continuación se desglosan las principales:

- **Estructuración de carpetas.** Una aplicación Web requiere que ciertos recursos estén en puntos concretos del servidor. Es por eso que a la hora de montar el .war necesitamos que el código, las imágenes, librerías… estén clasificados tal y como se define en el build.xml. Eclipse también requiere tener el código bien localizado y asilado de forma que se pueda desarrollar sobre él e incluso realizar debug. Así pues, definir una correcta estructura de carpetas es necesario antes de empezar a desarrollar la aplicación.
- **Archivos de propiedades**. Una de las problemáticas del código es que cualquier cambio en él provoca que tenga que tenga que ser recompilado. Los archivos de propiedades sirven para aislar el código de los

parámetros de configuración, como podrían ser los de conexión a una base de datos. Si en un momento dado se decide cambiar la base de datos sólo habría que cambiar los datos en el archivo sin necesidad de renovar la aplicación entera.

- **MessageResources.properties**. Este archivo de propiedades, proporcionado por el framework de Struts, facilita la centralización de los literales y expresiones de las pantallas del programa. Cada literal tendrá relacionado un identificador de forma de que estos segundos puedan ser fácilmente reaprovechados. Para hacer uso de este recurso se debe definir en el fichero struts-config.xml.
- **Ficheros CSS y JavaScript centralizados**. Las funcionalidades asociadas a estos archivos serán detalladas en futuros apartados. Se optó por crearlas, en la mayoría de casos, de forma centralizada para reducir al máximo la repetición de código.
- **Otras consideraciones**. Seguir una nomenclatura de clases y métodos uniforme, la creación de clases reutilizables y el uso de una clase para almacenar constantes son algunas otras prácticas que se han tenido en consideración.

5.5 Capa Modelo

Las aplicaciones web son una herramienta de gestión de información. En este apartado analizamos la capa modelo que será la responsable de albergar la base de datos.

5.5.1 Base de datos

Antes de diseñar cualquier base de datos es imprescindible conocer el contexto en que se engloba y definir muy bien que necesidades queremos cubrir con ella. En este proyecto la base de datos debía dar soporte a una aplicación de facturación y sus principales pilares debían ser clientes, artículos, proveedores y empresa.

Para llevar a cabo el diseño de la base de datos se tomó como referencia un modelo entidad-relación. Entre las ventajas de este modelo está el evitar al máximo la redundancia de datos y dificultar la introducción de datos inconsistentes. El primer paso para llevar a cabo este modelo es identificar las entidades, es decir, todos aquellos objetos que simulan personas, lugares, características... Franet contiene un total de 25 entidades: clientes, proveedores, empresa, etc. El segundo paso es identificar las relaciones entre entidades. La entidad población debe tener una entidad país relacionada. No solo eso, también es necesario conocer si es una relación uno a uno, uno a varios, varios a uno o varios a varios.

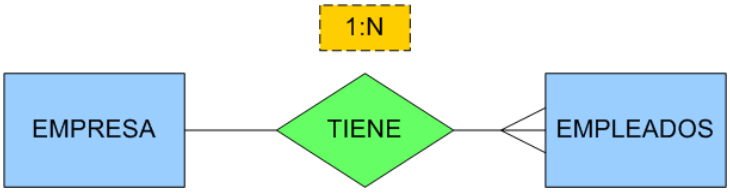


Fig 5.3 Relación entre entidades

Una vez decididas las entidades y conocidas sus relaciones fue necesario detallar cada entidad. Para ello se especificaron todos sus atributos. Por cada uno de ellos se debe definir:

- Nombre descriptivo
- Dominio (Tipo de dato y longitud)
- Valores por defecto del atributo (opcional).
- Si el atributo debe estar siempre informado (admite o no nulos).
- Otras características (atributos compuestos, derviados, multievaluados…)

Hay que mostrar especial atención a los atributos que se utilizan para relacionar otras entidades.

Tabla 5.1 Ejemplo. Entidad Artículos.

Entidad Artículos		
Atributo	Dominio	Clave
IDARTICULO	Int (5)	Primaria
ARTICULO	Varchar (50)	
IDFAMILIA	Int (5)	Secundaria
IDIVA	Int (2)	Secundaria
OBSERVACIONES	Varchar (1000)	
STOCK	Decimal (5,2)	
STOCK_PEDIDO	Decimal (5,2)	
PRECIO_VENTA	Decimal (10,2)	
IDESTADO	Int (2)	
F_ALTA	Timestamp	
F_BAJA	Timestamp	

El último paso del diseño del modelo consiste en dibujar el diagrama de entidad relación. En la figura 5.4 podemos encontrar un ejemplo. Para facilitar la comprensión del diagrama solo se han incluido los atributos claves (establecen una relación) y se han omitido entidades secundarias (poblaciones, provincias, tipos de envío…).

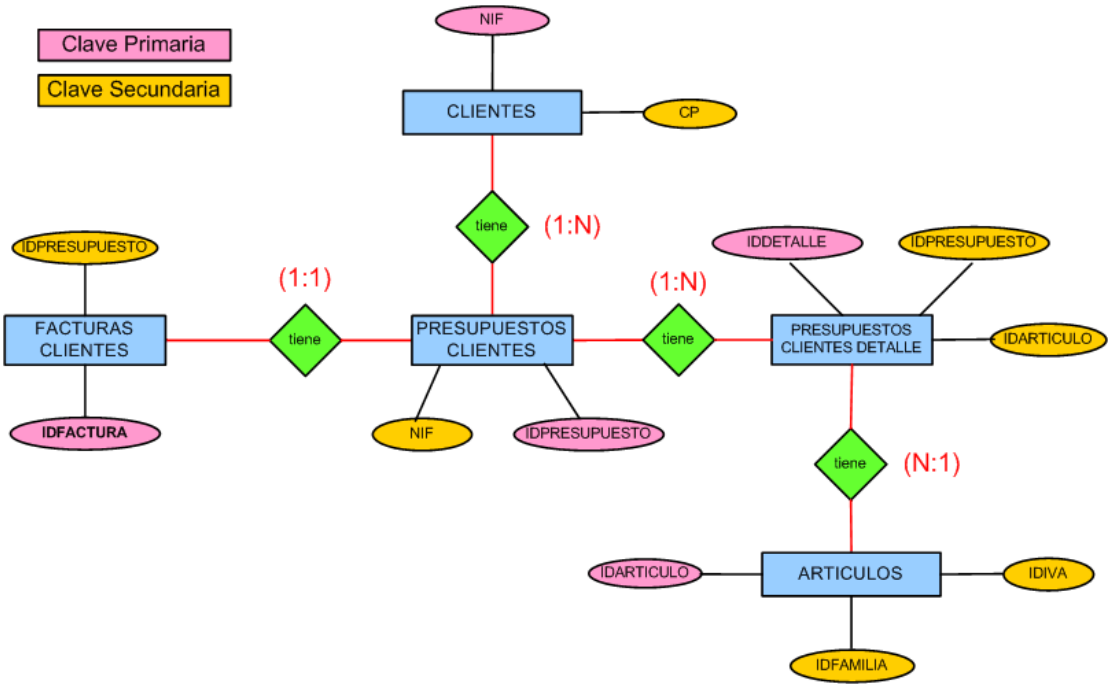
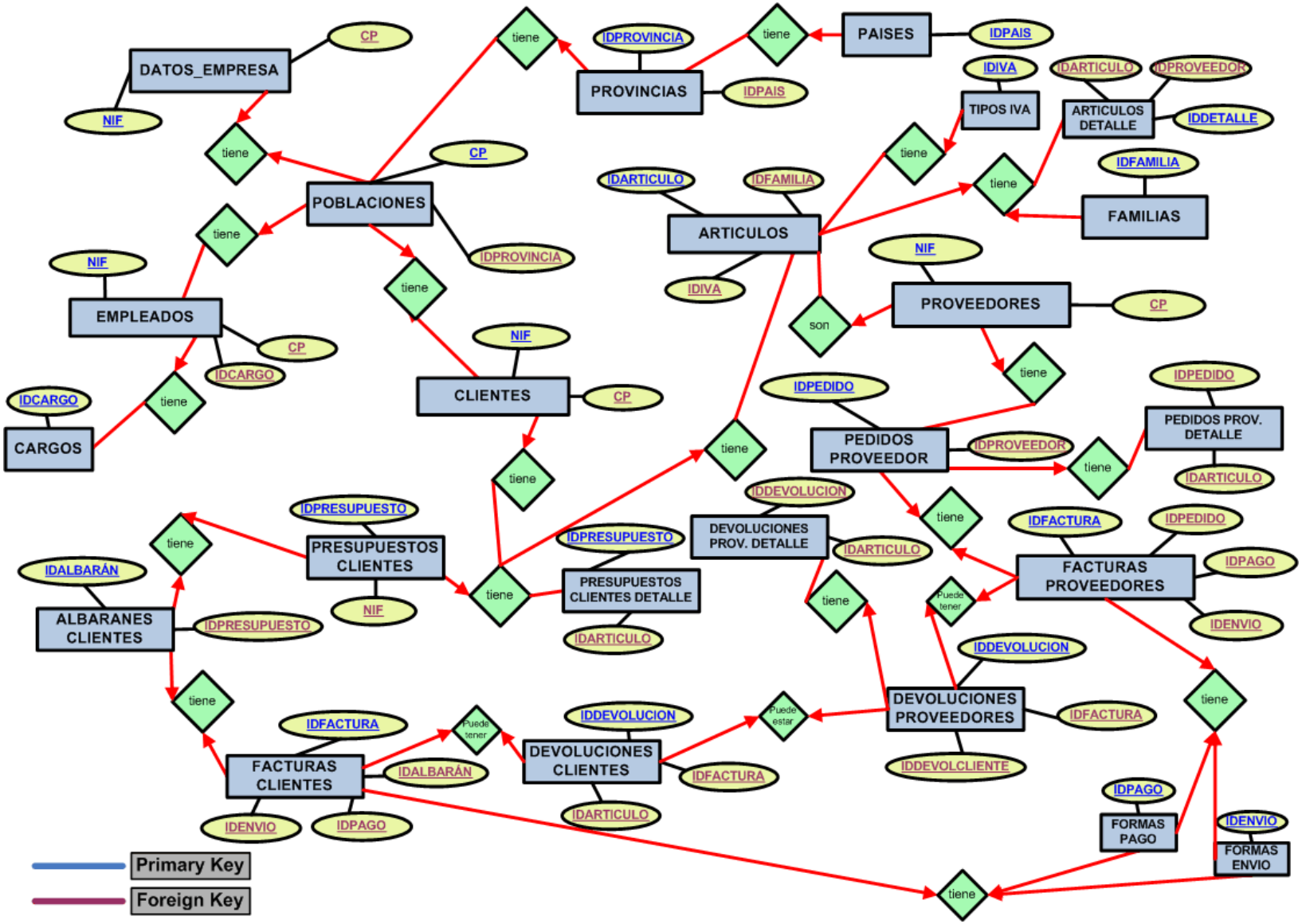


Fig 5.4 Modelo E/R simplificado

En la imagen 5.4 podemos ver las ramas que dependen de clientes. La primera entidad es propiamente Clientes y dispone de dos campos clave: NIF y CP. El primero será usado como clave primaria, mientras que el segundo será una clave foránea para relacionar el cliente con una población. La entidad Presupuestos Clientes, por su parte, tiene dos claves. El NIF será usado para relacionar un cliente con un presupuesto. Esta relación podemos observar que es uno a varios (1:N), por lo que un cliente puede tener varios presupuestos. Otras relaciones distintas que podemos encontrar son por ejemplo la que une Facturas Clientes con Presupuestos Clientes, donde un presupuesto sólo podrá tener asociada una factura y Presupuestos Clientes Detalle con Artículos donde un detalle sólo podrá tener un artículo pero un artículo puede pertenecer a más de un detalle. Esta última es una relación uno a varios pero vista de forma inversa.

Una vez finalizado el diseño es necesario trasladarlo a la base de datos MySQL. En este proyecto se ha optado por que cada entidad se convierta en una tabla, con sus atributos definidos y sus relaciones estipuladas. Todas las tablas, con la totalidad de las propiedades de sus atributos, se adjuntan en el anexo. Un aspecto importante a la hora de creación de tablas es el hecho de haber usado la tecnología InnoDB. Esta provoca que cada vez que se guarda un registro en la base de datos se realizan una serie de comprobaciones automáticas para asegurar que las relaciones entre tablas son coherentes.

En la siguiente página podremos consultar el diseño lógico de la base de datos.


DATOS_EMPRESA
CP
NIF
tiene
POBLACIONES
CP
IDPROVINCIA
PROVINCIAS
IDPROVINCIA
IDPAIS
PAISES
IDPAIS
EMPLEADOS
NIF
CP
IDCARGO
CARGOS
IDCARGO
CLIENTES
NIF
CP
ARTICULOS
IDARTICULO
IDFAMILIA
IDIVA
TIPOS IVA
IDIVA
ARTICULOS DETALLE
IDARTICULO
IDPROVEEDOR
IDDETALLE
FAMILIAS
IDFAMILIA
PROVEEDORES
NIF
CP
son
PEDIDOS PROVEEDOR
IDPEDIDO
IDPROVEEDOR
PEDIDOS PROV. DETALLE
IDPEDIDO
IDARTICULO
DEVOLUCIONES PROV. DETALLE
IDDEVOLUCION
IDARTICULO
FACTURAS PROVEEDORES
IDFACTURA
IDPEDIDO
IDPAGO
IDENVIO
Puede tener
DEVOLUCIONES PROVEEDORES
IDDEVOLUCION
IDFACTURA
IDDEVOLCLIENTE
Puede estar
PRESUPUESTOS CLIENTES
IDPRESUPUESTO
NIF
PRESUPUESTOS CLIENTES DETALLE
IDPRESUPUESTO
IDARTICULO
ALBARANES CLIENTES
IDALBARÁN
IDPRESUPUESTO
FACTURAS CLIENTES
IDFACTURA
IDALBARÁN
IDENVIO
IDPAGO
Puede tener
DEVOLUCIONES CLIENTES
IDDEVOLUCION
IDFACTURA
IDARTICULO
FORMAS PAGO
IDPAGO
FORMAS ENVIO
IDENVIO
tiene
Primary Key
Foreign Key

5.5.2 Patrón DAO

El modelo vista controlador en la teoría nos permite diferenciar muy bien tres capas. Sin embargo, en la realidad del código a veces es difícil saber definir donde acaba una capa y donde empieza otra. Un claro ejemplo de esto son los accesos a base de datos. La capa controlador es la que se encarga de procesar la información, sin embargo, hay muchas probabilidades que esa información deba ser recuperada o almacenada en base de datos. Si somos coherentes con la arquitectura, las capas controlador y modelo aunque deben estar coordinadas no tendrían que compartir funcionalidad. Es necesaria, pues, una estructura para aislar el código de cada capa. La arquitectura creada se basa en el patrón DAO y la podemos en la siguiente imagen (figura 5.5).

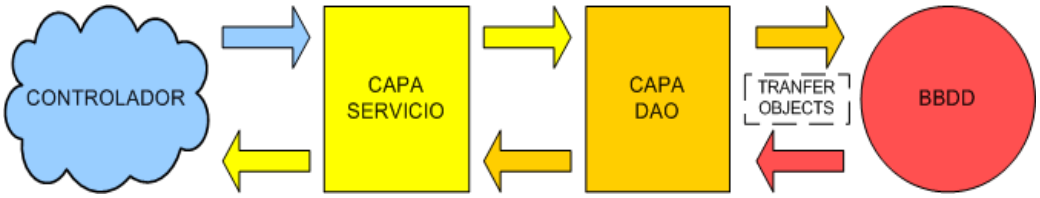


Fig 5.5 Arquitectura patrón DAO

La capa DAO de la figura contiene un seguido de clases DAO que serán las encargadas de realizar los accesos a base de datos. Una forma útil de trabajar es asociar una clase DAO a cada entidad de la base de datos y en ella se definirán las diferentes formas de manipular los datos. Si ponemos de ejemplo el DAO de empleado, EmpleadosDAOImpl.java (ver anexo), podremos encontrar métodos para insertar, actualizar, recuperar...

El patrón DAO define un concepto entre la capa DAO y la base de datos. Se trata de los Data Transfer Objects o Value Objects que simulan la conversión de un objeto Java a una tabla de base de datos. En nuestro proyecto será Hibernate el encargado de llevar a cabo este cometido tal y como veremos en el siguiente apartado.

La capa de Servicio es dónde se sitúa la lógica de negocio de aquellas funcionalidades que hacen uso de la base de datos y que en cualquier momento el controlador puede requerir. Esta forma de trabajar permite que el controlador no tenga conocimiento alguno de la arquitectura de la base de datos ya que desde su punto de vista realiza una petición y obtiene una respuesta. Esta capa también se divide en clases y será ella misma la encargada de coordinar diferentes Servicios y DAO si para generar la respuesta tiene que consultar información de diferentes entidades.

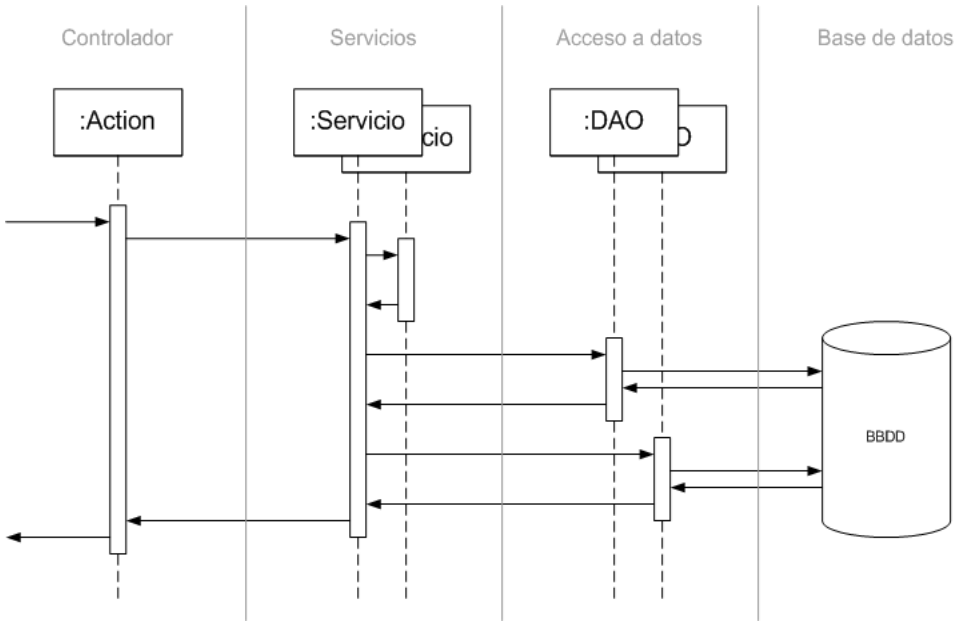


Fig 5.6 Acceso a bbdd mediante Servicios y DAO

5.5.3 Hibernate

Una de las problemáticas que surge a la hora de trabajar con la base de datos es que esta sigue un modelo relacional y en cambio el código Java está orientado objetos. Hibernate es una tecnología que nos permitirá convertir objetos en registros o viceversa sin necesidad de implementar complejas estructuras.

Para realizar este proceso de mapeo es necesario configurar unos archivos que establezcan la relación donde cada fichero simulará una tabla de la base de datos. La idea final es que la estructura formada por todos estos archivos logren proyectar una imagen de la base de datos, no solo con sus tablas y atributos, sino con las relaciones incluso. Así pues, los componentes que entran en juego en esta conversión son: una tabla en la base de datos, una clase Java que simule esa tabla y un fichero XML donde se informe la correspondencia entre atributos de la tabla y propiedades de la clase.

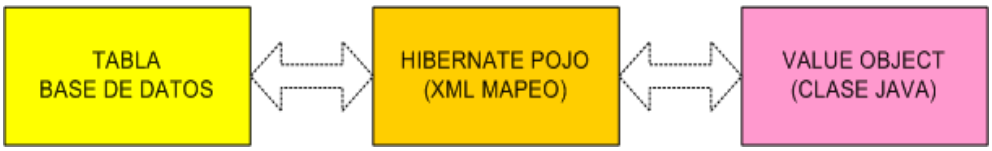


Fig 5.7 Mapeo Hibernate

La clase Java estará formada por un seguido de propiedades, constructores y los métodos para almacenar y recuperar las propiedades.

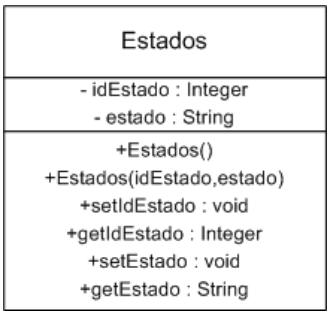


Fig 5.8 Diagrama de Clase Estados

El archivo de mapeo, conocido como POJO, es de formato XML. Haciendo uso de etiquetas tendremos que definir cada entidad. Para empezar, la etiqueta *class* nos especificará que entidad de la base de datos relaciona a que objeto Java. En el anexo podemos encontrar un fichero de mapeo con nombre Articulos.hbm.xml.

Posteriormente hay que etiquetar todos los atributos de la entidad empezando por el el identificador de la tabla. Definir un identificador es imprescindible al ser el valor referente para realizar la conversión y debe coincidir con la clave primaria de la tabla. Una vez definido el identificador se mapean el resto de atributos, que se diferencian en dos grupos, los que relacionan otras tablas y los que no. Los atributos simples se crean bajo la etiqueta property y en ella se parametrizan todas las características de ese atributo: tipo, columna, entidad… Por otra parte encontramos los atributos que sirven para relacionar otras tablas, que en este caso servirán para relacionar otros archivos POJO. Hibernate permite crear relaciones uno a varios, varios a uno, etc. Las etiquetas de estos atributos son many-to-one y set. En ellas se podrán especificar parámetros para dar forma a la relación. Uno parámetro importante es lazy que nos permitirá importar los datos de tablas relacionadas convertidas ya en objetos de forma transparente.

5.5.4 Spring

Spring es una tecnología compuesta por una serie de módulos orientados a facilitar el desarrollo de aplicaciones Web y es válido como framework para implementar un modelo vista controlador. Sin embargo, en este proyecto solo se ha hecho uso de algunas funcionalidades que complementen la arquitectura de Struts:

- Asumir el control del JDBC y simplificar su configuración.

- Facilitar instrumentos para implantar el patrón DAO.
- Coordinar con Hibernate para complementar el patrón DAO.
- Realizar las conexiones y gestión de sesiones a base de datos de forma transparente.
- Permitir el uso de transacciones en base de datos. Es posible implementar funcionalidades complejas sin riesgo a dejar datos inconsistentes si alguna operación no se realiza con éxito.

Entrando un poco en el detalle de su implementación es necesario recordar que hace uso de un Servlet (org.springframework.web.context.ContextLoaderServlet) distinto al de Struts y que se tendrá que especificar en el archivo web.xml.

El primer concepto que introduce Spring es la BeanFactory. En ella se deben definir una serie de beans/objetos, cada cual proporciona una funcionalidad concreta para la aplicación. El fichero applicationContext-Hibernate.xml es el responsable de albergar los beans.

La BeanFactory usada en Franet puede desgranarse en tres grupos:

- Inicialización y manipulación del JDBC.
- Inicialización y mando de Hibernate.
- Desarrollo del patrón DAO. Un bean por clase e interfaz.

Entrando un poco más en el detalle del último punto es bueno saber que la capa DAO se ha montado haciendo uso de una interfaz. Cada interfaz tiene una clase asociada donde se encontrará propiamente el código. Volviendo al caso de empleados, la clase EmpleadosDAO.java es la interfaz y la clase EmpleadosDAOImpl.java es donde encontraremos definidos los métodos de acceso a base de datos.

Por último mencionar la existencia de una clase ServiceFinder.java para recuperar beans desde cualquier punto de la aplicación y poder hacer así uso de una funcionalidad concreta. Algunos beans como JDBC o Hibernate se usan automáticamente pero para ello hay que identificarlos con un nombre estándar.

5.6 Capa Vista

Para llevar a término la capa visual se han usado diversas herramientas y se ha tomado como base las Java Server Pages (JSP).

5.6.1 Java Server Pages (JSP)

Esta funcionalidad basada en Java es la forma en que la aplicación muestra la información en el explorador. Por regla general una JSP está construida por código HTML y código Java. El código es tratado por el Servlet y se genera una respuesta en consecuencia. En nuestro caso además de los dos códigos anteriores disponemos de etiquetas que proporciona Struts y que potencian la funcionalidad.

El lenguaje HTML es el encargado de tratar los temas más visuales de la página y su simpleza facilita mucho el trabajo. Sin embargo, a la hora de introducir lógica de negocio resulta ser un mal aliado.

Las etiquetas de Struts se agrupan en bibliotecas (taglibs) y abren las puertas a un seguido de instrumentos que simplifican por ejemplo la creación y tratamiento de formularios. Estas bibliotecas se especifican en ficheros de extensión tld y para facilitar su uso deben estar referenciadas en el web.xml. Reservamos el siguiente apartado para analizarlas más a fondo.

Cuando las etiquetas de Struts resultan insuficientes el código Java cobra protagonismo. Es una solución menos elegante porque complica la lectura de la JSP pero resulta muy efectiva. Un ejemplo de ello podría ser un listado en que el número de filas puede ser variable (5.1). Tal y como podemos ver, situaríamos el código dentro de un típico while. El código java es rápidamente reconocible porque se limita entre etiquetas <% %> para que el servidor lo pueda detectar.

```
<% while (numFilas) { %>
      // Código asociado una fila                    (5.1)
<% } %>
```

5.6.2 Struts Taglibs

Las etiquetas de Struts tienen una estructura muy parecida a las de HTML. El framework nos ofrece cuatro bibliotecas: bean, html, logic y nested. Para conocer que nos proporcionan y como se implementan analizaremos un seguido de ejemplos, extraídos de la JSP del creación de un empleado. El código de esta página (nuevoEmpleadosBody.jsp) es consultable en el anexo.

Como primer paso y con tal de alertar al Servlet del uso de etiquetas externas, es decir, aquellas que no forman parte estrictamente de HTML es necesario incluir las bibliotecas antes de usar sus etiquetas.

```
<%@ taglib uri="/tags/struts-html" prefix="html" %>    (5.2)
```

En este caso se aceptan etiquetas de la biblioteca html. Ahora podemos empezar a usar sus etiquetas en la posición de la JSP que nos interese.

```
<html:form action="/nuevoEmpleado">
```
(5.3)

Esta sería una típica etiqueta dónde *html* hace referencia a la biblioteca, *form* es el nombre de la etiqueta, y *action* es uno de los parámetros que acepta dicha etiqueta. En este ejemplo la etiqueta nos informa que vamos a crear un formulario y el parámetro action nos define la clase que deberá llamar el Servlet al seleccionar el botón de envío de datos. Las etiquetas de la biblioteca html fueron diseñadas para facilitar la construcción de formularios. Así pues, los campos y botones del formulario serán creados con etiquetas de la misma biblioteca.

Una de las principales cualidades de estas etiquetas es que podemos mostrar datos de forma dinámica. Esto nos permitirá visualizar datos procedentes de la base de datos sin necesidad de tenerlos codificados en la propia JSP. El campo que despliega una lista de cargos (5.4) es un ejemplo de ello. La etiqueta *options* simula la lista y el parámetro *collection* referencia el objeto Java que contiene los valores de dicha lista. Para que la JSP disponga de dicho objeto en el momento de ser cargada es necesario recuperar los valores de la base de datos, almacenarlos en un objeto y añadir éste a nivel de *request*.

```
<html:options collection="cargosList" property="idCargo"
labelProperty="cargo"/>
```
(5.4)

La biblioteca bean fue ideada para el uso de contenedores (p.e. ActionFrom) y la recuperación de sus propiedades. También es útil para obtener las expresiones del fichero MessageResources.properties, el cual nos servía para centralizar los mensajes de la aplicación. El texto que aparece en el botón de guardar los datos del empleado será recuperado a partir de la etiqueta:

```
<bean:message key="boton.guardar"/>
```
(5.5)

Las dos otras bibliotecas que ofrece el framework son logic y nested. La primera es útil para conocer en que condiciones se encuentra una propiedad (existe, no existe, es igual a, es mayor a, etc.) y definir un resultado visual en consecuencia. Si el valor de una propiedad es blanco es posible que no queramos visualizar su campo asociado. Por su parte la biblioteca nested nos ofrece funciones de encadenar etiquetas de los otros grupos. Por ejemplo, nos puede hacer falta

realizar un bucle de etiquetas para mostrar las propiedades de una forma coherente (un empleado podría tener más de un cargo).

5.6.3 Tiles

Un punto importante en una aplicación Web es siempre su diseño. Puesto que estamos tratando de un instrumento dirigido a un grupo de usuarios es realmente importante la distribución de datos e imágenes en la pantalla del navegador. Para lograr dicho objetivo se ha hecho uso del framework Tiles, compatible con la filosofía Struts, y que es una herramienta que nos permite definir plantillas que se adapten a nuestras necesidades para la distribución de contenido. Cada plantilla tendrá diversas secciones y cada sección tendrá asignada una jsp.

La construcción de plantillas es relativamente simple. Tan solo es necesario crear una jsp en la que se incluyen etiquetas de Tiles para formar las secciones y en combinación con etiquetas HTML lograremos obtener la forma deseada. Una vez obtengamos el molde podremos reaprovecharlo tantas veces como queramos. En nuestro caso particular con tres hemos logrado construir todas las pantallas de la aplicación.

Para construir una nueva pantalla simplemente hará falta realizar una nueva jsp donde informar, otra vez con etiquetas Tiles, que plantilla usamos y que jsps queremos situar en cada sección. En la imagen 5.9 podemos ver a modo de ejemplo una pantalla con una cabecera, dos menús, un título, el contenido y un pie de página.

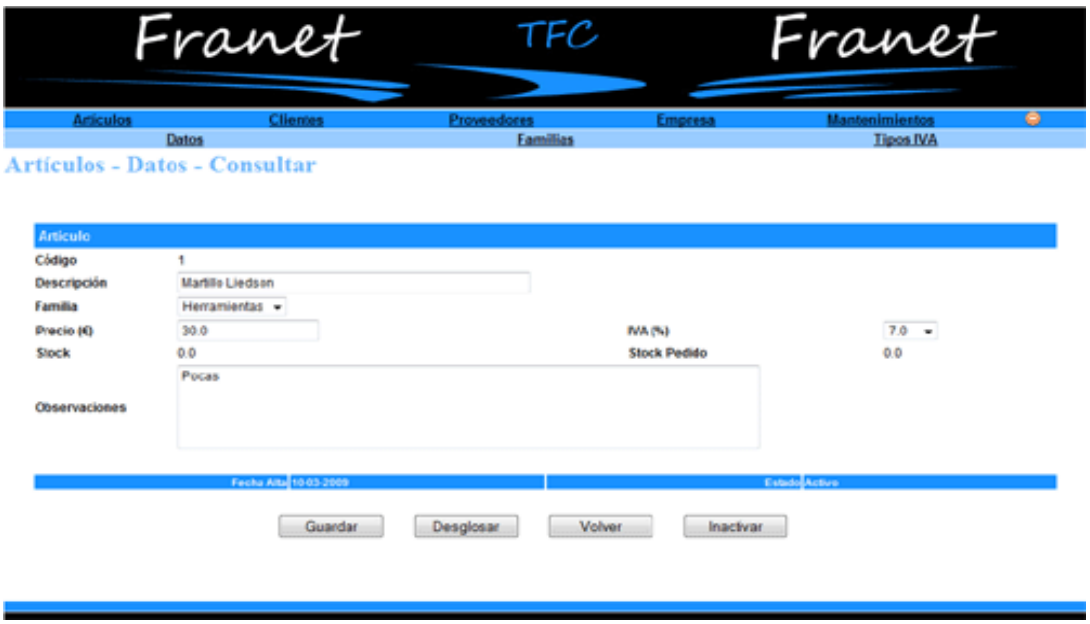


Fig 5.9 Pantalla Franet con múltiples secciones (Tiles)

5.6.4 Cascading Style Sheets (CSS)

Otro aspecto que se ha adoptado a nivel gráfico es el uso de una hoja de estilo externa (extracto del archivo style.css consultable en el anexo). Estas hojas de estilo, CSS, permiten separar la estructura de una página web de su presentación.

Las ventajas de esta forma de trabajar son varias. Para empezar y sabiendo que disponemos de múltiples listados y formularios que siguen el mismo patrón no tiene sentido repetir código y el día que se quiera cambiar tener que modificar incontables JSP. Haciendo referencias a la hoja de estilos permitimos que un cambio de color de azul a rojo solo requiera una pequeña modificación en la hoja de estilo. Además de esta manera las jsp quedan más libres de código HTML y su manipulación resulta más sencilla.

Por último cabe destacar que el W3C (World Wide Web Consortium) define un protocolo de uso de etiquetas HTML y hojas de estilo que algunos exploradores, como Mozilla Firefox siguen como estándar. En consecuencia, haciendo uso de las hojas de estilo facilitamos que sea compatible con más de un explorador Web y para el caso de Franet podemos comprobar que cualquiera de las últimas versiones de los exploradores más extenidos es válido para trabajar:

- Mozilla Firefox 3.5
- Internet Explorer 9
- Opera 10
- Google Chrome
- Safari 4

5.6.5 JavaScript

El Javascript es un lenguaje de programación muy simple pensado para añadir pequeñas funcionalidades a nivel de cliente (explorador). En Franet se ha estimado su uso por dos motivos principales que analizamos a continuación.

El primero es por el incremento de funcionalidad que le dan a las etiquetas de Struts. En el caso de la biblioteca html varias de sus etiquetas contienen métodos onclick, onchange, onpresskey… que en combinación con un método Javascript pueden definirse muchos comportamientos. Es un ejemplo de ello el enviar los datos de un formulario mediante la selección del botón return.

El segundo motivo es por eficiencia y para entenderlo vamos a exponer un caso práctico. Una de las funciones que puede realizar un usuario en Franet es inactivar un empleado, una familia de artículos, un cargo… Esto se realiza pulsando un botón y por motivos de seguridad se decidió que el usuario

corroborara que realmente quería llevar a cabo esa acción mediante la aparición de un mensaje. Para implantarlo se decidió llamar al método *confirm* de Javascript y así evitar que esa validación llegara al servidor y fuera necesario crear una nueva pantalla.

5.6.6 Struts Validator

La gran mayoría de formularios requieren que se lleven a cabo una serie de validaciones. Struts Validator es un módulo que complementa el framework de Struts y que mediante una combinación de ficheros de configuración y etiquetas permite validaciones en los formularios definidos en la JSP creando internamente una estructura Javascript. Para el funcionamiento de esta herramienta será necesario un plugin. Este se inicializa en struts-config.xml y se encargará de cargar los ficheros de configuración. Ambos ficheros se anexan.

El primero de los ficheros es validator-rules.xml y en él se deben definir las reglas para definir validaciones. Entre ellas podemos encontrar:

Tabla 5.2 Reglas de validación de Validator.

Nombre	Descripción de la validación
byte,short,integer, long,float,double	Valor de tipo esperado
creditCard	Valor con formato válido de tarjeta de crédito
date	Valor en formato fecha
email	Valor que siga la estructura de email
mask	Valor que cumpla unas condiciones de formato
maxLength	Valor menor a
minLength	Valor mayor a
range	Valor definido entre A y B
required	Valor informado

El otro fichero será el encargado de contemplar que formularios requieren validaciones y sobre que campos hay que aplicarlas. Veamos un ejemplo (5.6).

```
<form name="guardarProvincias">
        <field property="provincia" depends="required,mask">
                <arg key="provincias.modificar.provincia"/>
                <var>
                        <var-name>mask</var-name>                    (5.6)
                        <var-value>^[a-zA-Z]*$</var-value>
                </var>
        </field>
</form>
```

La etiqueta form nos informa del formulario. La etiqueta field nos especifica el campo y las condiciones de validación. En el ejemplo podemos ver que se trata del campo provincia y que las validaciones que hay que llevar a término son de campo requerido y de máscara. Como las validaciones definidas a nivel de validator-rules.xml son muy genéricas será necesario especificar algunos parámetros extra haciendo uso de las etiquetas var. Es el caso de aplicar una máscara.

Las validaciones hechas con Validator son a nivel de cliente por lo que el servidor no llegará a enterarse. Las JSPs reconocen que deben aplicar validación gracias al uso de la etiqueta html:javascript (5.7).

```
<html:javascript formName="guardarProvincias"
                 staticJavascript="true"/>
```
(5.7)

El mensaje de validación incumplida aparecerá haciendo uso de un método Javascript.

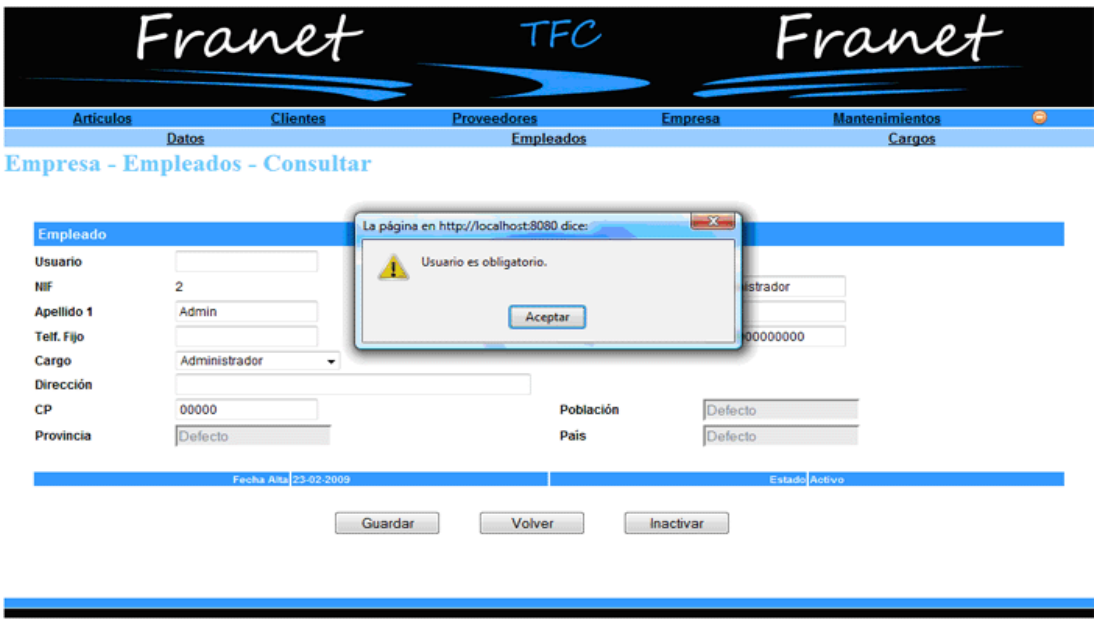


Fig 5.10 Error detectado por Validator

5.7 Capa Controlador

Presentadas las dos primeras capas es hora de hacerlas interactuar gracias a la capa Controlador.

5.7.1 ActionForm

Los formularios, como ya se ha ido introduciendo, son uno de los instrumentos más usados. Struts proporciona una clase ActionForm que nos facilita el tratamiento de datos de formularios. Para trabajar con esta herramienta es necesario crear lo que en programación se conoce como bean, es decir un contenedor que simule el formulario. Para llevarlo a cabo es necesario montar una clase Java donde se definan los mismos campos que tenga el formulario visual y que extienda de la clase ActionForm. Además será necesario definirlos en el fichero struts-config.xml y asociarlos a uno o varios de los Actions del ActionMapping, que habíamos visto en el apartado de configuración.

```
<form-bean
        name="registro"
        type="net.franet.forms.RegistroActionForm">
</form-bean>
```
(5.8)

Cuando un usuario envía los datos de un formulario estos llegan al controlador. En este punto se creará, si no lo estaba ya, el bean y se rellenará con los datos introducidos por el usuario. En el bean de registro de usuario, consultable en el anexo, podemos distinguir dos propiedades: usuario y password. Estas serán los responsables de albergar los datos que el usuario haya introducido en los campos de la pantalla. El bean por esencia no debería tener más contenido que sus propiedades y los métodos para recuperar, almacenar, inicializar o validar esas propiedades.

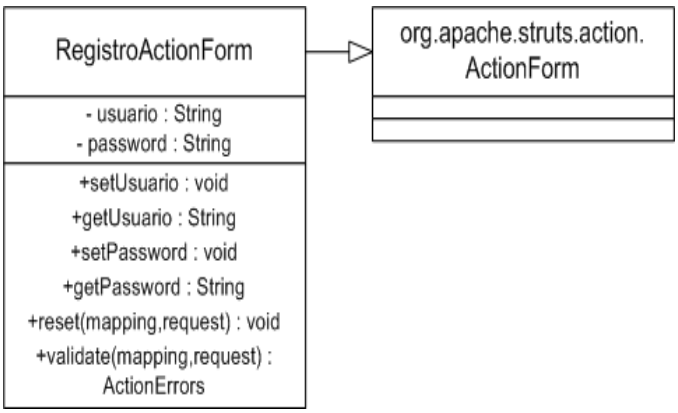


Fig 5.11 Diagrama de clases de un ActionForm

Hasta ahora hemos definido el bean como un objeto que viaja de la vista al controlador, sin embargo, también es posible realizar el camino inverso. Si por ejemplo queremos que un formulario ya disponga de datos al ser visualizado,

caso de modificación de un empleado, podemos rellenar ese bean en el controlador y disponer de la información en la vista.

El último aspecto que analizaremos del ActionForm es su método de validación. Estas validaciones se hacen en servidor e incrementan el nivel de seguridad del programa, al ser más efectivas que las proporcionadas en el cliente con Validator. Si se introducen unas validaciones evitaremos accesos innecesarios al Action, y en consecuencia a la base de datos. Se definen a nivel de bean dentro de un método llamado validate y se ejecutan antes de invocar la clase (action) que resolverá la funcionalidad. También es requisito a nivel de etiqueta action del fichero struts-config.xml activar la propiedad validate y definir una propiedad input con una url a la que redirigir en caso de que las validaciones se incumplan, que por regla general será la misma JSP en la que nos encontrábamos.

En la imagen 5.12 podemos observar las acciones que realiza el servidor al recibir una petición que tiene un ActionForm asociado.

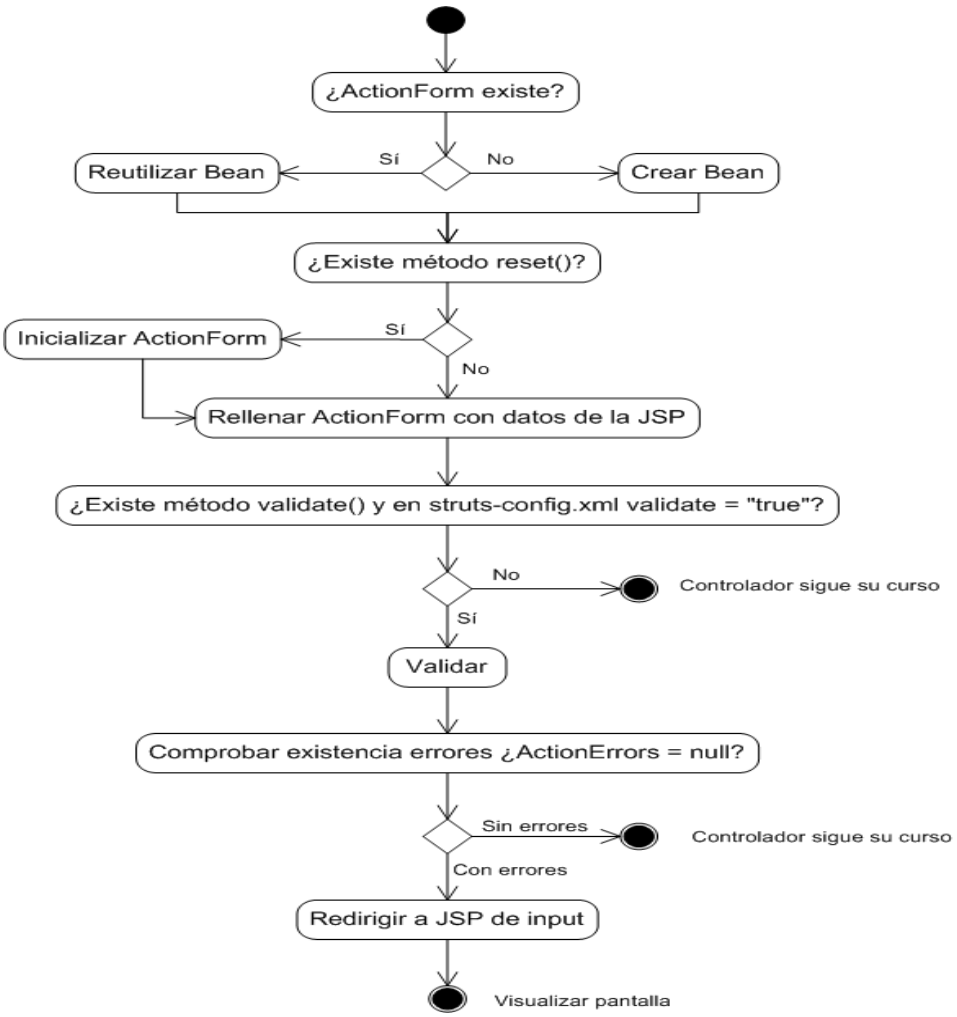


Fig 5.12 Diagrama de flujo de validación mediante ActionForm

5.7.2 ActionMessages

Uno de los puntos a tener en cuenta al introducir validaciones es la forma de informar al usuario en caso de que no se cumplan. Para validaciones mediante Struts Validator, al realizarse en el cliente, se invoca el método alert de JavaScript y aparece una ventana emergente. Caso a parte son las validaciones que se realizan a nivel de servidor ya que se debe devolver la pelota al cliente. Esta contestación puede ser una jsp de advertencia, pero es una solución engorrosa tanto a nivel de programación como de lógica de navegación.

Para solucionar esta problemática Struts proporciona los ActionMessages. Un ActionMessage es un objeto Java, creado en un ActionForm o un Action, que encapsula un mensaje (procedente del MessageResources.properties). Estos mensajes, pudiendo tener varios, se almacenan y cuando se muestra la jsp aparecen en el punto dónde hayamos colocado una etiqueta struts *messages* o *errors* de la biblioteca html.

5.7.3 Action

El Action es la clase del Servlet encargada de dar funcionalidad a la aplicación: mostrar clientes, crear artículos, inactivar empleados… Como ya hemos visto en el apartado de configuración estas clases deben definirse en el fichero struts-config.xml.

```
<action
        path="/modificarFamilias"
        type="net.franet.actions.articulos.familias.FamiliasModificarAction"
        name="modificarFamilias"
        scope="request"
        validate="false"
        input="/pages/articulos/familias/modificarFamilias.jsp">
        <forward name="success"
                path="/pages/articulos/familias/modificarFamilias.jsp"/>
</action>
```

(5.9)

Esta clase extiende de org.apache.struts.action.Action y tiene defendido el método execute. Este método es el que se invoca por defecto y se le pasan una serie de parámetros:

- **mapping**. El ActionMapping que hemos defendido en el fichero struts-config.xml y que relaciona esta clase.
- **form**. Si el Action tiene asociado un ActionForm lo recibiremos por este parámetro y podremos así recuperar y tratar sus datos.
- **request**. Es la petición HTTP que hemos recibido.

- **response**. Es la respuesta HTTP que se está creando.

En este método situaremos todo el código referente a la funcionalidad. Si necesitamos acceder a la base de datos, lo haremos a través de un servicio, tal y como se ha explicado en la capa del modelo. Una vez el Action llegue a su fin, será momento de escoger que JSP se tiene que visualizar. El método execute retorna un objeto ActionForward que contiene un identificador (5.10).

```
return mapping.findForward("success");
```
(5.10)

En este ejemplo (5.10) el identificador es *success*. La relación entre identificador y JSP está definida en el struts-config a nivel de action bajo la etiqueta forward (5.11).

```
<forward name="success"
path="/pages/articulos/datos/verArticulos.jsp"/>
```
(5.11)

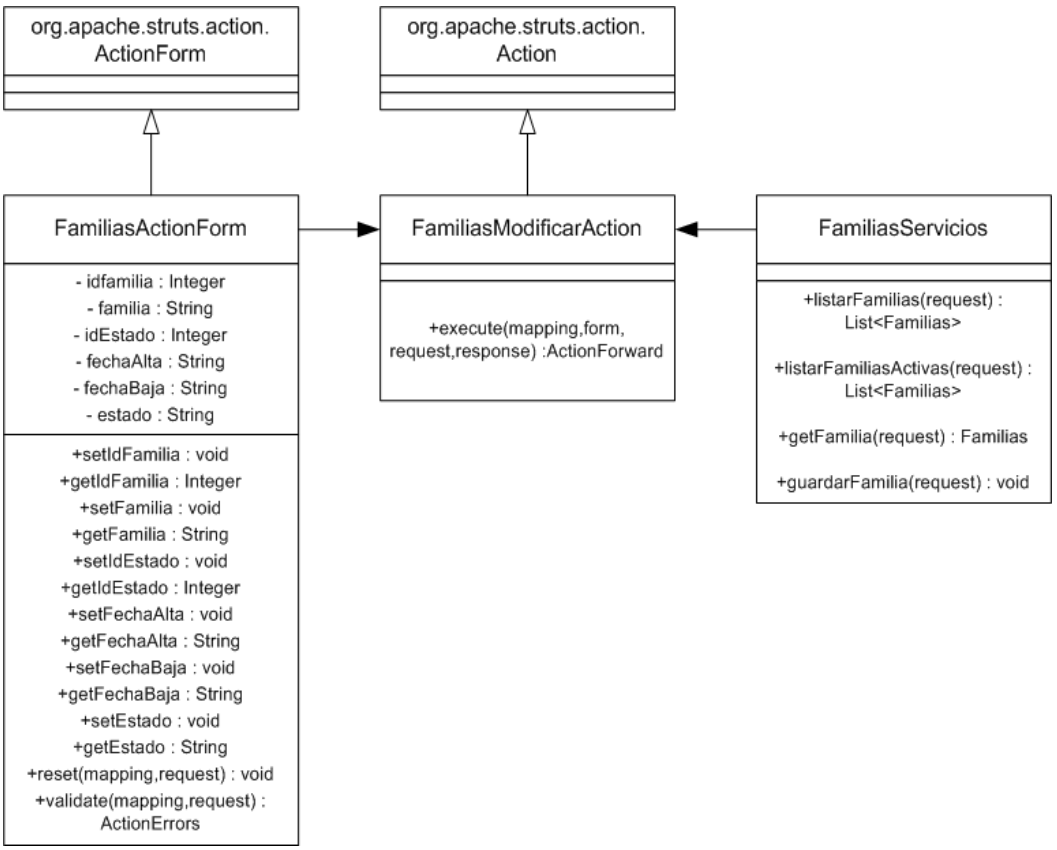


Fig 5.13 Diagrama de clases Action interactuando con ActionForm.

Para acabar, es interesante saber que debemos hacer uso del parámetro request (figura 5.14) para almacenar objetos que queremos que lleguen a la JSP de destino.

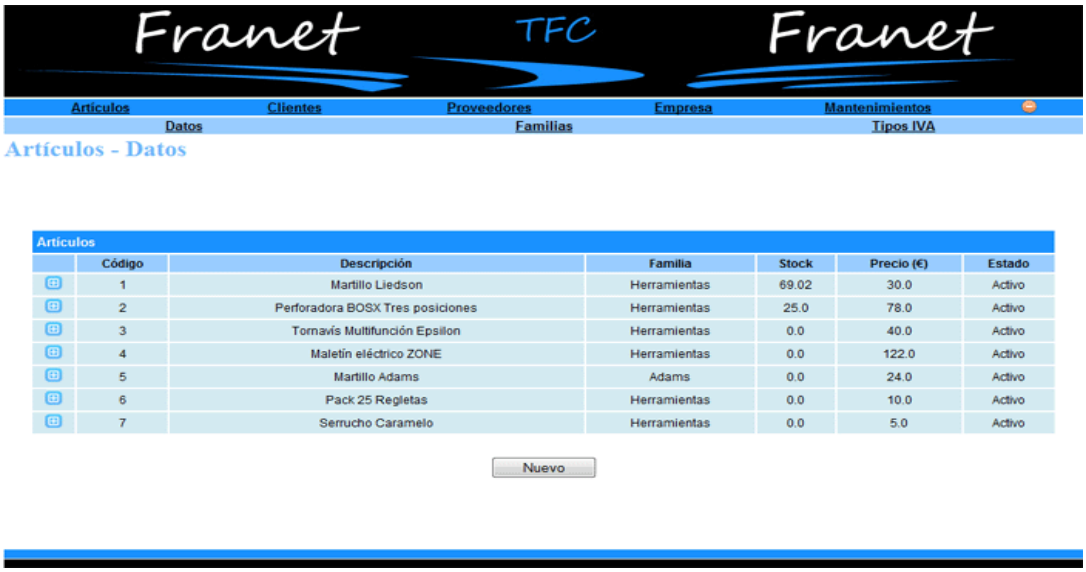


Fig 5.14 Captura con múltiples datos que se han recuperado de la request

5.7.4 DispatchAction

Hasta ahora sólo disponíamos de una funcionalidad por Action. Intuitivamente es lo más coherente, sin embargo a veces no resulta eficiente. Es el caso de una JSP con varios botones en un único formulario ya que por convenio un formulario tiene que tener un único Action asociado. Llegados a este punto tenemos dos formas de actuar. La primera sería conmutar por javascript el Action cada vez que se seleccione un botón. La segunda es montar el Action de forma que extienda de una clase de Struts, DispatchAction, que permite implementar varios métodos en un mismo Action y así a cada método se le asocian sus funciones correspondientes.

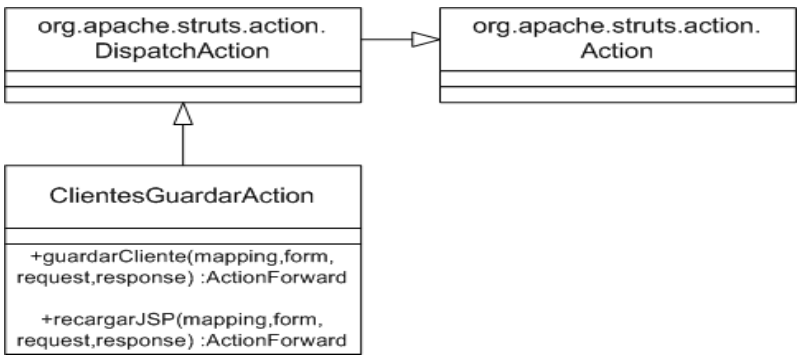


Fig 5.15 Diagrama de clases con ejemplo de DispatchAction

Además trabajando de esta segunda forma tendremos la ventaja de reducir clases, código, configuración e incluso facilitar el reaprovechamiento del ActionForm asociado puesto que todos los métodos usarán el mismo bean. Con el uso de métodos nos vemos con la obligación de realizar una pequeña modificación en el struts-config.xml.

```
<action
        path="/guardarEmpresa"
        parameter="method"
        type="net.franet.actions.empresa.EmpresaGuardarAction"
        name="guardarEmpresa"
        scope="request"
        validate="false"
        input="/pages/empresa/datos/modificarEmpresa.jsp">
        <forward name="guardarEmpresa"
                path="/pages/empresa/datos/modificarEmpresa.jsp"/>
        <forward name="recargarJSP"
                path="/pages/empresa/datos/modificarEmpresa.jsp"/>
</action>
```

(5.12)

La propiedad *parameter* (5.12) será la que informe de la existencia de un nuevo atributo a nivel de URL que informará del método del DispatchAction a ejecturar. Un aspecto a tener en cuenta es que si se usa DispatchAction no debemos definir el método execute o la aplicación seguirá entendiendo que es el método por defecto.

5.7.5 RequestProcessor

Las peticiones/requests HTTP procedentes de un usuario son recibidas por la clase del Servlet de Struts, ActionServlet. Esta clase hará de frontera y delegará la responsabilidad de tratar la petición a otra clase llamada RequestProcessor.

El objeto RequestProcessor será el encargado de relacionar peticiones con funcionalidades, es decir será el responsable de seleccionar el Action correcto. Podemos decir que esta será realmente la clase que realiza las tareas de controlador. Para conocer que Action debe invocar el RequestProcessor es necesario extraer cierta información de la URI y usarla para encontrar en el fichero struts-config.xml la clase del Action. La primera vez que se llama a un Action se crea una instancia de la clase y se almacena en caché de forma que en el futuro no es necesario repetir el proceso. Otra tarea asignada a la misma clase es la de rellenar el ActionForm y realizar su proceso de validación tal y como habíamos comentado.

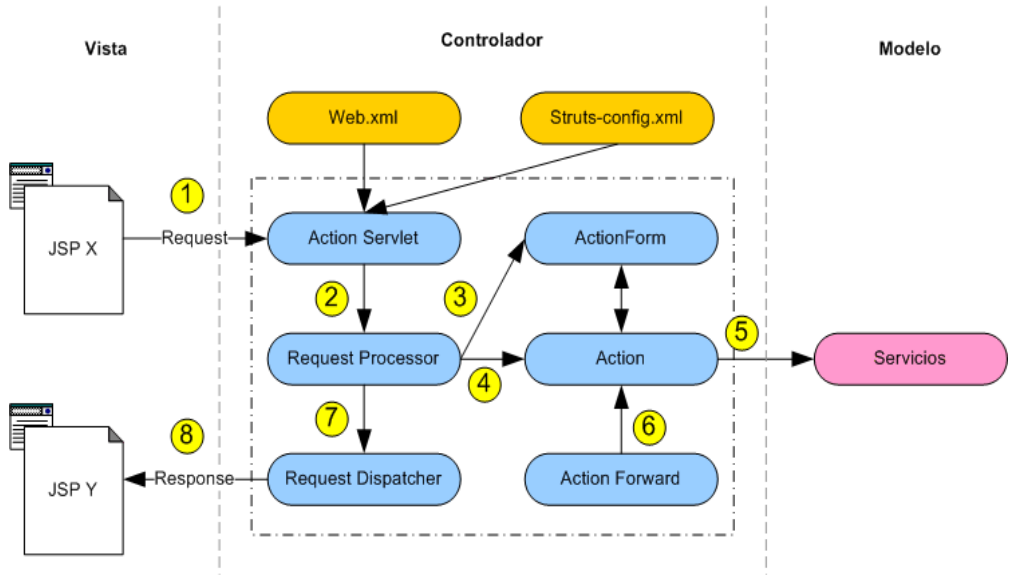


Fig 5.16 Cículo del framework Struts

El último método en ejecutarse de la clase RequestProcessor será el encargado de recoger el ActionForward que ha devuelto la ejecución del Action y acudir a la clase RequestDispatcher para generar la respuesta.

Es posible cambiar el comportamiento del RequestProcessor si se hereda de una subclase. En Franet se ha modificado el método que se ejecuta justo antes de invocar un Action. Así en función de las condiciones de entrada se decidirá si se acepta la ejecución de un Action o se redirige a otro. Esto ha sido útil para poder llevar a cabo un sistema de autenticación y otro de perfiles. Veremos más en detalle este aspecto en los próximos apartados.

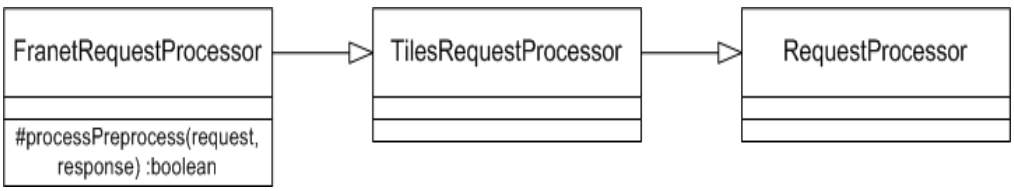


Fig 5.17 Diagrama de clases del RequestProcessor de Franet

La clase creada se llama FranetRequestProcessor.java y deberá extender de TilesRequestProcessor porque Tiles requiere de un controlador de peticiones propio. Será necesario añadir a nivel de struts-config.xml la clase que realizará las tareas de controlador (5.13).

```
<controller
processorClass="net.franet.comun.FranetRequestProcessor"/>
```

(5.13)

5.8 Autenticación

Esta es la primera de las funcionalidades que hemos realizado haciendo uso del procesador de peticiones propio. Llevar a cabo un proceso de autenticación en una aplicación como Franet era imprescindible y aunque realizar una pantalla de registro de usuario es teóricamente simple, comprobar que en todo momento ese usuario está autenticado no lo es tanto.

El registro de usuario se realiza en el Action asociado a la pantalla inicial. Cuando el usuario introduce sus datos el controlador los recibe y valida en la base de datos. Si son coherentes se añade un nuevo atributo en sesión para informar que el usuario ha sido autenticado.

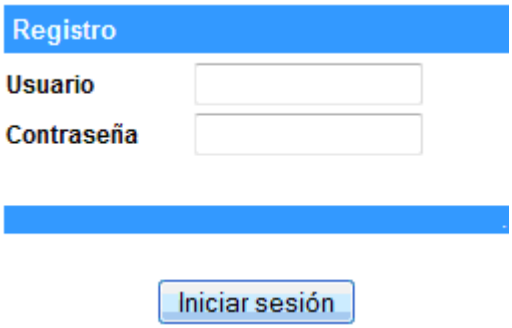


Fig 5.18 Formulario de registro de Franet

El RequestProcessor propio entra en juego a partir de entonces. Cada vez que llega una petición al controlador se comprobará la existencia del atributo de autenticación de la sesión. Esto permitirá que si un usuario accede desde una url diferente a la página de registro será expulsado por no haberse autenticado. Vemos el flujo completo en la siguiente figura (5.19).

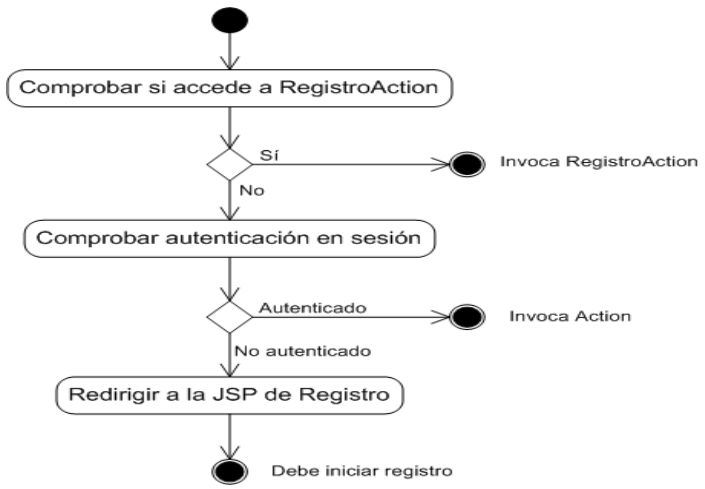


Fig 5.19 Diagrama de flujo de comprobación de autenticación.

5.9 Perfiles

Para realizar un control de los permisos que se asignan a cada usuario se ha implantado una funcionalidad en el RequestProcessor. Cada vez que un usuario realiza una acción (crear pedido, modificar factura, …) se lleva a cabo una comprobación de sus privilegios.

Si el cargo (Tabla Cargos) del usuario es Administrador este tendrá permitirá cualquier acción sobre la aplicación tal y como estaba definido en los requisitos funcionales.

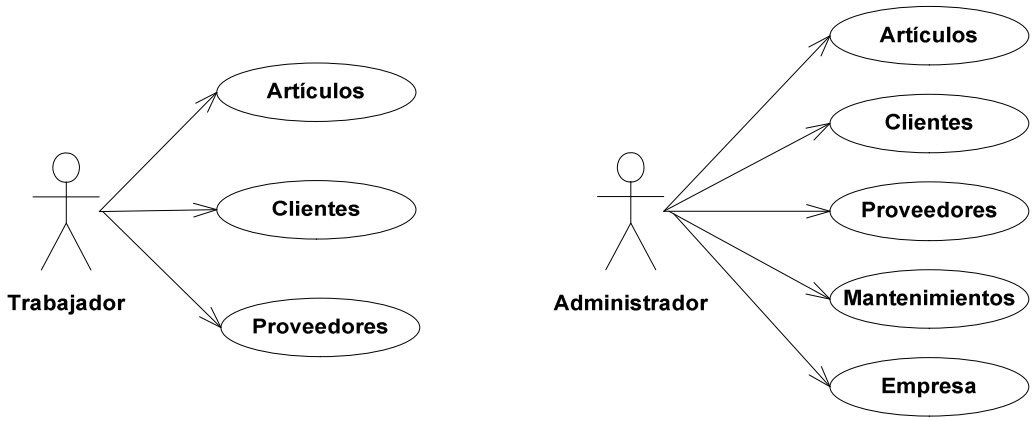


Fig 5.20 Diagrama de casos de uso de perfiles en Franet

En caso de que el cargo no sea administrador se hará una validación en base de datos. Cada empleado tiene definidos unos permisos sobre la aplicación y cada uno de ellos equivale a Action (figura 5.22). Por tanto un usuario no podrá llevar a cabo una acción si no tiene asociado el Action que la lleva a cabo. En el caso de no tener permiso se le avisará con una pantalla de aviso como vemos en la siguiente imagen.

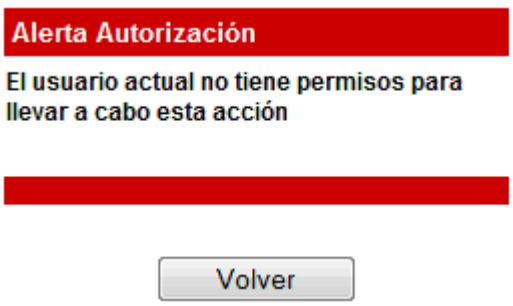


Fig 5.21 Pantalla de aviso cuando autenticación.

Así pues podemos lograr que un empleado pueda tiener acceso a todas las acciones de Artículos y Clientes y sólo algunas de Proveedores, como por ejemplo de consulta. De esta forma se realiza un control muy minucioso de las acciones que puede realizar cada empleado. Y si por ejemplo un usuario que se encarga del sector Clientes por una época tendrá que dar soporte a la sección de Proveedores no hará falta cambiarle el cargo, simplemente adjudicarle unas cuantas acciones.

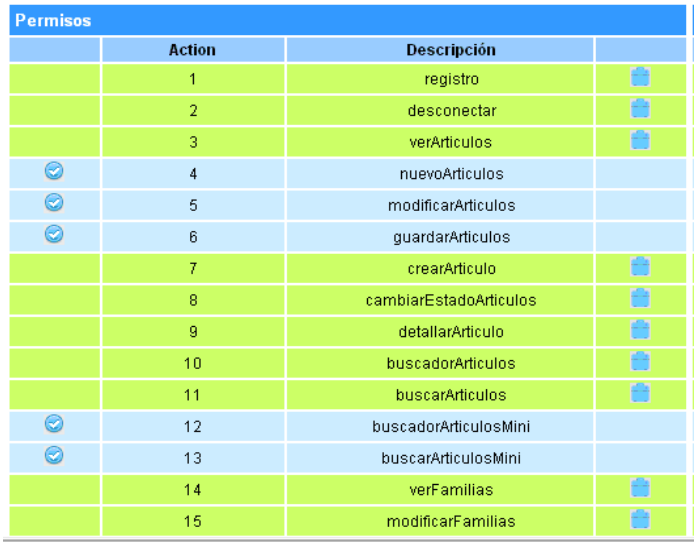


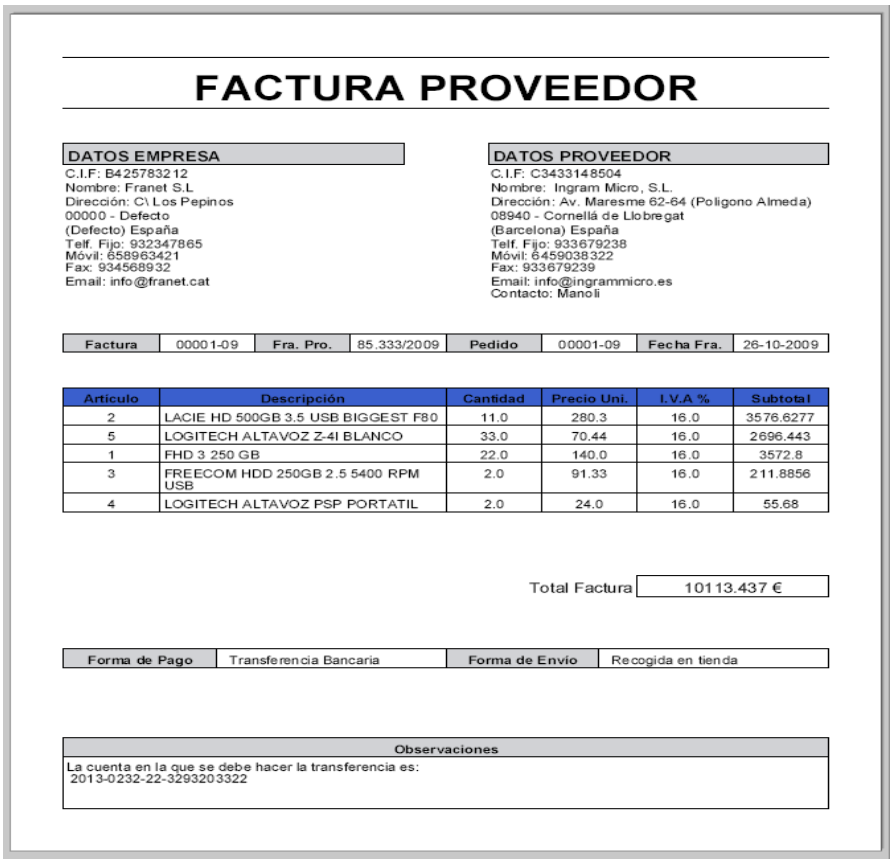
Fig 5.22 Pantalla de asignación de permisos, a nivel de Action.

5.10 Itext. Generación de PDFs

Uno de los requerimientos de Franet era listar facturas o presupuestos. Eso comporta la necesidad de extraer información del aplicativo para su envío por mail o impresión. Para cubrir esta necesidad se optó por generar dinámicamente archivos pdf.

La librería itext permite crear archivos pdf, pero no es una herramienta que deba usar el cliente. Está enfocado a trabajar en el servidor de forma de que el usuario haga una petición, Struts mediante un Action use la librería de itext y se devuelva una respuesta en forma de pdf. Para visualizar los archivos será necesario un programa que logre abrir este tipo de archivos. Para entornos web como Franet una versión avanzada de Adobe Reader es una buena alternativa.

El archivo pdf se construye a partir de objetos itext que simulan celdas, tablas, párrafos… y dentro se almacena toda la información que se necesite procedente de la base de datos. En la siguiente figura (5.23) podemos observar el resultado de generar una factura.



FACTURA PROVEEDOR

DATOS EMPRESA
C.I.F: B425783212
Nombre: Franet S.L
Dirección: C\ Los Pepinos
00000 - Defecto
(Defecto) España
Telf. Fijo: 932347865
Móvil: 658963421
Fax: 934568932
Email: info@franet.cat

DATOS PROVEEDOR
C.I.F: C3433148504
Nombre: Ingram Micro, S.L.
Dirección: Av. Maresme 62-64 (Polígono Almeda)
08940 - Cornellá de Llobregat
(Barcelona) España
Telf. Fijo: 933679238
Móvil: 6459038322
Fax: 933679239
Email: info@ingrammicro.es
Contacto: Manoli

Factura	00001-09	Fra. Pro.	85.333/2009	Pedido	00001-09	Fecha Fra.	26-10-2009

Artículo	Descripción	Cantidad	Precio Uni.	I.V.A %	Subtotal
2	LACIE HD 500GB 3.5 USB BIGGEST F80	11.0	280.3	16.0	3576.6277
5	LOGITECH ALTAVOZ Z-4I BLANCO	33.0	70.44	16.0	2696.443
1	FHD 3 250 GB	22.0	140.0	16.0	3572.8
3	FREECOM HDD 250GB 2.5 5400 RPM USB	2.0	91.33	16.0	211.8856
4	LOGITECH ALTAVOZ PSP PORTATIL	2.0	24.0	16.0	55.68

Total Factura 10113.437 €

Forma de Pago	Transferencia Bancaria	Forma de Envio	Recogida en tienda

Observaciones
La cuenta en la que se debe hacer la transferencia es:
2013-0232-22-3293203322

Fig 5.23 Archivo PDF creado con itext

5.11 Secure Sockets Layer (SSL)

Se ha hecho uso del protocolo SSL por motivos de privacidad, seguridad e integridad de datos. Además es un protocolo bastante transparente por lo que no perjudica a los usuarios de la aplicación.

Aunque el protocolo solo autentica el servidor, el cliente se asegura de que se conecta al servidor correcto y evita que terceros lo suplanten. Una vez la comunicación esté establecida ambos extremos podrán ser considerados seguros.

Para configurar SSL el primer paso es generar un certificado. Este será el encargado de almacenar las claves pública y privada. Estas claves serán usadas por un cifrado asimétrico que se usa, al establecer la conexión, para transmitir de forma segura la clave simétrica. Una vez la comunicación está estable el protocolo funcionará con un cifrado simétrico.

Para crear el certificado se hace uso de la herramienta Keytool. Durante la generación del certificado se requerirán datos como nombre y apellidos, unidad

organizativa, organización, ciudad, provincia y país. El certificado se almacena en un fichero de nombre de keystore y se sitúa en la carpeta conf de Apache.

El certificado de Franet contiene la siguiente información:

Emitido para	
Nombre común (CN)	Franet Application
Organización (O)	Franet Org
Unidad organizativa (OU)	Franet UO
Número de serie	4A:CA:73:0F
Emitido por	
Nombre común (CN)	Franet Application
Organización (O)	Franet Org
Unidad organizativa (OU)	Franet UO
Validez	
Emitido el	06/10/2009
Expira el	03/01/2010
Huellas digitales	
Huella digital SHA1	D1:FC:46:8A:59:D0:7A:C8:6E:AB:61:46:02:F2:A1:DB:FD:F9:F8:3C
Huella digital MD5	70:BD:9E:C9:2A:26:72:5E:65:13:FD:20:0E:08:F7:83

Fig 5.24 Certificado autofirmado de Franet

El certificado generado es autofirmado, lo que significa que no ha recibido el visto bueno de una entidad certificadora. El único inconveniente en este sentido será que el cliente deberá dar el consentimiento a ese certificado cada vez que inicie una comunicación.

Una vez creado y almacenado el certificado se debe configurar el servidor para que trabaje con el protocolo.

En el archivo server.xml:

- Se debe configurar el puerto 8443, usado para conexiones seguras, de modo que se active el uso del SSL. Tendremos que indicarle la localización del keystore.
- Se debe redireccionar el puerto 8080 al 8443. Así se asegura que cualquier conexión al servidor acabe en el puerto seguro.

En el archivo web.xml:

- Se debe añadir una nueva entrada de seguridad <security-constrain>

Con estas actuaciones habremos logrado configurar el SSL sin que sea necesaria ninguna actuación en la máquina cliente.

CAPÍTULO 6. PRUEBAS

6.1 Pruebas en la aplicación

Todas las funcionalidades de la aplicación han sido probadas antes de dar por finalizado el desarrollo. Primero se llevaron a cabo pruebas unitarias y una vez finalizada la aplicación se probaron el funcionamiento en conjunto. Estos test han tenido los siguientes propósitos:

- Evitar que se introduzcan datos inconsitentes en el sistema (campos obligatorios en blanco, campos con valores no esperados o mal formulados, etc.)

- Comprobar que la base de datos se ha actualizado tal y como define la funcionalidad.

- Validar que la navegación se realiza correctamente, se redirige a las pantallas esperadas y se visualizan todos los datos correctamente.

A continuación (tablas 5.3 y 5.4) hacemos una relación simplificada entre funcionalidades y resultado obtenido. En este apartado encontraremos las pruebas del gestor de proveedores y empresa. El resultado del resto podrá ser consultadas en el anexo.

Tabla 5.3 Pruebas realizadas en gestor de artículos

MÓDULO PROVEEDORES		
Entidad Relacionada	Acción	Resultado
Proveedor	Buscar	-Filtrar registros según criterios
	Listar	-Mostrar lista registros
	Crear	-Registro creado
	Modificar	-Registro modificado
	Cambiar estado	-Cambio de estado del registro (Activo / Inactivo)
Pedido	Listar	-Mostrar lista registros
	Crear	-Registro creado -Crear detalles del pedido -Crear detalle artículo
	Consultar	-Registro consultado
	Cambiar estado	-Cambio de estado del registro (Activo / Inactivo)
Factura	Generar	-Crear registro -Cambiar estado Pedido -Actualizar detalle artículo
	Consultar	-Registro consultado

Tabla 5.4 Pruebas realizadas en gestor de empresa

MÓDULO EMPRESA		
Entidad Relacionada	Acción	Resultado
Empresa	Modificar	-Registro modificado
Empleado	Listar	-Mostrar lista registros
	Crear	-Registro creado
	Modificar	-Registro modificado
	Cambiar estado	-Cambio de estado del registro (Activo / Inactivo)
	Asignar permiso	- Asignar un permiso de navegación al empleado
	Desasignar permiso	- Quitar un permiso de navegación al empleado
Cargos	Listar	-Mostrar lista registros
	Crear	-Registro creado
	Modificar	-Registro modificado
	Cambiar estado	-Cambio de estado del registro (Activo / Inactivo)
Actions	Listar	-Mostrar lista registros
	Crear	-Registro creado
	Modificar	-Registro modificado
	Cambiar estado	-Cambio de estado del registro (Activo / Inactivo)

Una vez montada la aplicación en un hipotético escenario real y validados los requerimientos funcionales pasamos a comprobar que se sumplan los requisitos no funcionales.

6.2 Verificación de los requisitos no funcionales

En esta sección verificaremos los diferentes requerimientos no funcionales definidos en el capítulo de Análisis y especificaciones. Se reserva un apartado (6.3) para el requisito de carga.

6.2.1 Calidad

- **Integridad y seguridad**. Se ha implantado el protocolo SSL que dota al aplicativo de una seguridad suficiente para una aplicación de estas características y se ha desarrollado un sistema de roles para controlar los privilegios que cada usuario tiene.
- **Flexibilidad y protabilidad**. La estructuración modular del aplicativo permite aplicar mejoras y facilita posibles cambios de escenario. Además

la arquitectura es fácilmente modificable y no se requiere grandes recursos para que el programa entre en funcionamiento.

- **Eficiencia y fiabilidad**. Se ha procurado no usar el mínimo número de accesos a base de datos, se ha intentado definir un modelo de base de datos coherente y dentro de las posibilades se ha intentado actuar de la forma más eficiente posible. En cuanto a fiabilidad se han detectado escasos errores y habría que realizar o bien pruebas de stress o bien tener usar la aplicación durante un largo periodo de tiempo para comprobar que a lo largo del tiempo sigue siendo fiable.
- **Interoperabilidad**. La estructura Web que se ha definido no tiene que ser impedimento para unir la aplicación con otras.

6.2.2 Coste

El coste del sofware ha sido cero. En todo momento se han usado herramientas y tecnologías con licencias de sofware libre y eso ha permitido lograr un coste nulo. A nivel de recursos humanos no se ha sumado coste porque los dos miembros del proyecto han sido los encargados de realizar todo el trabajo. Finalmente dejamos el coste del hardware a disposición de los posibles destinatarios de la aplicación. Puede ser viable reutilizar equipos para evitar también coste de hardware.

6.2.3 Requerimientos tecnológicos

Se ha diseñado Franet con tecnología Java y las tecnologías complementarias usadas no tenían limitaciones en cuanto a sistema operativo. Se comprueba además que cualquier resolución superior a 800 x 600 es válida para trabajar en Franet. Volvemos a resaltar el hecho de haber evitado el uso de tecnologías propietarias para el desarrollo habiendo potenciado el software libre.

6.2.4 Interfícies

Se han creado las pantallas siguiendo patrones lo que ha permitido que su visualización sea muy poco agresiva. También se ha procurado repartir la información de forma clara y que fuera fácilmente manipulable por el usuario. El sistema de menú combinado con los múltiples gestores del aplicativo permitirá una navegación simple y bastante intuitíva.

6.3 Pruebas de carga

El sistema final debe ser validado a nivel de carga para acabar de cumplir los requisitos no funcionales. Se estimó que la aplicación no tardaría más de 10

segundos en mostrar las páginas teniendo a 20 usuarios conectados como caso más crítico. Se han realizado dos pruebas definiendo dos escenarios diferentes.

La primera prueba se ha realizado en una Intranet con la ayuda del programa WAPT. El usuario modelo que se ha definido lo podemos considerar como muy activo puesto que accede a muchas páginas de funcionalidades complejas y que son las más críticas en la base de datos. La duración del test ha sido de 3 minutos con un volumen creciente de usuarios (de 5 a 16).

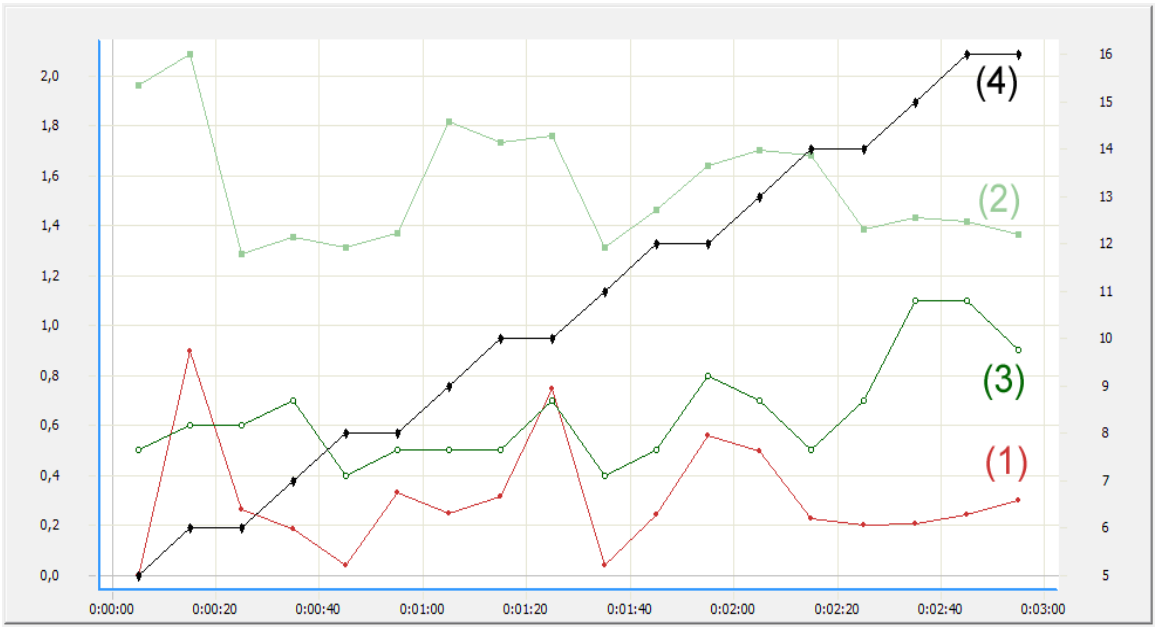


Fig 6.1 Valores de carga en intranet

- **Tiempo medio de descarga (1)**: tiempo que transcurre entre que se inicia a recibir la página hasta que se visualiza entera.
- **Tiempo medio de resupesta (2)**: tiempo que transcurre des de que el usuario envia una petición hasta que recibe toda la página.
- **Páginas por segundo (3)**
- **Número de usuarios (4)**

Los resultados de la prueba intranet (figura 6.1) demuestran que el incremento de usuarios no afecta en gran medida al rendimiento de la aplicación, pudiendo comprobar que el tiempo medio de respuesta se sitúa entre uno y dos segundos. No se constatan errores durante la ejecución de la prueba y tampoco se comprueba ningún pico de mal rendimiento.

El segundo escenario es una simulación de una red WAN como Internet. Para ello necesitamos el programa WAPT, para multiplicar los usuarios conectados, y

el programa TMnetSim, para simular las condiciones de la red. En este caso la prueba dura 5 minutos y el número de usuarios oscila entre 15 y 20 a lo largo de la ejecución. Se establecen 300ms como tiempo de tiempo de retardo en la red.

Los resultados que los podemos ver en la figura 6.2 y como en el caso anterior no se constatan valores demasiado críticos situándose el tiempo medio de respuesta por encima de un segundo pero muy lejos de los 10 segundos como máximo que se habían fijado como requisito.

En este segundo escenario no aparece tampoco ningún error por lo que se cumple el requisito que estipula que el número de errores no puede superar el 1% de del total de operaciones.

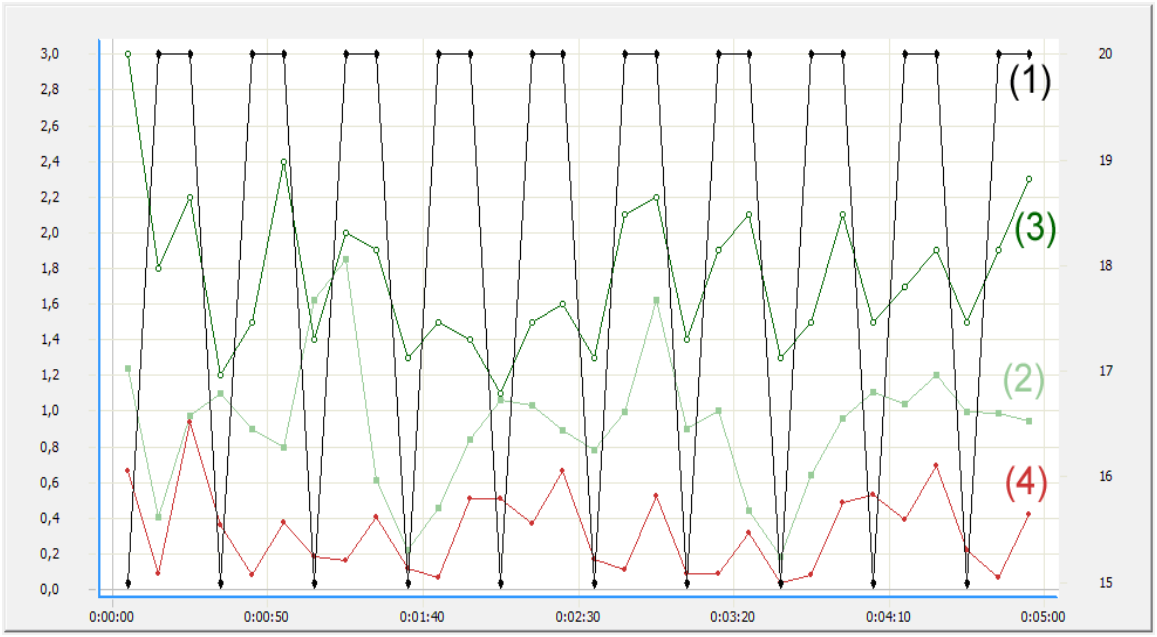


Fig 6.1 Valores de carga en red WAN

CAPÍTULO 7. CONCLUSIONES

7.1 Verificación de objetivos

El propósito de este proyecto era la construcción de una aplicación desde la fase de diseño hasta la fase de pruebas de la implementación. Los objetivos cumplidos son los siguientes:

- Realizar un estudio de los requisitos que serán necesarios para el desarrollo de la aplicación tanto a nivel funcional como a nivel no funcional. La cara más teórica del proyecto ha sido documentada y ha sido básica a la hora de llevar a cabo el desarrollo.
- Diseñar la base de datos. Para logralo ha sido necesario un conocimiento pleno de los requerimientos y una previa familiarización con el modelo entidad-relación.
- Iniciar un aprendizaje de las tecnologías que giran en torno al proyecto, con framework Struts, Hibernate y MySQL como principales protagonistas. Esta fase ha sido clave para adquirir conocimientos de cara a potenciar al máximo la implementación del código.
- Implantar la solución. La implantación ha supuesto el reto más importante con la combinación de todas las tecnologías y con el propósito de lograr que la aplicación se adapte al diseño teórico. Este período ha significado un esfuerzo y un aprendizaje continuo, que ha permitido que finalmente se haya logrado el total desarrollo de las funcionalidades inicialmente propuestas. Además han ido surgiendo nuevas mejoras que serán tratadas en el apartado 7.3.
- Realizar las pruebas ha permitido comprobar que la aplicación es útil, eficiente, intuitiva y en todo momento se ha intentado hacer un buen uso de los recursos disponibles.

En resumen, se puede dar por finalizado el proyecto habiendo cumplido los objetivos propuestos y con la certeza de poderlo implantar en cualquier entorno que cumpla con los requisitos del diseño.

7.2 Balance de las herramientas usadas

En primer lugar se debe resaltar el lenguaje de programación Java. Este lenguaje ha demostrado ser intuitivo y muy práctico a la hora de ser combinado con tecnologías como Hibernate o Spring. Su estructura web permite además crear una arquitectura descentralizada.

También ha jugado un papel importante el Framework Struts cuyo estudio e implantación ha facilitado mucho el desarrollo y nos ha proporcionado un

conocimiento más amplio de cómo se estructura una aplicación web y qué papel lleva a cabo cada elemento de la arquitectura modelo vista controlador.

Hibernate también merece mención especial al adquirir un papel más protagonista del que inicialmente se esperaba. Es una herramienta muy potente y no sólo es una forma de trasladar objetos a entidades de base de datos y viceversa. Permite entre otras cosas crear relaciones entre objetos de forma sencilla y es capaz incluso de crear automáticamente la base de datos tomando como referencia el mapeo de objetos.

Spring, MySQL, Eclipse y otras tecnologías y herramientas también han tomado parte en el transcurso del proyecto y han ocupado el rol esperado en la solución final.

Quizás Spring ha sido la tecnología que menos juego ha dado. Al tener ciertas similitudes con Struts podía suceder, pero no se esperaba que su impacto fuera tan menor. Finalmente, tan solo ha facilitado el trabajo a la hora de definir un patrón DAO y es que el hecho de que tome la posesión del JDBC e Hibernate no se ha demostrado excesivamente útil e incluso ha dificultado algunas acciones tal y como detallaremos en el siguiente apartado.

7.3 Mejoras y líneas futuras

A medida que evoluciona el aplicativo fueron surgiendo diferentes campos de mejora.

En cuanto a Spring sería interesante conocer mejor como interactúa con Hibernate o el JDBC. Uno de los principales problemáticas relacionadas con este hecho han sido las transacciones ya que Spring no permitía un control ágil de la conexión a base de datos y al mismo tiempo resultaba muy costoso lograr que interactuase correctamente con Hibernate. Para llevar a cabo las transacciones de Franet se han encontrado soluciones interesantes pero si complicáramos el escenario sería del todo necesario actuar en este punto.

Hay algunas mejoras a nivel de navegación y uso de la aplicación que podrían ser tomadas en consideración como un uso mayor de drilldowns (links) para moverse de un punto a otro sin ayuda del menú, el uso de Web Services para proporcionar información externa (por ejemplo de artículos o proveedores), y el uso de librerías de pdf o excel para extraer de forma más simple la información de la aplicación para su impresión o archivo. Cabe remarcar, sin embargo que tal y como se ha estructurado el programa, basado en diferentes módulos y funcionalidades bien detalladas, este tipo de mejoras no deberían provocar un impacto grande si fuera necesaria su implantación en un futuro.

Por último, es una opción muy interesante el uso de un programa como Subversion para que varias personas puedan trabajar al mismo tiempo en el

proyecto y en todo momento se guarden copias de seguridad en un servidor. En este proyecto se ha desestimado por falta de recursos y para no añadirle un coste económico al proyecto. Para un proyecto de mayor envergadura sería muy recomendable.

7.4 Impacto medioambiental

El uso de la tecnología inevitablemente implica la existencia de un impacto medioambiental. Este proyecto no es una excepción.

Tanto el servidor como las máquinas cliente necesitarán electricidad y en la producción de esta hallaremos el principal impacto medioambiental. Tanto en el caso del servidor como en el de los clientes se puede trabajar con equipos que incluyan un programa de rendimiento, sin embargo, al ser una solución diseñada para terceros será voluntad del receptor hacer un uso responable.

En la fase de desarrollo del proyecto el principal impacto mediambiental vuelve a ser la necesidad de conectase a la red eléctrica. Tanto el diseño del aplicativo, como su implantación o la redacción de la memória son claros ejemplos.

Quizás un punto a favor de una aplicación como Franet puede ser que está destinada a tareas de gestión, que permitirán substituir o al menos reducir el coste de papel, de impresión, etc. que suele relacionarse con estas actividades.

7.5 Conclusiones personales

Rara vez se da la posibilidad de realizar un proyecto de principio a fin. Los dos integrantes responsables del desarrollo de este trabajo coincidimos plenamente en que ha supuesto un reto muy importante englobarse en cada fase del proyecto. Ha sido muy beneficioso en el aspecto de formación y nos ha aportado una experiencia que difícilmente hubiéramos adquirido de otra forma.

Trabajar en equipo ha sido a su vez un punto clave. A lo largo de la carrera nos han ido introduciendo los valores de trabajar en equipo, y ha sido en este proyecto donde lo hemos podido explotar al máximo. Es un trabajo que se ha compaginado con la vida laboral de ambos integrantes y difícilmente se hubiera finalizado sin un buen entendimiento y coordinación.

Otro aspecto indiscutible es que el proyecto nos ha servido para especializarnos en la rama más informática de la carrera. Era a nivel personal el principal objetivo que nos marcábamos y hemos adquirido un sin fin de conceptos de aplicaciones distribuidas, base de datos, entornos de desarrollo… que durante la carrera habían tenido muy poco o ningún protagonismo.

Se han logrado los objetivos a nivel de proyecto, se han cumplido largamente nuestros propósitos personales y a nivel educativo este proyecto creemos que ha sido muy enriquecedor.

CAPÍTULO 8. REFERENCIAS BIBLIOGRÁFICAS

[1] Silberschatz/Korth/Sudarshan, Fundamentos de Diseño de bases de datos, McGrawHill, 5ª Edición (2006).

[2] Arnold Doray, Apache Struts, Anaya Multimedia, 1ª Edición (2007)

[3] Bauer, Christian; King, Gavin, Hibernate in action, Hanning (2ª Edición)

[4] Metodología del diseño conceptual
http://www3.uji.es/~mmarques/f47/apun/node84.html
Introducción al modelo entidad relación

[5] Diseño lógico de base de datos
http://bd.eui.upm.es/BD/docbd/tema/tema2.pdf
Diagramas simples del modelo entidad relación

[6] Conectividad JDBC
http://www.itapizaco.edu.mx/paginas/JavaTut/froufe/parte21/cap21-3.html
Explicación de JDBC

[7] Conectividad JDBC
http://struts.apache.org/
Explicación de JDBC

[8] Spring
http://www.springsource.org/
Página official de la tecnología Spring. Foro con posts interesantes.

[9] Installing and using Eclipse Web Tools
http://rm.mirror.garr.it/mirrors/eclipse/technology/phoenix/demos/install-wtp/install-wtp.html
Manual para realizar debug en una aplicación Web mediante Eclipse

[10] Rose India
http://www.roseindia.net/
Web con multiples ejemplos de como implantar Struts y sus funcionalidades

[11] Configuring Struts
http://www.allapplabs.com/struts/configuring_struts.htm
Manual de ayuda para la configuración de Struts

[12] Struts in Action: Developing Applications with Tiles
http://www.developer.com/java/ent/article.php/10933_2192411_1
Capítulo introductorio a Tiles

[13] Manual básico Struts
http://www.programacion.com/java/tutorial/joa_struts/4/
Introducción a Struts

[14] Beans ActionForm
http://jose.cl/muestramemoria.php?nodoname=node28.html
Explicación detallada de ActionForm

[15] Extending Struts
http://www.onjava.com/pub/a/onjava/2004/11/10/ExtendingStruts.html
Descripción del RequestProcessor de Struts

[16] Using the Validator Framework with Struts
http://www.onjava.com/pub/a/onjava/2002/12/11/jakartastruts.html?page=1
Validator de Struts

[17] Patrón “Data Access Object”
http://www.proactiva-calidad.com/java/patrones/DAO.html
Detalle de los patrones DAO

[18] Beans, BeanFactory and the ApplicationContext
http://static.springframework.org/spring/docs/1.2.x/reference/beans.html
Explicación de los beans de Struts

[19] El protocolo SSL
http://penta2.ufrgs.br/gereseg/unlp/tut1998/ssl.htm
Marco teórico del protocolo SSL

[20] Secure Socket Layer (SSL)
http://www.geocities.com/CapeCanaveral/2566/ssl/ssl.html
Teoría complementaria del protocolo SSL

[21] Configuración de Tomcat para usar SSL
http://libra.dif.um.es/~edumart/asignaturas/servicios/apuntes/TomcatSSL.pdf
Manual para la implantación de SSL en el servidor Apache Tomcat

[22] Step by step: Configuring SSL Under Apache
http://www.onlamp.com/pub/a/onlamp/2008/03/04/step-by-step-configuring-ssl-under-apache.html
Manual de implantación de SSL en Apache

CAPÍTULO 9. ANEXOS

9.1 Pantallas

En este apartado podemos hallar las pantallas más representativas de la aplicación Franet.

9.1.1 Pantalla Registro de Usuarios

9.1.1.1 Vista Acceso Aplicación

Es la vista primaria y única de la pantalla. Permite al empleado acceder al menú de la aplicación. Primero se comprobará que se le permite el acceso comprobando que exista su usuario, su password, su cargo y que el empleado esté activo.

Un empleado activo significa que tiene privilegios para acceder a la aplicación y por lo tanto puede desempeñar tareas de Usuario.

Tabla 9.1 Vista acceso aplicación

Elemento	Tipo	Comentarios
Formulario Acceso		
Registro Usuario	Título	
Usuario	Texto Editable	
Password	Texto Editable	
Botones		
Entrar	Botón	Previo comprobación del registro da acceso a la Vista Principal

9.1.2 Pantalla Principal

9.1.2.1 Vista Principal

Es la vista primaria de Pantalla Principal y estará formada tan sólo por una página introductoria de la empresa. Su única función es tener acceso al menú.

Una vez hayamos accedido a esta vista el menú será visible en todo momento. Tendrá cinco opciones a seleccionar.

- Clientes

El botón Clientes nos llevará a la Pantalla Clientes.

- Proveedores

El botón Proveedores nos llevará a la Pantalla Proveedores.

- Artículos

El botón empresa mostrará un submenú con las siguientes opciones:

- Datos
- Familias
- Mantenimientos

El botón mantenimientos mostrará un submenú con las siguientes opciones:

- Forma envío
- Forma pago
- Países
- Poblaciones
- Tipos IVA
- Empresa

El botón empresa mostrará un submenú con las siguientes opciones:

- Datos
- Empleados
- Cargos
- Actions

9.1.3 Pantalla Clientes

9.1.3.1 Vista Listado Clientes

Es la vista primaria de la Pantalla de Clientes. En ella tendremos una lista de todos los clientes.

Tabla 9.2 Vista Listado Clientes

Elemento	Tipo	Comentarios
Listado Clientes		
Clientes	Título	
NIF	Texto No Editable	
Nombre	Texto No Editable	
Apellido 1	Texto No Editable	
Apellido 2	Texto No Editable	
Móvil	Texto No Editable	
Estado	Texto No Editable	
Botones		
Nuevo	Botón	Da acceso a la Vista Nuevo Cliente
Modificar	Botón en el listado (imagen lupa)	Da acceso a la Vista Modificar Cliente
Presupuestos	Botón	Da acceso a la Vista Listado Presupuestos
Pedidos	Botón	Da acceso a la Vista Listado Pedidos

9.1.3.2 Vista Nuevo Cliente

En esta vista podremos introducir un nuevo cliente.

Tabla 9.3 Vista nuevo cliente

Elemento	Tipo	Comentarios
Formulario Nuevo Cliente		
Nuevo Cliente	Título	
NIF	Texto Editable	
Fecha	Texto Editable	
Nombre	Texto Editable	
Apellido 1	Texto Editable	
Apellido 2	Texto Editable	
Dirección	Texto Editable	
Población	Texto No Editable	
Provincia	Texto No Editable	
País	Texto No Editable	
CP	Texto Editable	Habrá alguna forma de seleccionar el CP y se informen Población, Provincia y País de forma automática.
Teléfono Fijo	Texto Editable	
Teléfono Móvil	Texto Editable	

Email	Texto Editable	
Botones		
Guardar	Botón	Permite crear un nuevo cliente en base de datos.

9.1.3.3 Vista Modificar Cliente

En esta vista podremos ver y modificar un cliente.

Tabla 9.4 Vista modificar cliente

Elemento	Tipo	Comentarios
Formulario Modificar Cliente		
Modificar Cliente	Título	
NIF	Texto Editable	
Fecha	Texto Editable	
Estado	Texto Editable	Seleccionar por combobox.
Nombre	Texto Editable	
Apellido 1	Texto Editable	
Apellido 2	Texto Editable	
Dirección	Texto Editable	
Población	Texto No Editable	
Provincia	Texto No Editable	
País	Texto No Editable	
CP	Texto Editable	Habrá alguna forma de seleccionar el CP y se informen Población, Provincia y País de forma automática.
Teléfono Fijo	Texto Editable	
Teléfono Móvil	Texto Editable	
Email	Texto Editable	
Acepta Servicio	Texto Editable	Mediante Check Box
Botones		
Modificar	Botón	Permite modificar el cliente en base de datos.

9.1.3.4 Vista Listado Presupuestos

Aquí tendremos todos los Presupestos realizados a un cliente. Al seleccionar uno podremos realizar distintas acciones.

Tabla 9.5 Vista listado presupuestos

Elemento	Tipo	Comentarios
Listado Presupuestos		
Presupuestos Cliente	Título	
ID Presupuesto	Texto No Editable	
Fecha Vigente	Texto No Editable	Fecha de Creación o Modificación.
Estado	Texto No Editable	
Botones		
Nuevo	Botón	Da acceso a la Vista Nuevo Presupuesto
Modificar	Botón	Da acceso a la Vista Modificar Presupuesto
Borrar	Botón	Pasará el estado a anulado. Siempre y cuando el estado este pendiente.

9.1.3.5 Vista Nuevo Presupuesto

En esta vista tendremos dos formularios y un listado de artículo. El listado inicialmente estará vacío. El primer formulario servirá introducir un nuevo artículo al listado. Con todos los artículos introducidos en el listado podremos hacer el presupuesto. El segundo formulario nos servirá para acabar de cumplimentar el presupuesto.

Tabla 9.6 Vista nuevo presupuesto

Elemento	Tipo	Comentarios
Formulario Nuevo Artículo Nuevo Presupuesto		
Nuevo Presupuesto	Título	
Añadir Artículo	Título	
ID Artículo	Texto Editable	
Cantidad	Texto Editable	
Botones Nuevo Artículo		
Añadir	Botón	Al seleccionar apareceremos en la misma vista pero habrá un nuevo registro en el listado.
Listado Artículos Nuevo Presupuesto		
ID Artículo	Texto No Editable	
Cantidad	Texto No Editable	
Precio Unidad	Texto No Editable	
Precio Total	Texto No Editable	
Botones Listado Artículos Nuevo Presupuesto		
Total Presupuesto	Texto No Editable	Será la cantidad por la que se define el presupuesto.

Eliminar Artículo	Botón	Eliminará del listado el artículo seleccionado.
Formulario Nuevo Presupuesto		
Observaciones	Texto Editable	
Botones Nuevo Presupuesto		
Guardar	Botón	Generará un nuevo presupuesto. Iremos a la vista Listado Presupuestos donde habrá el nuevo registro.

9.1.3.6 Vista consulta/genera Factura

Vista donde tendremos los detalles de una factura para generar o consultar.

Tabla 9.7 Vista acceso aplicación

Elemento	Tipo	Comentarios
Factura Cliente	Título	
ID Factura	Texto No Editable	
Fecha Alta	Texto No Editable	
Estado	Texto No Editable	
Botones		
Listar	Botón	Generará PDF
Volver	Botón	Ir al listado de presupuestos

9.1.4 Pantalla Proveedores

9.1.4.1 Vista Listado Proveedores

Es la vista primaria de la Pantalla de Proveedores. En ella tendremos una lista de todos los proveedores.

Tabla 9.8 Vista listado proveedores

Elemento	Tipo	Comentarios
Listado Proveedores		
Proveedores	Título	
Número Proveedor	Texto No Editable	
Nombre	Texto No Editable	
NIF	Texto No Editable	
Botones		

Nuevo	Botón	Da acceso a la Vista Nuevo Proveedor
Modificar	Botón	Da acceso a la Vista Modificar Cliente
Borrar	Botón	Llamará al Action que se ocupará de cambiarle el estado al proveedor a Inactivo.
Pedidos	Botón	Da acceso a la Vista Listado Pedidos
Facturas	Botón	Da acceso a la Vista Listado Facturas

9.1.4.2 Vista Nuevo Proveedor

En esta vista podremos introducir un nuevo proveedor.

Tabla 9.9 Vista nuevo proveedor

Elemento	Tipo	Comentarios
Formulario Nuevo Proveedor		
Nuevo Proveedor	Título	
Número Proveedor	Texto Editable	
NIF	Texto Editable	
Estado	Texto Editable	'Activo' por defecto. Seleccionar por combobox.
Nombre	Texto Editable	
Dirección	Texto Editable	
Población	Texto Editable	
CP	Texto Editable	
País	Texto Editable	
Teléfono 1	Texto Editable	
Teléfono 2	Texto Editable	
Mail	Texto Editable	
Entidad	Texto Editable	
Oficina	Texto Editable	
DC	Texto Editable	
Cuenta	Texto Editable	
Contacto	Texto Editable	
www	Texto Editable	
Botones		
Guardar	Botón	Permite crear un nuevo proveedor en base de datos.

9.1.4.3 Vista Modificar Proveedor

En esta vista podremos ver y modificar un proveedor.

Tabla 9.10 Vista modificar proveedor

Elemento	Tipo	Comentarios
Formulario Modificar Proveedor		
Nuevo Proveedor	Título	
Número Proveedor	Texto Editable	
NIF	Texto Editable	
Estado	Texto Editable	'Activo' por defecto. Seleccionar por combobox.
Nombre	Texto Editable	
Dirección	Texto Editable	
Población	Texto Editable	
CP	Texto Editable	
País	Texto Editable	
Teléfono 1	Texto Editable	
Teléfono 2	Texto Editable	
Mail	Texto Editable	
Entidad	Texto Editable	
Oficina	Texto Editable	
DC	Texto Editable	
Cuenta	Texto Editable	
Contacto	Texto Editable	
Página Web	Texto Editable	
Botones		
Guardar	Botón	Permite crear un nuevo proveedor en base de datos.

9.1.4.4 Vista Listado Pedidos

Aquí tendremos todos los Pedidos realizados a un proveedor. Al seleccionar uno podremos realizar distintas acciones.

Tabla 9.11 Vista listado pedidos

Elemento	Tipo	Comentarios
Listado Albaranes		
Pedidos Proveedor	Título	
ID Pedido	Texto No Editable	
Fecha Vigente	Texto No Editable	Fecha de Creación o Modificación.

Estado	Texto No Editable	
Botones		
Nuevo	Botón	Da acceso a la vista Nuevo Pedido
Consulta	Botón	Da acceso a la vista Consulta Pedido
Generar Factura	Botón	Creará la factura del pedido seleccionado.
Borrar	Botón	Llamará al Action que se ocupará de cambiarle el estado al albarán a anulado.

9.1.4.5 Vista Nuevo Pedido

En esta vista tendremos dos formularios y un listado de artículo. El listado inicialmente estará vacío. El primer formulario servirá introducir un nuevo artículo al listado. Con todos los artículos introducidos en el listado podremos hacer el presupuesto. El segundo formulario nos servirá para acabar de cumplimentar el presupuesto.

Tabla 9.12 Vista nuevo pedido

Elemento	Tipo	Comentarios
Formulario Nuevo Artículo Nuevo Pedido		
Nuevo Pedido	Título	
Añadir Artículo	Título	
ID Artículo	Texto Editable	
Cantidad	Texto Editable	
Botones Nuevo Artículo		
Añadir	Botón	Al seleccionar apareceremos en la misma vista pero habrá un nuevo registro en el listado.
Listado Artículos Nuevo Pedido		
ID Artículo	Texto No Editable	
Cantidad	Texto No Editable	
Botones Listado Artículos Nuevo Pedido		
Eliminar Artículo	Botón	Eliminará del listado el artículo seleccionado.
Formulario Nuevo Pedido		
Observaciones	Texto Editable	
Botones Nuevo Pedido		
Guardar	Botón	Generará un nuevo pedido. Iremos a la vista Listado Pedidos donde habrá el nuevo registro.

9.1.4.6 Vista Consulta Pedido

Será la vista para ver el detalle de los artículos que contienen el pedido. No se podrá modificar.

Tabla 9.13 Vista consulta pedido

Elemento	Tipo	Comentarios
Formulario Consulta Pedido		
Consulta Pedido	Título	
Observaciones	Texto No Editable	
Listado Artículos Consulta Pedido		
ID Artículo	Texto No Editable	
Cantidad	Texto No Editable	
Botones Consulta Pedido		
Volver	Botón	Irá a la vista listado Pedidos

9.1.4.7 Vista Consultar/Generar Facturas

Vista donde podremos consultar una factura o generarla.

Tabla 9.14 Vista consultar/generar faturas

Elemento	Tipo	Comentarios
Facturas Proveedor	Título	
ID Factura	Texto No Editable	
Núm. FRA Proveedor	Texto No Editable	
Fecha Alta	Texto No Editable	
Estado	Texto No Editable	
Botones		
Generar PDF	Botón	Generar PDF
Vovler	Botón	Ir a pedidos proveedor

9.1.5 Pantalla Artículos

9.1.5.1 Vista Listado Artículos

Es la vista primaria de la Pantalla de Artículos. En ella tendremos una lista de todos los artículos.

Tabla 9.15 Vista listado artículos

Elemento	Tipo	Comentarios
Listado Artículos		
Artículos	Título	
Código Artículo	Texto No Editable	
Descripción Artículo	Texto No Editable	
Código Familia	Texto No Editable	
Estado	Texto No Editable	
Botones		
Nuevo	Botón	Da acceso a la Vista Nuevo Artículo
Modificar	Botón	Da acceso a la Vista Modificar Artículo

9.1.5.2 Vista Nuevo Artículo

En esta vista podremos introducir un nuevo artículo.

Tabla 9.16 Vista nuevo artículo

Elemento	Tipo	Comentarios
Formulario Nuevo Artículo		
Nuevo Artículo	Título	
Código Artículo	Texto Editable	
Descripción Artículo	Texto Editable	
Estado	Texto Editable	'Activo' por defecto. Seleccionar por combobox.
Código de Familia	Texto Editable	
Descripción Familia	Texto No Editable	Informada cuando se introduzca el código de Familia.
Observaciones	Texto Editable	
Botones		
Guardar	Botón	Permite crear un nuevo artículo en base de datos.

9.1.5.3 Vista Modificar Artículo

En esta vista podremos introducir un nuevo artículo.

Tabla 9.17 Vista modificar artículo

Elemento	Tipo	Comentarios
Formulario Modificar Artículo		
Nuevo Artículo	Título	

Código Artículo	Texto Editable	
Descripción Artículo	Texto Editable	
Estado	Texto Editable	'Activo' por defecto. Seleccionar por combobox.
Código de Familia	Texto Editable	
Descripción Familia	Texto No Editable	Informada cuando se introduzca el código de Familia.
Observaciones	Texto Editable	
Botones		
Modificar	Botón	Permite modificar un artículo en base de datos.

9.1.6 Pantalla Empleados

9.1.6.1 Vista Listado Empleados

Es la vista primaria de la Pantalla Empleados. En ella tendremos una lista de todos los empleados. Al seleccionar una entrada podremos seleccionar alguno de los botones.

Tabla 9.18 Vista pantalla empleados

Elemento	Tipo	Comentarios
Listado Empleados		
Empleados	Título	
Usuario	Texto No Editable	
Nombre	Texto No Editable	
Apellido 1	Texto No Editable	
Apellido 2	Texto No Editable	
Cargo	Texto No Editable	
Botones		
Nuevo	Botón	Da acceso a la Vista Nuevo Usuario
Modificar	Botón	Da acceso a la Vista Modificar Usuario

Las entidades que serán necesarias para esta vista son:

- Empleados

9.1.6.2 Vista Nuevo Empleado

Esta pista nos permitirá registrar a un usuario previa comprobación que los campos Password y Confirmar Password sean idénticos.

Tabla 9.19 Vista nuevo empleado

Elemento	Tipo	Comentarios
Formulario Nuevo Empleado		
Nuevo Usuario	Título	
Código	Texto Editable	
Estado	Texto Editable	Activo por defecto. Seleccionable por ComboBox.
Nombre	Texto Editable	
Apellido 1	Texto Editable	
Apellido 2	Texto Editable	
Cargo	Texto Editable	Seleccionable por ComboBox.
Usuario	Texto Editable	
Password	Texto Editable	
Confirmación Password	Texto Editable	
Botones		
Guardar	Botón	Llama al Action asociado para crear un nuevo registro en el servidor después de hacer las respectivas verificaciones.

9.1.6.3 Vista Modificar Empleado

Esta pista nos permitirá modificar un usuario previa comprobación que los campos Password y Confirmar Password sean idénticos.

Tabla 9.20 Vista modificar empleado

Elemento	Tipo	Comentarios
Formulario Modificar Empleado		
Modificar Usuario	Título	
Código	Texto Editable	
Estado	Texto Editable	Seleccionable por ComboBox.
Nombre	Texto Editable	
Apellido 1	Texto Editable	
Apellido 2	Texto Editable	
Cargo	Texto Editable	Seleccionable por ComboBox.
Usuario	Texto Editable	
Password	Texto Editable	
Botones		
Guardar	Botón	Llama al Action asociado para modificar los datos en el servidor después de hacer las respectivas verificaciones.

9.2 Pruebas aplicación

Tabla 9.21 Pruebas módulo artículos

MÓDULO ARTÍCULOS		
Entidad Relacionada	Acción	Resultado
Artículo	Buscar	-Filtrar registros según criterios
	Listar	-Mostrar lista registros
	Crear	-Registro creado
	Modificar	-Registro modificado
	Cambiar estado	-Cambio de estado del registro (Activo / Inactivo)
	Desglosar	-Se muestran los detalles del artículo (Creando primero un pedido del proveedor)
Família	Listar	-Mostrar lista registros
	Crear	-Registro creado
	Modificar	-Registro modificado
	Cambiar estado	-Cambio de estado del registro (Activo / Inactivo)
Tipo IVA	Listar	-Mostrar lista registros
	Crear	-Registro creado
	Modificar	-Registro modificado
	Cambiar estado	-Cambio de estado del registro (Activo / Inactivo)

Tabla 9.22 Pruebas módulo clientes

MÓDULO CLIENTES		
Entidad Relacionada	Acción	Resultado
Cliente	Buscar	-Filtrar registros según criterios
	Listar	-Mostrar lista registros
	Crear	-Registro creado
	Modificar	-Registro modificado
	Cambiar estado	-Cambio de estado del registro (Activo / Inactivo)
Presupuestos	Listar	-Mostrar lista registros
	Crear	-Registro creado -Detalles del presupuesto creados
	Consultar	-Registro consultado
	Cambiar estado	-Cambio de estado del registro (Activo / Inactivo)
Factura	Generar	-Crear registro -Cambiar estado Presupuesto -Actualizar detalle artículo
	Consultar	-Registro consultado

Tabla 9.23 Pruebas módulo artículos

MÓDULO MANTENIMIENTOS		
Entidad Relacionada	Acción	Resultado
Poblaciones	Buscar	-Buscar registro
	Listar	-Mostrar lista registros
	Crear	-Registro creado
	Modificar	-Registro modificado
	Cambiar estado	-Cambio de estado del registro (Activo / Inactivo)
Provincias	Listar	-Mostrar lista registros
	Crear	-Registro creado
	Modificar	-Registro modificado
	Cambiar estado	-Cambio de estado del registro (Activo / Inactivo)
Países	Listar	-Mostrar lista registros
	Crear	-Registro creado
	Modificar	-Registro modificado
	Cambiar estado	-Cambio de estado del registro (Activo / Inactivo)
Tipos Envíos	Listar	-Mostrar lista registros
	Crear	-Registro creado
	Modificar	-Registro modificado
	Cambiar estado	-Cambio de estado del registro (Activo / Inactivo)
Tipos Pago	Listar	-Mostrar lista registros
	Crear	-Registro creado
	Modificar	-Registro modificado
	Cambiar estado	-Cambio de estado del registro (Activo / Inactivo)

9.3 Secure Sockets Layer (SSL)

A continuación se hace una breve introducción a los diferentes protocolos que conforman el SSL.

9.3.1 SSL Record Protocol

Este es el protocolo que entra en juego una vez la sesión está estable. Será el encargado de transmitir los datos e incluso los mensajes de los otros protocolos SSL. La información procedente de la aplicación será cifrada mediante un algoritmo de clave simétrica el cual agiliza el proceso de cifrado/descifrado.

El procedimiento que sigue este protocolo cuando recibe datos de la aplicación empieza en la recogida de los mensajes de la aplicación y los divide en bloques de 16Kb como máximo. Estas particiones reciben el nombre de *records*.

Opcionalmente los bloques creados pueden ser comprimidos. Posteriormente se incluye una cabecera, unos datos de padding si fuera necesario y se añade el MAC. Estas siglas representan el resultado de una función y posterior algoritomo de hash y se genera para asegurar la integridad de los datos en destino. Por último se realiza el cifrado aplicando una clave negociada al inicializar la sesión y el bloque obtenido está preparado para ser transportado en la capa de transporte.

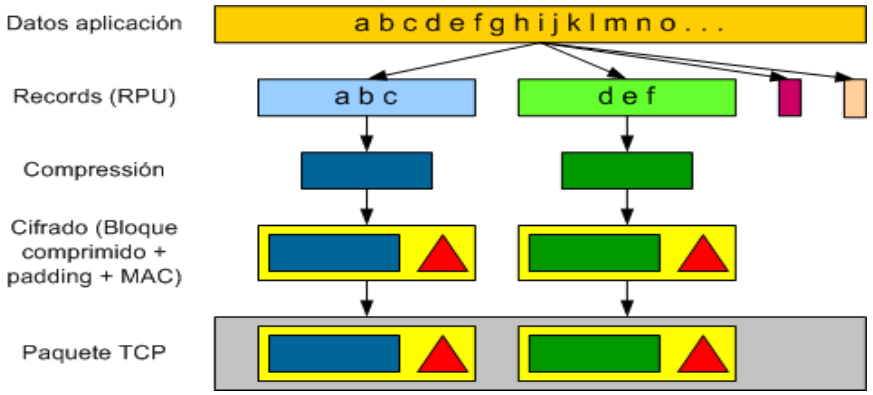


Fig 9.1 SSL Record Protocol

9.3.2 SSL Handshake Protocol

El protocolo SSL Handhake es el encargado de establecer una sesión mediante un intercambio de mensajes entre cliente y servidor. Proporcionar una sesión estable será imprescindible para que el protocolo SSL Record pueda llevar a cabo el cifrado. Durante el establecimiento de la sesión se negociaran algoritmos y claves para el cifrado de datos. Este será a la vez el protocolo que nos proporcione autenticación. El proceso de handshake es el siguiente:

- *Client Hello*. El cliente SSL envía un mensaje “client hello” con su versión de SSL y los algoritmos de encriptación que soporta.

- *Server Hello y envío de certificado*. El servidor responde con un mensaje “server hello” con el algoritmo de encriptación escogido y el id de sesión. Además le mandará su certificado digital con su clave pública. En este punto el servidor también puede pedir un certificado al cliente para que ambos extremos estén autenticados.

- *Aceptación del cliente*. El cliente verifica el certificado y genera una clave secreta aleatoria que cifrará con la clave pública del servidor. Esta clave(de sesión) será la usada por el protocolo SSL Record para cifrar los datos. Si fuera necesario enviar el certificado del cliente se realizaría en este punto. El cliente envía un mensaje “finished” cifrado con la nueva clave de sesión para informar que ha acabado este paso.

- *Aceptación del servidor*. El servidor descifra la clave que le manda el cliente gracias a la clave privada. Además autenticará el cliente en el caso de que le haya mandado certificado. Para anunciar que ha finalizado enviará un mensaje “finished” también cifrado con la clave de sesión.

- Un certificado contiene toda la información necesaria para la autenticación del servidor o del cliente, como puede ser dominio, el dueño, el domicilio, la fecha de validez e incluso la llave pública.

- Existen dos formas de obtenerlo. La primera es tramitarlo a partir de una autoridad de certifacacion(CA). El solicitante paga una cantidad establecida por la CA y tiene un tiempo de vigencia. La otra es obtener un certificado autofirmado, creado por nosotros mismos, con la desventaja que el navegador no lo reconocerá y pedirá al usuario una confirmación.

9.3.3 Alert Protocol y ChangeCipherSpec protocol

El protocolo de Alerta es el encargado de señalizar cualquier incidencia en la sesión SSL. Distingue dos tipos de mensajes, los avisos “warning” y los errores “fatal”. En el caso de recibir uno de los segundos la sesión queda anulada.

Por su parte el ChangeCipherSpec es un mensaje muy simple para confirmar que la sesión sigue establecida. Gracias a esto los extremos sabrán cuando deben renegociar la sesión.

9.4 Especificaciones de los casos de uso

En este apartado encontramos los diferentes casos de uso que no se han visto reflejados en el capítulo de Análisis y especificaciones.

9.4.1 Modificar

- Caso de uso: Modificar
- Actores: Gestor y Servidor.
- Propósito: El gestor quiere modificar un concepto de cualquier mantenimiento.
- Resumen: El usuario puede modificar el concepto de cualquier tipo de mantenimiento, después que se valide que la información introducida es coherente.
- Tipo: Primario.

Tabla 9.1 Curso tipo del acontecimiento

Acciones del Gestor	Respuesta del Sistema
1. Este caso de uso empieza cuando un usuario clicka en el botón de “Modificar” de cualquier concepto	
	2. El sistema muestra la pantalla con el concepto solicitado por el usuario con la información que hay en la bases de datos
3. El usuario introduce los datos que son necesarios, para modificar el concepto	
	4. El sistema comprueba que los datos introducidos por el usuario sean correctos.
	5. El sistema hace un update en la base de datos con la información que ha introducido el usuario
	6. El sistema vuelve a mostrar la lista de todos los conceptos que hay en la B.D.

Existe un curso alternativo. Error en el caso que los datos no sean los correctos, la aplicación devuelve un error informando el campo incorrecto.

9.4.2 Activar estado

- Caso de uso: Activar estado
- Actores: Gestor y Servidor.
- Propósito: El gestor quiere modificar el estado de cualquier mantenimiento.
- Resumen: El usuario puede reactivar cualquier concepto de cualquier tipo de concepto inactivado previamente.
- Tipo: Primario.

Tabla 9.2 Curso tipo del acontecimiento

Acciones del Gestor	Respuesta del Sistema
1. Este caso de uso empieza cuando un usuario clicka en el botón de “Modificar” de cualquier concepto	
	2. El sistema muestra la pantalla con el concepto solicitado por el usuario con la información que hay en la bases de datos
3. El usuario clicka en el botón de “Activar” el concepto	
	4. El sistema hace un update en la base de datos con el estado = 0
	5. El sistema vuelve a mostrar la lista de todos los conceptos que hay en la B.D.

9.4.3 Consulta

- Caso de uso: Consulta concepto
- Actores: Gestor y Servidor.
- Propósito: El gestor puede consultar la información de cualquier concepto si este este está en estado inactivo.
- Resumen: El gestor puede consultar la información de cualquier concepto clickando en el botón modificar de un concepto inactivado.
- Tipo: Secundario.

Tabla 9.3 Curso tipo del acontecimiento

Acciones del Gestor	Respuesta del Sistema
1. Este caso de uso empieza cuando un usuario clicka en el botón de “Modificar” de cualquier concepto inactivado.	
	2. El sistema muestra la pantalla con el concepto solicitado por el usuario con la información que hay en la bases de datos. No es modificable.

9.4.4 Mostrar detalle

- Caso de uso: Mostrar detalle
- Actores: Gestor y Servidor.
- Propósito: El gestor muestra el detalle de un concepto específico
- Resumen: El gestor puede mostrar el detalle de los conceptos de artículos, factura, presupuesto y albarán
- Tipo: Primario.

Tabla 9.4 Curso tipo del acontecimiento

Acciones del Gestor	Respuesta del Sistema
1. Este caso de uso empieza cuando un usuario selecciona el botón de “Detalle” de cualquier concepto	
	2. El sistema muestra la pantalla con los datos de la tabla detalle de cualquier concepto

9.5 Archivos de configuración Struts y extractos de código

En este apartado encontramos diferentes porciones de código y archivos de configuración que son referenciados desde el capítulo de Diseño e Implementación.

9.5.1 Web.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app
        PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
        "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">

<web-app>
        <display-name>Struts Blank Application</display-name>

        <!-- Spring context Configuration Begins-->
        <context-param>
                <param-name>log4jConfigLocation</param-name>
                <param-value>/WEB-INF/log4j.properties</param-value>
        </context-param>

        <context-param>
                <param-name>contextConfigLocation</param-name>
                <param-value>/WEB-INF/applicationContext-hibernate.xml</param-value>
        </context-param>

        <servlet>
                <servlet-name>context</servlet-name>
                <servlet-class>
                        org.springframework.web.context.ContextLoaderServlet
                </servlet-class>
                <load-on-startup>1</load-on-startup>
        </servlet>

        <!-- Standard Action Servlet Configuration (with debugging) -->
        <servlet>
        <servlet-name>action</servlet-name>
        <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
                <init-param>
                <param-name>config</param-name>
                <param-value>
                        /WEB-INF/struts-config.xml,
                        /WEB-INF/struts-config-articulos.xml,
                        /WEB-INF/struts-config-clientes.xml,
                        /WEB-INF/struts-config-proveedores.xml,
                        /WEB-INF/struts-config-mtos.xml,
                        /WEB-INF/struts-config-empresa.xml
                </param-value>
        </init-param>
        <init-param>
                <param-name>debug</param-name>
                <param-value>2</param-value>
        </init-param>
        <init-param>
                <param-name>detail</param-name>
                <param-value>2</param-value>
        </init-param>
        <load-on-startup>2</load-on-startup>
        </servlet>

        <!-- Cewolf Servlet - Graficas -->
        <servlet>
        <servlet-name>CewolfServlet</servlet-name>
```

```
        <servlet-class>de.laures.cewolf.CewolfRenderer</servlet-class>
        <init-param>
        <param-name>storage</param-name>
        <param-value>de.laures.cewolf.storage.TransientSessionStorage</param-value>
        </init-param>
        <init-param>
        <param-name>overliburl</param-name>
        <param-value>/WEB-INF/etc/overlib.js</param-value>
        </init-param>
        <init-param>
        <param-name>debug</param-name>
        <param-value>false</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
        </servlet>

        <!-- Standard Action Servlet Mapping -->
        <servlet-mapping>
                <servlet-name>action</servlet-name>
                <url-pattern>*.do</url-pattern>
        </servlet-mapping>

        <!-- Cewolf Servlet Mapping - Graficas -->
        <servlet-mapping>
           <servlet-name>CewolfServlet</servlet-name>
           <url-pattern>/cewolf/*</url-pattern>
        </servlet-mapping>

        <!-- The Usual Welcome File List -->
        <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
        </welcome-file-list>

        <!-- Struts Tag Library Descriptors -->
        <taglib>
        <taglib-uri>/tags/struts-bean</taglib-uri>
        <taglib-location>/WEB-INF/struts-bean.tld</taglib-location>
        </taglib>

        <taglib>
        <taglib-uri>/tags/struts-html</taglib-uri>
        <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
        </taglib>

        <taglib>
        <taglib-uri>/tags/struts-logic</taglib-uri>
        <taglib-location>/WEB-INF/struts-logic.tld</taglib-location>
        </taglib>

        <taglib>
        <taglib-uri>/tags/struts-nested</taglib-uri>
        <taglib-location>/WEB-INF/struts-nested.tld</taglib-location>
        </taglib>

        <taglib>
        <taglib-uri>/tags/struts-tiles</taglib-uri>
        <taglib-location>/WEB-INF/struts-tiles.tld</taglib-location>
        </taglib>

        <taglib>
        <taglib-uri>/tags/cewolf</taglib-uri>
        <taglib-location>/WEB-INF/cewolf.tld</taglib-location>
        </taglib>

</web-app>
```

9.5.2 Struts-config.xml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<!DOCTYPE struts-config PUBLIC
        "-//Apache Software Foundation//DTD Struts Configuration 1.2//EN"
        "http://jakarta.apache.org/struts/dtds/struts-config_1_2.dtd">

<struts-config>

<!-- ============================================== Form Bean Definitions -->

<form-beans>
            <form-bean
                        name="registro"
                        type="net.franet.forms.RegistroActionForm">
            </form-bean>
</form-beans>


<!-- ========================================== Global Exception Definitions -->

   <global-exceptions>
      <!-- sample exception handler
      <exception
         key="expired.password"
         type="app.ExpiredPasswordException"
         path="/changePassword.jsp"/>
      end sample -->
   </global-exceptions>


<!-- ========================================== Global Forward Definitions -->

   <global-forwards>

                        <forward name="common-stylesheet" path="/css/style.css"/>
   </global-forwards>


<!-- ========================================== Action Mapping Definitions -->

   <action-mappings>

                        <!-- Registro - Iniciar sesión -->
                        <action
         path="/registro"
         type="net.franet.actions.registro.RegistroAction"
         name="registro"
         scope="request"
         validate="true"
         input="/pages/registro/registro.jsp">
                                    <forward name="success" path="/pages/inicio/inicio.jsp"/>
                                    <forward name="error" path="/pages/registro/registro.jsp"/>
                        </action>

                        <!-- Desconectar - Cerrar sesión -->
                        <action
         path="/desconectar"
         type="net.franet.actions.registro.DesconectarAction"
         name="desconectar"
         scope="request"
         validate="false"
         input="/pages/registro/registro.jsp">
                                    <forward name="success" path="/pages/registro/registro.jsp"/>
                        </action>

            </action-mappings>
```

```
<!-- ========================================== Controller Configuration -->

        <controller
                processorClass="net.franet.comun.FranetRequestProcessor"/>

<!-- ======================================== Message Resources Definitions -->

   <message-resources parameter="MessageResources" />


<!-- ================================================ Plug Ins Configuration -->

  <!-- ======================================================= Tiles plugin -->

   <plug-in className="org.apache.struts.tiles.TilesPlugin" >

     <!-- Path to XML definition file -->
     <set-property property="definitions-config"
                value="/WEB-INF/tiles-defs.xml" />
     <!-- Set Module-awareness to true -->
     <set-property property="moduleAware" value="true" />
   </plug-in>


  <!-- ==================================================== Validator plugin -->

  <plug-in className="org.apache.struts.validator.ValidatorPlugIn">
    <set-property
       property="pathnames"
       value="/WEB-INF/validator-rules.xml,/WEB-INF/validation.xml"/>
  </plug-in>

</struts-config>
```

9.5.3 EmpleadosDaoImpl.java

```
package net.franet.entity.dao.impl;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.List;

import net.franet.entity.Empleados;
import net.franet.entity.dao.EmpleadosDAO;

import org.hibernate.Criteria;
import org.hibernate.criterion.Order;
import org.springframework.dao.DataAccessException;
import org.springframework.orm.hibernate3.support.HibernateDaoSupport;

/**
 * Accesos a base de datos para la clase Empleado
 *
 */
public class EmpleadosDAOImpl extends HibernateDaoSupport implements
                EmpleadosDAO {

        public void insertEmpleado(Empleados obj) throws DataAccessException{
                getHibernateTemplate().save(obj);
        }

        public void updateEmpleado(Empleados obj) throws DataAccessException{
                getHibernateTemplate().update(obj);
        }
```

```
	public Empleados getEmpleado(String id)
				throws DataAccessException {

			return (Empleados) getHibernateTemplate().get(
					Empleados.class, id);
	}

	@SuppressWarnings("unchecked")
	public List<Empleados> listarEmpleados() throws DataAccessException
	{
			Criteria crit = this.getSession().createCriteria(Empleados.class);
			crit.addOrder(Order.asc("apellido1"));
			return crit.list();
	}

	public Integer validarRegistro(String usuario, String password) throws
DataAccessException,java.sql.SQLException
	{
			Connection conn = this.getSession().connection();
			Statement smt = conn.createStatement();
			ResultSet rs;

			String query="select IDESTADO from empleados where USUARIO='"+usuario+"' and
PASSWORD='"+password+"'";
			rs=smt.executeQuery(query);

			Integer estadoUsuario = null;
			if(rs.next()== true)
				estadoUsuario = rs.getInt("IDESTADO");

			smt.close();
			rs.close();
			conn.close();

			return estadoUsuario;
	}

	public boolean comprobarUsuarioExistente(String usuario) throws DataAccessException,java.sql.SQLException
	{
			Connection conn = this.getSession().connection();
			Statement smt = conn.createStatement();
			ResultSet rs;

			String query="select NIF from empleados where USUARIO='"+usuario+"'";
			rs=smt.executeQuery(query);

			boolean existe = false;

			if(rs.next() == true)
				existe=true;
			else
				existe=false;

			smt.close();
			rs.close();
			conn.close();

			return existe;
	}

	public Empleados getUsuario(String user) throws DataAccessException, SQLException
	{
			Connection conn = this.getSession().connection();
			Statement smt = conn.createStatement();
			ResultSet rs;

			String query="select NIF from empleados where USUARIO='"+user+"'";
			rs=smt.executeQuery(query);
```

```
                String idEmpleado = null;
                if(rs.next()== true)
                        idEmpleado = rs.getString("NIF");

                smt.close();
                rs.close();
                conn.close();

                return (Empleados) getHibernateTemplate().get(
                                Empleados.class, idEmpleado);
        }

}
```

9.5.4 Articulos.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD//EN"
                "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping package="net.franet.entity">

        <class
           name="Articulos"
           table="articulos"
        >

           <id
              name="idArticulo"
              type="integer"
              column="idarticulo"
           >
              <generator class="increment" />
           </id>

                <property
              name="descripcion"
              type="string"
              column="articulo"
              not-null="true"
              length="50"
           />

           <many-to-one
                        name="idFamilia"
                        column="idfamilia"
                        class="Familias"
                        not-null="true"
                        lazy="false"
                        fetch="join"
                        unique="true"
                />

                <many-to-one
                        name="idIVA"
                        column="idiva"
                        class="TiposIva"
                        not-null="true"
                        lazy="false"
                        fetch="join"
                        unique="true"
                />

                <property
              name="observaciones"
              type="string"
              column="observaciones"
```

```
            not-null="false"
            length="1000"
        />

        <property
                    name="stock"
                    type="float"
                    column="stock"
                    not-null="true"
                    length="5"
                    precision="2"
            />

            <property
                    name="stockPedido"
                    type="float"
                    column="stock_pedido"
                    not-null="true"
                    length="5"
                    precision="2"
            />

            <property
                    name="precioVenta"
                    type="float"
                    column="precio_venta"
                    not-null="true"
                    length="5"
                    precision="2"
            />

            <many-to-one
                    name="idEstado"
                    column="idestado"
                    class="Estados"
                    not-null="true"
                    lazy="false"
                    fetch="join"
                    unique="true"
            />

        <property
            name="fechaAlta"
            type="timestamp"
            column="f_alta"
            not-null="true"
            length="19"
        />

          <property
            name="fechaBaja"
            type="timestamp"
            column="f_baja"
            not-null="false"
            length="19"
        />

        <set name="ArticulosDetalle" inverse="false" order-by="fecha" lazy="false" cascade="save-update,delete">
                    <key column="idarticulo" not-null="true"/>
                    <one-to-many class="ArticulosDetalle" />
            </set>

    </class>

</hibernate-mapping>
```

9.5.5 Style.css

```
<style type="text/css" media="screen">

/*** BODY ***/
.DIVlayout {
		text-align: center;
	}


/*** TABLE ***/

	.TABLEtitulo {
		/*border-collapse: collapse;*/
	}

	.TABLEtituloDet {
		border-collapse: collapse;
	}

	.TABLElistado {
		padding: 0;
		border:1px solid #000000;
		border-top:0px;
	}


/*** TD ***/

	.TDlistT {
		background-color: #99CCFF;
		font-family: Arial;
		font-size: 12px;
		padding: 2px;
		text-align: left;
		font-weight: bold;
		border: 0px;
		text-align: center;
	}

	.TDlistTDet {
		background-color: #ADFF2F;
		font-family: Arial;
		font-size: 12px;
		padding: 2px;
		text-align: left;
		font-weight: bold;
		border: 0px;
		text-align: center;
	}

	.TDlistF {
		background-color: #D3EAF1;
		font-family: Arial;
		font-size: 12px;
		padding: 2px;
		text-align: left;
		font-weight: normal;
		border: 0px;
		text-align: center;
	}
```

9.5.6 RegistroActionForm.java

```
package net.franet.forms;
```

```
import javax.servlet.http.HttpServletRequest;

import org.apache.struts.action.*;

public class RegistroActionForm extends ActionForm
{
	private static final long serialVersionUID = 1L;
	private String usuario=null;
	private String password=null;

	public void setUsuario(String usuario){
		this.usuario=usuario;
	}

	public String getUsuario(){
		return this.usuario;
	}

	public void setPassword(String password){
		this.password=password;
	}

	public String getPassword(){
		return this.password;
	}

	public void reset(ActionMapping mapping, HttpServletRequest request)
	{
		this.usuario=null;
		this.password=null;
	}

	public ActionErrors validate(
		ActionMapping mapping, HttpServletRequest request ) {
		ActionErrors errors = new ActionErrors();

		if( getUsuario() == null || getUsuario().length() < 1)
			errors.add("usuario",new ActionMessage("registro.error.usuario"));

		if(getPassword() == null || getPassword().length() < 1)
			errors.add("password",new ActionMessage("registro.error.password"));

		return errors;
	}

}
```

9.6 Entidades BBDD

Las diferentes entidades de base de datos de la aplicación Franet se pueden consultar en este apartado.

9.6.1 Actions

Contiene la información de los diferentes Actions.

actions

Field Name	Field Type	AllowNull	Key	Default	Extra
IDACTION	int(5)	NO	PRI		auto_increment
DESCRIPCION	varchar(200)	NO			
IDESTADO	int(2)	NO	MUL	0	
F_ALTA	timestamp	NO		CURRENT_TIMESTAMP	
F_BAJA	timestamp	YES			

Key Name	Field Name	NonUnique	Collation	Cardinality	Subpart	Packed
PRIMARY	IDACTION		A	2		
IDESTADO	IDESTADO		A	2		

Fig 9.2 Tabla actions

9.6.2 Articulos

Contiene la información de todos los artículos que tiene la empresa.

articulos

Field Name	Field Type	AllowNull	Key	Default	Extra
IDARTICULO	int(5)	NO	PRI		auto_increment
ARTICULO	varchar(50)	NO			
IDFAMILIA	int(5)	NO	MUL		
IDIVA	int(2)	NO	MUL		
OBSERVACIONES	varchar(1000)	YES			
STOCK	decimal(5,2)	NO		0.00	
STOCK_PEDIDO	decimal(5,2)	NO		0.00	
PRECIO_VENTA	decimal(7,2)	NO			
IDESTADO	int(2)	NO	MUL	0	
F_ALTA	timestamp	NO		CURRENT_TIMESTAMP	
F_BAJA	timestamp	YES			

Key Name	Field Name	NonUnique	Collation	Cardinality	Subpart	Packed
PRIMARY	IDARTICULO		A	158		
IDFAMILIA	IDFAMILIA		A	6		
IDIVA	IDIVA		A	8		
IDESTADO	IDESTADO		A	2		

Fig 9.3 Tabla articulos

9.6.3 Articulos_Detalle

Contiene la información detallada de cada artículo.

articulos_detalle

Field Name	Field Type	AllowNull	Key	Default	Extra
IDDETALLE	int(10)	NO	PRI		auto_increment
IDARTICULO	int(5)	NO	MUL		
IDPEDIDO	char(8)	NO	MUL		
CANTIDAD	decimal(7,2)	NO			
PRECIO_COMPRA	decimal(7,2)	NO			
IDESTADO	int(2)	NO	MUL	1	
FECHA	timestamp	NO		CURRENT_TIMESTAMP	

Key Name	Field Name	NonUnique	Collation	Cardinality	Subpart	Packed
PRIMARY	IDDETALLE		A	53		
IDARTICULO	IDARTICULO		A	10		
IDESTADO	IDESTADO		A	4		
IDPEDIDO	IDPEDIDO		A	53		

Fig 9.4 Tabla articulos_detalle

9.6.4 Autorizaciones

Contiene la información de la relación entre actions y empleados.

autorizaciones

Field Name	Field Type	AllowNull	Key	Default	Extra
EMPLEADO	varchar(11)	NO	MUL		
IDACTION	int(5)	NO	MUL		

Key Name	Field Name	NonUnique	Collation	Cardinality	Subpart	Packed
EMPLEADO	EMPLEADO		A	2		
IDACTION	IDACTION		A	2		

Fig 9.5 Tabla autorizaciones

9.6.5 Cargos

Contiene la información detallada de cada cargo.

cargos

Field Name	Field Type	AllowNull	Key	Default	Extra
IDCARGO	int(5)	NO	PRI		auto_increment
CARGO	varchar(50)	NO			
IDESTADO	int(2)	NO	MUL	0	
F_ALTA	timestamp	NO		CURRENT_TIMESTAMP	
F_BAJA	timestamp	YES			

Key Name	Field Name	NonUnique	Collation	Cardinality	Subpart	Packed
PRIMARY	IDCARGO		A	2		
IDESTADO	IDESTADO		A	2		

Fig 9.6 Tabla cargos

9.6.6 Clientes

Contiene la información de un cliente.

clientes

Field Name	Field Type	AllowNull	Key	Default	Extra
NIF	char(11)	NO	PRI		
NOMBRE	varchar(50)	NO			
APELLIDO1	varchar(50)	NO			
APELLIDO2	varchar(50)	YES			
DIRECCION	varchar(100)	YES			
CP	char(5)	YES	MUL		
TEL_FIJO	char(13)	YES			
TEL_MOVIL	char(13)	NO			
EMAIL	varchar(50)	YES			
IDESTADO	int(2)	NO	MUL	0	
F_ALTA	timestamp	NO		CURRENT_TIMESTAMP	
F_BAJA	timestamp	YES			

Key Name	Field Name	NonUnique	Collation	Cardinality	Subpart	Packed
PRIMARY	NIF		A	2		
IDESTADO	IDESTADO		A	2		
CP	CP		A	2		

Fig 9.7 Tabla clientes

9.6.7 Datos_Empresa

Contiene la información de la empresa.

datos_empresa

Field Name	Field Type	AllowNull	Key	Default	Extra
NIF	char(11)	NO	PRI		
NOMBRE	varchar(50)	NO			
DIRECCION	varchar(100)	NO			
CP	char(5)	NO	MUL		
TEL_FIJO	char(13)	YES			
TEL_MOVIL	char(13)	YES			
TEL_FAX	char(13)	YES			
EMAIL	varchar(50)	NO			
ENTIDAD	char(4)	YES			
OFICINA	char(4)	YES			
DC	char(2)	YES			
CUENTA	char(10)	YES			

Key Name	Field Name	NonUnique	Collation	Cardinality	Subpart	Packed
PRIMARY	NIF		A	1		
CP	CP		A	1		

Fig 9.8 Tabla datos_empresa

9.6.8 Empleados

Contiene la información de un empleado de la empresa, que a su vez puede ser un usuario de la aplicación.

empleados

Field Name	Field Type	AllowNull	Key	Default	Extra
NIF	char(11)	NO	PRI		
NOMBRE	varchar(50)	NO			
APELLIDO1	varchar(50)	NO			
APELLIDO2	varchar(50)	YES			
DIRECCION	varchar(100)	YES			
CP	char(5)	YES	MUL		
TEL_FIJO	char(13)	YES			
TEL_MOVIL	char(13)	NO			
IDCARGO	int(5)	NO	MUL		
USUARIO	varchar(8)	NO	MUL		
PASSWORD	varchar(8)	NO			
IDESTADO	int(2)	NO	MUL	0	
F_ALTA	timestamp	NO		CURRENT_TIMESTAMP	
F_BAJA	timestamp	YES			
idempleado	int(11)	YES			

Key Name	Field Name	NonUnique	Collation	Cardinality	Subpart	Packed
PRIMARY	NIF		A	2		
IDCARGO	IDCARGO		A	2		
USUARIO	USUARIO		A	2		
IDESTADO	IDESTADO		A	2		
CP	CP		A	2		

Fig 9.9 Tabla empleados

9.6.9 Facturas_Clientes

Contiene la información de todas las facturas realizadas a un cliente.

facturas_clientes

Field Name	Field Type	AllowNull	Key	Default	Extra
IDFACTURA	char(8)	NO	PRI		
IDPRESUPUESTO	char(8)	NO	MUL		
IDPAGO	int(2)	NO	MUL		
IDENVIO	int(2)	NO	MUL		
OBSERVACIONES	varchar(1000)	YES			
IDESTADO	int(2)	NO	MUL	0	
F_ALTA	timestamp	NO		CURRENT_TIMESTAMP	
F_MODIF	timestamp	YES			
F_BAJA	timestamp	YES			

Key Name	Field Name	NonUnique	Collation	Cardinality	Subpart	Packed
PRIMARY	IDFACTURA		A	2		
IDPAGO	IDPAGO		A	2		
IDENVIO	IDENVIO		A	2		
IDPRESUPUESTO	IDPRESUPUESTO		A	2		
IDESTADO	IDESTADO		A	2		

Fig 9.10 Tabla facturas_clientes

9.6.10 Facturas_Proveedores

Contiene la información de las facturas de un proveedor.

facturas_proveedores

Field Name	Field Type	AllowNull	Key	Default	Extra
IDFACTURA	char(8)	NO	PRI		
FRAPROVEEDOR	varchar(50)	NO			
IDPEDIDO	char(8)	NO	MUL		
IDPAGO	int(2)	NO	MUL		
IDENVIO	int(2)	NO	MUL		
OBSERVACIONES	varchar(1000)	YES			
IDESTADO	int(2)	NO	MUL	0	
F_ALTA	timestamp	NO		CURRENT_TIMESTAMP	
F_MODIF	timestamp	YES			
F_BAJA	timestamp	YES			

Key Name	Field Name	NonUnique	Collation	Cardinality	Subpart	Packed
PRIMARY	IDFACTURA		A	2		
IDPEDIDO	IDPEDIDO		A	2		
IDPAGO	IDPAGO		A	2		
IDENVIO	IDENVIO		A	2		
IDESTADO	IDESTADO		A	2		

Fig 9.11 Tabla facturas_proveedores

9.6.11 Familias

Contiene la información de una família de artículos.

familias

Field Name	Field Type	AllowNull	Key	Default	Extra
IDFAMILIA	int(5)	NO	PRI		auto_increment
FAMILIA	varchar(100)	NO			
IDESTADO	int(2)	NO	MUL	0	
F_ALTA	timestamp	NO		CURRENT_TIMESTAMP	
F_BAJA	timestamp	YES			

Key Name	Field Name	NonUnique	Collation	Cardinality	Subpart	Packed
PRIMARY	IDFAMILIA		A	2		
IDESTADO	IDESTADO		A	2		

Fig 9.12 Tabla familias

9.6.12 Formas_envio

Contiene la información de los diferentes tipos de envío.

formas_envios

Field Name	Field Type	AllowNull	Key	Default	Extra
IDENVIO	int(2)	NO	PRI		auto_increment
ENVIO	varchar(50)	NO			
IDESTADO	int(2)	NO	MUL	0	
F_ALTA	timestamp	NO		CURRENT_TIMESTAMP	
F_BAJA	timestamp	YES			

Key Name	Field Name	NonUnique	Collation	Cardinality	Subpart	Packed
PRIMARY	IDENVIO		A	2		
IDESTADO	IDESTADO		A	2		

Fig 9.13 Tabla formas_envio

9.6.13 Formas_pago

Contiene la información de los diferentes tipos de pago.

formas_pagos

Field Name	Field Type	AllowNull	Key	Default	Extra
IDPAGO	int(2)	NO	PRI		auto_increment
PAGO	varchar(50)	NO			
IDESTADO	int(2)	NO	MUL	0	
F_ALTA	timestamp	NO		CURRENT_TIMESTAMP	
F_BAJA	timestamp	YES			

Key Name	Field Name	NonUnique	Collation	Cardinality	Subpart	Packed
PRIMARY	IDPAGO		A	2		
IDESTADO	IDESTADO		A	2		

Fig 9.14 Tabla formas_pago

9.6.14 Paises

Contiene la información de los diferentes países.

paises

Field Name	Field Type	AllowNull	Key	Default	Extra
IDPAIS	char(3)	NO	PRI		
PAIS	varchar(50)	NO			
IDESTADO	int(2)	NO	MUL	0	
F_ALTA	timestamp	NO		CURRENT_TIMESTAMP	
F_BAJA	timestamp	YES			

Key Name	Field Name	NonUnique	Collation	Cardinality	Subpart	Packed
PRIMARY	IDPAIS		A	2		
IDESTADO	IDESTADO		A	2		
IDPAIS	IDPAIS		A	2		

Fig 9.15 Tabla paises

9.6.15 Pedidos_Proveedores

Contiene la información de todos los pedidos de un proveedor.

pedidos_proveedores

Field Name	Field Type	AllowNull	Key	Default	Extra
IDPEDIDO	char(8)	NO	PRI		
IDPROVEEDOR	char(11)	NO	MUL		
OBSERVACIONES	varchar(1000)	YES			
IDESTADO	int(2)	NO	MUL	0	
F_ALTA	timestamp	NO		CURRENT_TIMESTAMP	
F_MODIF	timestamp	YES			
F_BAJA	timestamp	YES			

Key Name	Field Name	NonUnique	Collation	Cardinality	Subpart	Packed
PRIMARY	IDPEDIDO		A	37		
IDPROVEEDOR	IDPROVEEDOR		A	6		
IDESTADO	IDESTADO		A	6		

Fig 9.16 Tabla pedidos_proveedores

9.6.16 Pedidos_Proveedores_Detalle

Contiene la información detallada de cada pedido.

pedidos_proveedores_detalle

Field Name	Field Type	AllowNull	Key	Default	Extra
IDDETALLE	int(10)	NO	PRI		auto_increment
IDPEDIDO	char(8)	NO	MUL		
IDARTICULO	int(5)	NO	MUL		
CANTIDAD	decimal(7,2)	NO			
IDIVA	int(2)	NO	MUL		
PRECIO	decimal(7,2)	NO			

Key Name	Field Name	NonUnique	Collation	Cardinality	Subpart	Packed
PRIMARY	IDDETALLE		A	55		
IDPEDIDO	IDPEDIDO		A	55		
IDARTICULO	IDARTICULO		A	11		
IDIVA	IDIVA		A	9		

Fig 9.17 Tabla pedidos_proveedores_detalle

9.6.17 Poblaciones

Contiene la información de las diferentes poblaciones.

poblaciones

Field Name	Field Type	AllowNull	Key	Default	Extra
CP	char(5)	NO	PRI		
POBLACION	varchar(60)	NO			
IDPROVINCIA	int(10)	NO	MUL		
IDESTADO	int(2)	NO	MUL		
F_ALTA	timestamp	NO		CURRENT_TIMESTAMP	
F_BAJA	timestamp	YES			

Key Name	Field Name	NonUnique	Collation	Cardinality	Subpart	Packed
PRIMARY	CP		A	2		
IDESTADO	IDESTADO		A	2		
IDPROVINCIA	IDPROVINCIA		A	2		

Fig 9.18 Tabla poblaciones

9.6.18 Presupuestos_Clientes

Contiene la información de un presupuesto realizado a un cliente.

presupuestos_clientes

Field Name	Field Type	AllowNull	Key	Default	Extra
IDPRESUPUESTO	char(8)	NO	PRI		
IDCLIENTE	char(11)	NO	MUL		
OBSERVACIONES	varchar(1000)	NO			
IDESTADO	int(2)	NO	MUL	0	
F_ALTA	timestamp	NO		CURRENT_TIMESTAMP	
F_MODIF	timestamp	YES			
F_BAJA	timestamp	YES			

Key Name	Field Name	NonUnique	Collation	Cardinality	Subpart	Packed
PRIMARY	IDPRESUPUESTO		A	35		
IDESTADO	IDESTADO		A	7		
IDCLIENTE	IDCLIENTE		A	2		

Fig 9.19 Tabla presupuestos_clientes

9.6.19 Presupuestos_Clientes_Detalle

Contiene la información de cada artículo que se incluya en un presupuesto de un cliente.

presupuestos_clientes_detalle

Field Name	Field Type	AllowNull	Key	Default	Extra
IDDETALLE	int(10)	NO	PRI		auto_increment
IDPRESUPUESTO	char(8)	NO	MUL		
IDARTICULO	int(5)	YES	MUL		
IDIVA	int(2)	NO	MUL		
CANTIDAD	decimal(7,2)	NO			
PRECIO	decimal(7,2)	NO			

Key Name	Field Name	NonUnique	Collation	Cardinality	Subpart	Packed
PRIMARY	IDDETALLE		A	40		
IDARTICULO	IDARTICULO		A	40		
IDPRESUPUESTO	IDPRESUPUESTO		A	40		
IDIVA	IDIVA		A	10		

Fig 9.20 Tabla presupuestos_clientes_detalle

9.6.20 Proveedores

Contiene la información de los proveedores.

proveedores

Field Name	Field Type	AllowNull	Key	Default	Extra
NIF	char(11)	NO	PRI		
PROVEEDOR	varchar(50)	NO			
DIRECCION	varchar(100)	YES			
CP	char(5)	YES	MUL		
TEL_FIJO	char(13)	NO			
TEL_MOVIL	char(13)	YES			
TEL_FAX	char(13)	YES			
EMAIL	varchar(50)	YES			
ENTIDAD	char(4)	YES			
OFICINA	char(4)	YES			
DC	char(2)	YES			
CUENTA	char(10)	YES			
CONTACTO	varchar(50)	YES			
WEB	varchar(50)	YES			
IDESTADO	int(2)	NO	MUL	0	
F_ALTA	timestamp	NO		CURRENT_TIMESTAMP	
F_BAJA	timestamp	YES			

Key Name	Field Name	NonUnique	Collation	Cardinality	Subpart	Packed
PRIMARY	NIF		A	2		
CP	CP		A	2		
IDESTADO	IDESTADO		A	2		

Fig 9.21 Tabla proveedores

9.6.21 Provincias

Contiene la información de las provincias.

provincias

Field Name	Field Type	AllowNull	Key	Default	Extra
IDPROVINCIA	int(10)	NO	PRI		auto_increment
PROVINCIA	varchar(50)	NO			
IDPAIS	char(3)	NO	MUL		
IDESTADO	int(2)	NO	MUL	0	
F_ALTA	timestamp	NO		CURRENT_TIMESTAMP	
F_BAJA	timestamp	YES			

Key Name	Field Name	NonUnique	Collation	Cardinality	Subpart	Packed
PRIMARY	IDPROVINCIA		A	2		
IDPAIS	IDPAIS		A	2		
IDESTADO	IDESTADO		A	2		

Fig 9.22 Tabla provincias

9.6.22 Tipos_estados

Contiene la información de los diferentes tipos de estado.

tipos_estados

Field Name	Field Type	AllowNull	Key	Default	Extra
IDESTADO	int(2)	NO	PRI		
ESTADO	varchar(20)	NO			

Key Name	Field Name	NonUnique	Collation	Cardinality	Subpart	Packed
PRIMARY	IDESTADO		A	4		

Fig 9.23 Tabla tipos_estado

9.6.23 Tipos_IVA

Contiene la información de los diferentes tipos de IVA.

tipos_iva

Field Name	Field Type	AllowNull	Key	Default	Extra
IDIVA	int(2)	NO	PRI		auto_increment
DESCRIPCION	varchar(50)	NO			
IVA	decimal(5,2)	NO			
IDESTADO	int(2)	NO	MUL	0	
F_ALTA	timestamp	NO		CURRENT_TIMESTAMP	
F_BAJA	timestamp	YES			

Key Name	Field Name	NonUnique	Collation	Cardinality	Subpart	Packed
PRIMARY	IDIVA		A	2		
IDESTADO	IDESTADO		A	2		

Fig 9.24 Tabla tipos_iva