TRABAJO FINAL DE CARRERA - J2EE

Aplicación Web para la gestión de facturación de una empresa de cerrajería

Sara Gutiérrez Melero
ITIG

Jose Juan Rodriguez

Palafolls, a 18 de junio de 2012

Agradecimientos

Me gustaría dedicar este proyecto a muchas personas por la paciencia, ayuda o tiempo que no les he podido prestar. En especial quiero dedicárselo a mi novio, Toni ya que sin el esto no hubiera sido posible.
A los demás, mis padres, mi hermana, mis sobrinos, mis suegros, gracias por haberme ayudado a hacer esto posible y apoyado en todo momento.

Resumen

Este es el documento de memoria para el trabajo final de carrera (TCF) de Ingeniería Técnica en Informática de Gestión. En el encontraremos diversas secciones donde se detalla desde la fase de planificación hasta la fase de implementación y puesta en marcha.

El presente TFC centra su esfuerzo en la creación de un producto y se basa en el análisis, diseño e implementación de una aplicación web para una pequeña empresa de cerrajería que desea llevar a cabo una correcta gestión de su facturación así como el mantenimiento de datos de sus clientes.

El área en la que se encuadra este TFC es J2EE y por lo tanto ésta ha sido la plataforma bajo la que he desarrollado la implementación de la aplicación web. Gracias a esta plataforma se ha podido utilizar diferentes tecnologías e implementar un marco de trabajo dividido en capas.
El lenguaje de programación utilizado ha sido Java utilizando Eclipse Indigo como entorno de desarrollo y Javascript para la parte cliente de la aplicación.

Se ha procurado que la aplicación tenga un nivel adecuado de seguridad, permitiendo únicamente a los usuarios autenticados realizar la gestión de facturación de la empresa. Del mismo modo, garantizar el acceso exclusivo de los administradores acreditados a la gestión referente a los datos de la empresa, a través de las cuales poder realizar las altas, bajas y modificaciones de los usuarios del sistema, así como consultar y modificar los datos relativos a la empresa.

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 permite seguir la evolución de la construcción del sistema, así como familiarizarse con distintas herramientas usadas en el desarrollo de aplicaciones web como son Java, Struts2, Spring, Hibernate, iText, JavaScript o jQuey.

Palabras clave: TFC, Java, J2EE, marco de trabajo, Struts 2, Eclipse, Spring, Hibernate, iText, JavaScript, jQuery.

Índice de contenidos

Índice de figuras y tablas

Capítulo 1. Introducción

1.1 Justificación del TFC y contexto en el cual se desarrolla: punto de partida y aportación del TFC

Las aplicaciones web son herramientas que han cogido mucho protagonismo en los últimos años. Es un escenario mundial cada vez más global y con empresas cada vez menos centralizadas, la necesidad de 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 proporcionando un escenario simple.

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 y gestionar su facturación.

Este TFC se ha realizado en base a la necesidad de crear una aplicación web que permita la consulta y gestión por Internet de todo lo relacionado con la facturación de la empresa (clientes, obras, facturas, presupuestos, etc.) por parte de los usuarios (empleados y administradores), empleando las tecnologías oportunas para conseguir que esta aplicación tenga además un nivel aceptable de seguridad y un tiempo de respuesta lo más bajo posible en las interacciones de los usuarios con la aplicación. Todo esto haciendo uso de la plataforma J2EE.

A nivel personal, la realización del presente TFC me ha permitido por un lado, poder agrupar en un solo proyecto los conocimientos adquiridos a lo largo de la Ingeniería en diferentes asignaturas, como Programación orientada a objetos, Bases de Datos, Ingeniería del Software o Técnicas de desarrollo de software y por otro, la oportunidad de poder introducirme en una arquitectura totalmente desconocida para mí, como es J2EE.

Este proyecto me ha brindado también la oportunidad de adentrarme en Frameworks y tecnologías como Struts 2, Spring, Hibernate, iText, Tiles, JSP y jQuery, e integrarlas en el proyecto implementando sus mejores funciones. Pero principalmente, lo más significativo para mí, ha sido el poder desarrollar un producto completo, desde la toma de requisitos hasta la implementación y pruebas pertinentes, pasando por las fases de análisis y diseño.

El punto de partida no ha sido precisamente el idóneo, puesto que aprender nuevas tecnologías implica dedicar tiempo a obtener una serie de conocimientos teniendo en cuenta que en el presente TFC estábamos limitados en cuanto a tiempo se refiere. Al tener que adquirir esos conocimientos, realizar pruebas con las diferentes tecnologías, resolver errores que han ido surgiendo a lo largo de la implementación y realizar en sí el TFC me ha llevado a consumir un considerable número de horas de dedicación. No obstante, tener que resolver individualmente todos los errores e investigar sobre alternativas me ha llevado a un aprendizaje aún más profundo y del cual puedo decir que saco provecho absoluto.

En conclusión, la aportación del TFC es por una parte una solución a las necesidades de una empresa de disponer de una aplicación web orientada a la gestión de facturación de la misma, de la cual se espera un nivel aceptable de seguridad y unos buenos tiempos de respuesta a las peticiones realizadas por los usuarios y por otra, de carácter personal, la posibilidad de desarrollar un producto completo y el aprendizaje de nuevas tecnologías que me han llegado a motivar hasta el punto de decidirme a realizar, seguidamente a la finalización de la ITIG, el máster universitario en ingeniería del software para la web.

1.2 Objetivos del TFC

El objetivo principal de este TFC es profundizar en el uso de la tecnología Java y adquirir conocimientos sobre la arquitectura J2EE mediante el desarrollo del proyecto GestDifran. Con este fin, se trabaja en las siguientes áreas:

- Análisis, diseño e implementación de una aplicación similar a las existentes en el mundo empresarial utilizando la orientación a objetos.
- Patrones de diseño aplicables a la arquitectura J2EE.
- Comunicación con bases de datos relacionales.
- Tecnología J2EE: Java Server Pages (JSP), Servlets, etc.
- Frameworks: Struts 2, Spring, Hibernate, etc.

Por otra parte, al pretender ser este un trabajo de síntesis de los conocimientos adquiridos en la carrera, se siguen las metodologías de desarrollo y programación ya estudiadas en otras asignaturas. Al final del TFC la aplicación debe ser completamente funcional.

Crear una aplicación que permita el acceso de los usuarios, previamente autentificados, a la creación y actualización de datos de facturación de la empresa.

Que dicha aplicación permita a uno o más administradores introducir, modificar o deshabilitar a usuarios del sistema, así como actualizar los datos relativos a la misma empresa.

1.3 Enfoque y método a seguir

Para el desarrollo de la aplicación se ha intentado seguir el enfoque de una empresa de desarrollo de software. Esto implica que no sólo es importante el resultado final, sino que éste debe haberse acotado previamente de manera clara, debe generarse la documentación adecuada a cada fase y deben alcanzarse la consecución de los objetivos en los plazos establecidos.

Para ello, se ha seguido un ciclo de vida en cascada clásico en el que se han previsto y realizado las siguientes etapas:

- Definición funcional
- Planificación del proyecto
- Análisis
- Diseño
- Implementación
- Pruebas
- Documentación

En paralelo, desde el comienzo del TFC se ha avanzado en el estudio de la arquitectura J2EE y del software necesario para su ejecución. En primer lugar, se ha realizado una breve investigación de las opciones disponibles. Una vez vistas las opciones más adecuadas al proyecto se han realizado algunas pruebas de implementación para evaluarlas. Finalmente, definidas las principales tecnologías a utilizar, en concreto Struts 2, JSP, Spring e Hibernate, se ha buscado material bibliográfico que permitiese la formación en dichas tecnologías y sirviese de referencia durante la implementación del proyecto.

Por lo tanto, el enfoque que se le ha dado al presente TFC ha sido planificar y organizar el trabajo a realizar distribuyéndolo en el tiempo. Se ha intentado llevar estrictamente la planificación inicial del proyecto dado que se disponía de un tiempo limitado para la total realización y presentación del mismo.

Realizar el análisis con el fin de obtener la funcionalidad que debía ofrecer la aplicación en base a las necesidades de los usuarios, obteniendo los diagramas de caso de uso de cada subsistema, así como sus especificaciones textuales.

Posteriormente, se ha realizado el diseño técnico definiendo la arquitectura de la aplicación, obteniendo los diagramas de clases, diseño de la persistencia y diseño de la interfaz de usuario.

Finalizada la etapa de análisis y diseño, se ha llevado a cabo la implementación codificando los subsistemas existentes, realizando posteriormente pruebas para verificar y depurar posibles fallos.

Por último, se ha redactado la presente memoria y creado una presentación virtual con LibreOffice Impress.

1.4 Planificación del proyecto

FASE	COMIENZO	FIN
APRENDIDAJE DE LA TECNOLOGIA	**29/02/12**	**04/06/12**
Estudios previos	29/02/12	04/06/12
Instalación y configuración del software necesario	24/03/12	26/03/12
Pruebas iniciales	26/03/12	30/03/12
PLANIFICACIÓN	**29/02/12**	**14/03/12**
Descripción y objetivos del TFC	29/02/12	06/03/12
Organización del tiempo y descomposición de tareas	07/03/12	09/03/12
Elaboración del calendario	10/03/12	11/03/12
Revisión y correcciones oportunas	12/03/12	14/03/12
Entrega PAC1	14/03/12	14/03/12
ANÁLISIS FUNCIONAL	**15/03/12**	**31/03/12**
Definición de requerimientos funcionales y no funcionales	15/03/12	16/03/12
Definición de las funcionalidades de cada subsistema	17/03/12	18/03/12
Especificación textual de los casos de uso	19/03/12	22/03/12
Definición de actores	23/03/12	24/03/12
Extensibilidad prevista	25/03/12	26/03/12
Revisión y correcciones oportunas	27/03/12	31/03/12
DISEÑO TÉCNICO	**01/04/12**	**18/04/12**
Descripción y diseño de la arquitectura a implementar	01/04/12	06/04/12
Definición de Frameworks a utilizar	07/04/12	12/04/12
Diagrama de clases	13/04/12	13/04/12
Script para la creación de la BD	14/04/12	15/04/12

Revisión y correcciones oportunas	16/04/12	18/04/12
Entrega PAC2	18/04/12	18/04/12
IMPLEMENTACIÓN	**19/04/12**	**04/06/12**
Creación del proyecto en Eclipse e inclusión de las librerías necesarias	19/04/12	22/04/12
Configuración de Struts2	23/04/12	25/04/12
Integración y configuración de los Frameworks Spring e Hibernate	26/04/12	29/04/12
Creación de la Base de Datos	30/04/12	30/04/12
Creación de POJOs y mapeos correspondientes	01/05/12	04/05/12
Creación e implementación de los DAOs	05/05/12	07/05/12
Diseño de las interfaces	08/05/12	02/06/12
Creación e implementación de las acciones	08/05/12	02/06/12
Revisión y correcciones oportunas	03/06/12	04/06/12
Entrega PAC3	04/06/12	04/06/12
MEMÓRIA Y PRESENTACIÓN VIRTUAL	**05/06/12**	**18/06/12**
Estructuración y redactado de la memoria	05/06/12	11/06/12
Elaboración de la presentación virtual	12/06/12	16/06/12
Revisión y correcciones oportunas	17/06/12	18/06/12
Entrega PAC4	18/06/12	18/06/12

Tabla 1. Plan de trabajo

1.5 Productos obtenidos

Como resultado final, se han obtenido los siguientes productos:

sgutierrezmel_memoria.pdf, que es la presente memoria.
sgutierrezmel_presentacion.zip, que contiene la presentación virtual del TFC en LibreOffice Impress y en pdf.
sgutierrezmel_producto.zip, que es la aplicación web resultante y que contiene el archivo .war para poder importarlo directamente al entorno de desarrollo integrado y dos scripts sql, uno para la creación de la Base de Datos y otro para las inserciones a modo de prueba de la aplicación.

1.6 Breve descripción de los otros capítulos de la memoria

En los próximos capítulos de la memoria, se comentaran las fases de análisis funcional, diseño técnico e implementación, que se han llevado a cabo para desarrollar la aplicación.

Habrá un capítulo donde se detallará la valoración económica del proyecto.

Seguidamente, en otro capítulo se plasmarán las conclusiones personales donde se hará una justificación y razonamiento de las decisiones adoptadas durante el proyecto. Se especificaran qué experiencias o conclusiones he extraído de este TFC.

I, finalmente, otros tres capítulos más dedicados a lineas futuras de la aplicación, glosario de la memoria y bibliografía consultada.

Capítulo 2. Análisis funcional

En este capítulo se realiza un análisis teórico del sistema desarrollado. Se hace una primera descripción del funcionamiento del sistema y a continuación se detallan tanto los requisitos funcionales del sistema como los no funcionales.

2.1 Descripción del funcionamiento del sistema

La aplicación que se ha planteado construir en este proyecto pretende cubrir la necesidad del control de gestión de facturación para la empresa *Serralleria Difran*.
La aplicación debe permitir gestionar la facturación de productos, según sean los trabajos realizados, a los clientes así como el mantenimiento de los mismos.
Se ha tenido en cuenta el rol del usuario para controlar qué acciones se puede hacer sobre el sistema. Un empleado no debe tener acceso a la gestión de usuarios de la aplicación ni al mantenimiento de datos personales de la empresa. Por eso, se ha implantado un sistema de control de usuarios para la aplicación.
Por último, también se ha controlado la seguridad en el acceso a la aplicación mediante autenticación de usuarios. De esta manera, ingresando una URL de cualquier página de acción de la aplicación ésta redireccionará a la pantalla de acceso si el usuario no ha sido autentificado previamente.

Hecha esta breve descripción del sistema se analizará a continuación 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 qué 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.

2.2. Análisis de requisitos funcionales

La aplicación está compuesta por diferentes subsistemas. Para analizar los requisitos funcionales se detallará cada uno de estos subsistemas mediante la explicación de lo que ofrecen, acompañados de sus casos de uso y las especificaciones de estos últimos.

- Gestión de Empresa
- Gestión de Clientes
- Gestión de Obras
- Conexión

2.2.1 Gestión de empresa

La gestión de empresa se centra en los administradores de ésta que son los usuarios potenciales de la aplicación.
Mediante una interfaz amigable se podrá consultar y modificar los datos de empresa. Existirá

también una relación de usuarios con la cual se podrán crear, buscar (por NIF), consultar, modificar, activar e inactivar. Si un usuario se inactiva perderá todos los permisos de la aplicación.
En el apartado Actores se podrá ver más en detalle.

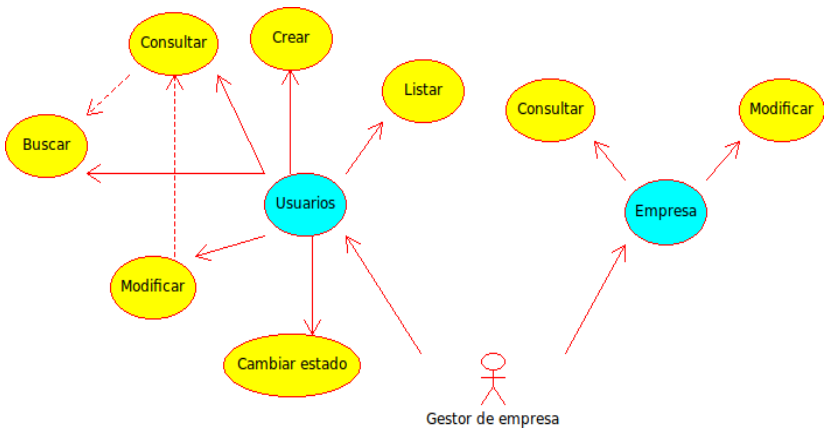

Figura 1. Diagrama de casos de uso Gestión de Empresa

2.2.2 Gestión de clientes

Este es el subsistema principal de la aplicación y por ello es también imprescindible.
Mediante una interfaz amigable se podrá crear, consultar, buscar (por NIF), modificar, activar e inactivar clientes. Al mismo tiempo se permitirá visualizar las facturas o presupuestos asociados al cliente. Estos también se podrán crear, consultar, modificar, cambiar estado y generar en PDF. De cada presupuesto se podrá generar automáticamente una factura relativa a los trabajos realizados a un cliente, la cual irá asociada a este presupuesto.
Al dar de alta una nueva factura o presupuesto se podrá o bien seleccionar una obra existente o darla de alta en ese momento.

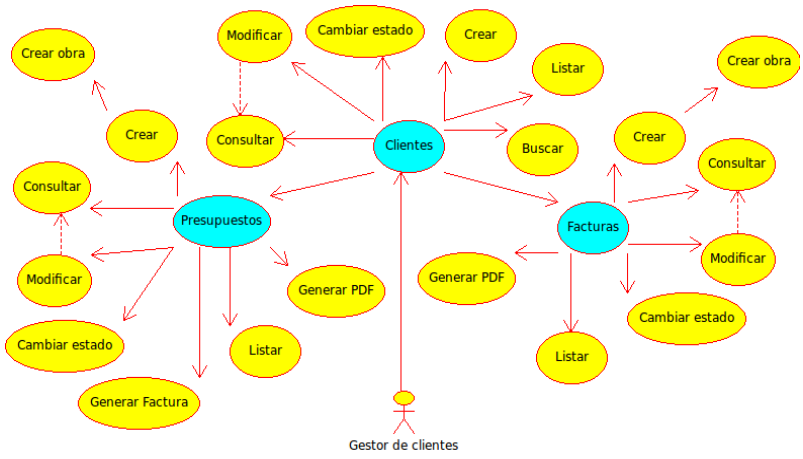

Figura 2. Diagrama de casos de uso Gestión de Clientes

2.2.3 Gestión de obras

El sistema debe permitir la gestión de las obras de los clientes para las cuales han contratado los servicios de la empresa Serralleria Difran.
Mediante una interfaz amigable se podrá crear, consultar, modificar, activar e inactivar cualquier obra.
A la hora de crear una nueva obra en el sistema se podrá o bien seleccionar un cliente ya existente en la empresa o bien darlo de alta en el mismo momento (reenviando a la vista de

Nuevo Cliente).

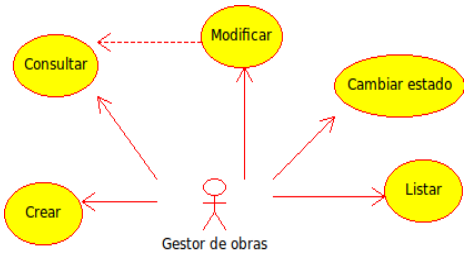

Figura 3. Diagrama de casos de uso Gestión de Obras

2.2.4 Conexión

Finalmente, el sistema también debe permitir gestionar las conexiones. Esto se entiende como el acceso a la aplicación mediante validación de datos con un nombre de usuario y contraseña, y al cierre de sesión de un usuario.
Antes de iniciar la aplicación se deberá introducir correctamente los datos de acceso mediante una interfaz de bienvenida. A este proceso se le llama autenticación de usuarios. Mediante esta interfaz, y solamente antes de haber iniciado sesión, también se tendrá la opción de cambiar la contraseña actual.
Una vez se acceda a la aplicación se dispondrá de una opción para salir de la sesión iniciada y volver a la interfaz de bienvenida y acceso a la aplicación de GestDifran.

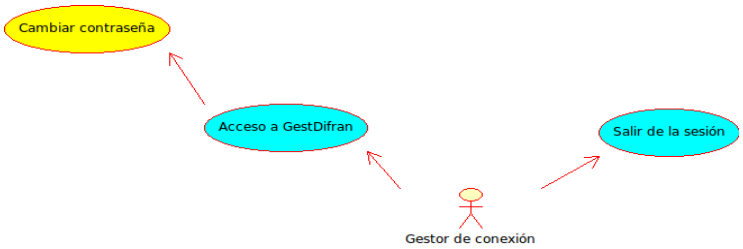

Figura 4. Diagrama de casos de uso Conexión

2.3 Actores

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 son diferenciados en dos grupos: administradores y empleados.

Hay un usuario excepcional. Para poder realizar tareas de mantenimiento de forma sencilla se establece que cualquier usuario de tipo administrador no tendrá limitaciones a nivel de autorizaciones.

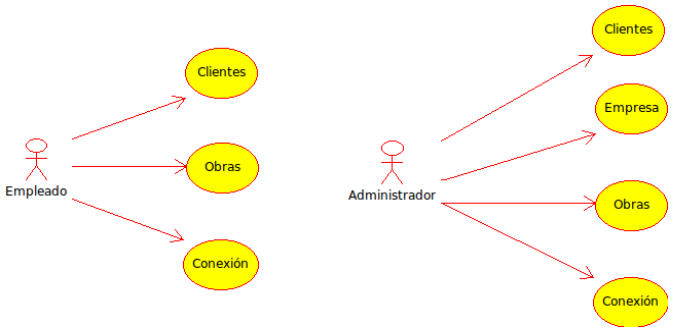

Figura 5. Diagrama de casos de uso de actores en GestDifran

2.4 Especificación de los casos de uso

Caso de uso núm. 1.1:	"Empresa"
Resumen de la funcionalidad:	Visualiza los datos almacenados correspondientes a la empresa.
Papel del usuario:	Forma parte de la gestión de empresa del usuario.
Actores:	**Gestor de empresa**
Casos de uso relacionados:	Modificar
Precondición:	No hace falta ninguna condición previa a la consulta.
Postcondición:	No se ha de cumplir ninguna condición después de la consulta.
Descripción:	El usuario puede visualizar los datos correspondientes a la empresa.
Alternativas de proceso y excepciones:	
Acciones disponibles:	Se tiene la opción de modificar los datos.
Observaciones:	

Tabla 2. Especificación caso de uso Empresa

Caso de uso núm. 1.2:	"Usuarios"
Resumen de la funcionalidad:	Gestiona las cuentas de usuario del sistema.
Papel del usuario:	Forma parte de la gestión de empresa del usuario.
Actores:	**Gestor de empresa**
Casos de uso relacionados:	Listar, Crear, Buscar
Precondición:	Cierto
Postcondición:	Cierto
Descripción:	El sistema muestra las principales opciones disponibles para la gestión de los usuarios de la empresa.
Alternativas de proceso y excepciones:	
Acciones disponibles:	
Observaciones:	

Tabla 3. Especificación caso de uso Usuarios

Caso de uso núm. 1.3:	"Clientes"
Resumen de la funcionalidad:	Gestiona los clientes de la empresa.
Papel del usuario:	Forma parte de la gestión de clientes del usuario.
Actores:	**Gestor de clientes**
Casos de uso relacionados:	Listar, Crear, Buscar
Precondición:	Cierto
Postcondición:	Cierto
Descripción:	El sistema muestra las principales opciones disponibles para la gestión de los clientes de la empresa.
Alternativas de proceso y excepciones:	
Acciones disponibles:	
Observaciones:	

Tabla 4. Especificación caso de uso Clientes

Caso de uso núm. 1.4:	"Facturas"
Resumen de la funcionalidad:	Muestra un listado de todas las facturas realizadas a un cliente de la empresa.
Papel del usuario:	El gestor desea gestionar las facturas de clientes de la empresa.
Actores:	**Gestor de clientes**
Casos de uso relacionados:	Crear, Consultar, Modificar, Cambiar estado, Generar PDF

Precondición:	Cierto
Postcondición:	Cierto
Descripción:	El sistema muestra todas las acciones posibles para la gestión de las facturas de la empresa.
Alternativas de proceso y excepciones:	
Acciones disponibles:	
Observaciones:	La lista se podrá ordenar a través de sus columnas.

Tabla 5. Especificación caso de uso Facturas

Caso de uso núm. 1.5:	"Presupuestos"
Resumen de la funcionalidad:	Muestra un listado de todos los presupuestos realizados a un cliente de la empresa.
Papel del usuario:	El gestor desea gestionar los presupuestos de clientes de la empresa.
Actores:	**Gestor de clientes**
Casos de uso relacionados:	Crear, Consultar, Modificar, Cambiar estado, Generar factura, Generar PDF
Precondición:	Cierto
Postcondición:	Cierto
Descripción:	El sistema muestra todas las acciones posibles para la gestión de los presupuestos de la empresa.
Alternativas de proceso y excepciones:	
Acciones disponibles:	
Observaciones:	La lista se podrá ordenar a través de sus columnas.

Tabla 6. Especificación caso de uso Presupuestos

Caso de uso núm. 1.6:	"Obras"
Resumen de la funcionalidad:	Gestiona las obras de los clientes de la empresa.
Papel del usuario:	Forma parte de la gestión de obras del usuario.
Actores:	**Gestor de obras**
Casos de uso relacionados:	Listar, Crear
Precondición:	Cierto
Postcondición:	Cierto
Descripción:	El sistema muestra las principales opciones disponibles para la gestión de los clientes de la empresa.
Alternativas de proceso y excepciones:	
Acciones disponibles:	
Observaciones:	

Tabla 7. Especificación caso de uso Obras

Caso de uso núm. 1.7:	"Acceso a GestDifran"
Resumen de la funcionalidad:	Controlar los accesos a la aplicación de forma que únicamente puedan acceder unos usuarios determinados.
Papel del usuario:	El gestor accede al sistema y puede utilizar la aplicación.
Actores:	**Gestor de conexión**
Casos de uso relacionados:	Cambiar contraseña
Precondición:	El usuario es válido si está dado de alta en el sistema.
Postcondición:	El sistema valida el usuario e inicia la aplicación.
Descripción:	El sistema pide el nombre de usuario (*login*) al usuario y su contraseña (*password*), valida los datos y, según proceda, inicia o no la aplicación con sus perfil asociado.
Alternativas de proceso y excepciones:	Existe un curso alternativo. Error en el caso que los datos no sean

	los correctos, el sistema devuelve un error de usuario / contraseña.
Acciones disponibles:	
Observaciones:	

Tabla 8. Especificación caso de uso Acceso a GestDifran

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 correctos y a qué parte tiene acceso.
	3. El sistema muestra la pantalla principal.

Tabla 9. Curso tipo del acontecimiento

Caso de uso núm. 1.8:	**“Cambiar contraseña”**
Resumen de la funcionalidad:	Cambiar la contraseña de los usuarios que se encuentren activos en el sistema.
Papel del usuario:	El gestor desea cambiar su contraseña de acceso a la aplicación.
Actores:	**Gestor de conexión**
Casos de uso relacionados:	Acceso a GestDifran
Precondición:	El usuario es válido si está dado de alta en el sistema y conoce su contraseña actual.
Postcondición:	Si el usuario por el cual se pide el cambio de contraseña está activo y se ha identificado correctamente, se hará efectivo el cambio de contraseña.
Descripción:	
Alternativas de proceso y excepciones:	
Acciones disponibles:	
Observaciones:	

Tabla 10. Especificación caso de uso Cambiar contraseña

Caso de uso núm. 1.9:	**“Salir de la sesión”**
Resumen de la funcionalidad:	Dar la sesión del usuario por finalizada y eliminar cualquier rastro de los datos de sesión.
Papel del usuario:	El gestor quiere cerrar la sesión de la aplicación.
Actores:	**Gestor de conexión**
Casos de uso relacionados:	-
Precondición:	El usuario ha entrado en el sistema
Postcondición:	El usuario ha salido del sistema.
Descripción:	Se mostrará un aviso de confirmación al usuario por si está seguro de abandonar el sistema.
Alternativas de proceso y excepciones:	
Acciones disponibles:	
Observaciones:	

Tabla 11. Especificación caso de uso Salir de la sesión

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

Tabla 12. Curso tipo del acontecimiento

Caso de uso núm. 1.10:	**“Listar”**

Resumen de la funcionalidad:	Muestra una lista de conceptos/objetos de la base de datos del sistema.
Papel del usuario:	El gestor quiere listar conceptos/objetos.
Actores:	**Gestor de empresa, Gestor de clientes, Gestor de obras**
Casos de uso relacionados:	Usuarios, Clientes, Facturas, Presupuestos, Obras, Crear
Precondición:	Existe almenos un concepto/objeto en la base de datos.
Postcondición:	La lista se ha creado satisfactoriamente.
Descripción:	El usuario puede visualizar un listado de conceptos/objetos.
Alternativas de proceso y excepciones:	
Acciones disponibles:	
Observaciones:	

Tabla 13. Especificación caso de uso Listar

Caso de uso núm. 1.11:	**“Crear”**
Resumen de la funcionalidad:	Crea un nuevo concepto/objeto en la base de datos del sistema.
Papel del usuario:	El gestor quiere crear un nuevo concepto/objeto.
Actores:	**Gestor de empresa, Gestor de clientes, Gestor de obras**
Casos de uso relacionados:	Usuarios, Clientes, Facturas, Presupuestos, Obras, Listar
Precondición:	El concepto/objeto no existe en el sistema.
Postcondición:	El concepto/objeto se ha creado satisfactoriamente en el sistema.
Descripción:	El usuario puede dar de alta, con validación de la información incluida.
Alternativas de proceso y excepciones:	Existe un curso alternativo. Error en caso que los datos no sean los correctos, la aplicación devuelve un error informando el campo incorrecto.
Acciones disponibles:	En la creación de una factura o presupuesto de cliente se tendrá la opción de seleccionar una obra existente o darla de alta en ese momento. Para la creación de una factura o presupuesto se tendrá también la opción (a parte de la obra) de seleccionar un cliente existente o darlo de alta en ese momento. En la creación de una obra se tendrá la posibilidad de seleccionar un cliente existente o bien de darlo de alta.
Observaciones:	

Tabla 14. Especificación caso de uso Crear

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 solicita 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 correctos.
	5. El sistema hace un INSERT en la base de datos con la información que ha introducido el usuario.

Tabla 15. Curso tipo del acontecimiento

Caso de uso núm. 1.12:	**“Modificar”**
Resumen de la funcionalidad:	Modifica y actualiza los datos almacenados en la base de datos del sistema, previamente validados.
Papel del usuario:	El gestor quiere modificar un concepto/objeto.
Actores:	**Gestor de empresa, Gestor de clientes, Gestor de obras**
Casos de uso relacionados:	Empresa, Usuarios, Clientes, Facturas, Presupuestos, Obras

Precondición:	Los datos están correctamente cumplimentados.
Postcondición:	Los datos se han actualizado.
Descripción:	El usuario edita los datos que desea modificar y los actualiza.
Alternativas de proceso y excepciones:	
Acciones disponibles:	
Observaciones:	

Tabla 16. Especificación caso de uso Modificar

Acciones del Gestor	**Respuesta del Sistema**
1. Este caso de uso empieza cuando un usuario selecciona el botón de “Modificar” de cualquier concepto de la lista.	
	2. El sistema muestra la pantalla con el formulario rellenado con los datos actuales del concepto.
3. El usuario modifica los datos deseados y clicka en el botón “Actualizar”.	
	4. El sistema a partir de los datos modificados por el usuario, hace un UPDATE para actualizar el concepto en la B.D.
	5. El sistema muestra la pantalla anterior a la modificación.

Tabla 17. Curso tipo del acontecimiento

Caso de uso núm. 1.13:	**“Cambiar estado”**
Resumen de la funcionalidad:	Activa o inactiva el estado de un concepto/objeto.
Papel del usuario:	El gestor quiere modificar el estado de cualquier concepto/objeto.
Actores:	**Gestor de empresa, Gestor de clientes, Gestor de obras**
Casos de uso relacionados:	Usuarios, Clientes, Facturas, Presupuestos, Obras
Precondición:	Los datos están correctamente cumplimentados.
Postcondición:	Los datos se han actualizado.
Descripción:	El usuario puede dar de baja cualquier concepto/objeto de cualquier tipo de gestión, para que no pueda ser utilizado en la aplicación.
Alternativas de proceso y excepciones:	
Acciones disponibles:	
Observaciones:	

Tabla 18. Especificación caso de uso Cambiar estado

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 base de datos.
3. El usuario clicka en el botón de “Activar/Inactivar” el concepto.	
	4. El sistema hace un UPDATE en la base de datos cambiando el estado del concepto/objeto.
	5. El sistema vuelve a mostrar la lista de todos los conceptos que hay en la B.D.

Tabla 19. Curso tipo del acontecimiento

Caso de uso núm. 1.14:	**“Buscar”**
Resumen de la funcionalidad:	Busca conceptos/objetos almacenados en la B.D.
Papel del usuario:	El gestor busca información de cualquier concepto/objeto.

Actores:	**Gestor de empresa, Gestor de clientes**
Casos de uso relacionados:	Clientes, Usuarios
Precondición:	Los datos mínimos coinciden con al menos un concepto/objeto de la B.D.
Postcondición:	Se muestra el concepto/objeto coincidente.
Descripción:	El gestor puede buscar cualquier cliente introduciendo el NIF, para que el servidor muestre la pantalla de consulta de ese cliente.
Alternativas de proceso y excepciones:	
Acciones disponibles:	
Observaciones:	

Tabla 20. Especificación caso de uso Buscar

Acciones del Gestor	**Respuesta del Sistema**
1. Este caso de uso empieza cuando un usuario introduce los datos requeridos en el campo de texto y selecciona el botón de “Buscar” de cualquier gestión.	
	4. El sistema a partir de los datos introducidos por el usuario, hace un SELECT para buscar el concepto que cumple con los requisitos.
	5. El sistema muestra la pantalla de consulta de ese concepto en cuestión.

Tabla 21. Curso tipo del acontecimiento

Caso de uso núm. 1.15:	**“Consultar”**
Resumen de la funcionalidad:	Consulta los datos de conceptos/objetos almacenados en la B.D.
Papel del usuario:	El gestor consulta la información de cualquier concepto/objeto.
Actores:	**Gestor de empresa, Gestor de clientes, Gestor de obras**
Casos de uso relacionados:	Empresa, Usuarios, Clientes, Facturas, Presupuestos, Obras
Precondición:	Se la seleccionado un concepto/objeto del listado.
Postcondición:	Se muestran los datos del concepto/objeto seleccionado por el usuario.
Descripción:	El gestor puede consultar cualquier concepto/objeto seleccionándolo previamente en el listado.
Alternativas de proceso y excepciones:	
Acciones disponibles:	
Observaciones:	

Tabla 22. Especificación caso de uso Consultar

Caso de uso núm. 1.16:	**“Generar Factura”**
Resumen de la funcionalidad:	Genera la factura asociada a un presupuesto ya existente.
Papel del usuario:	El gestor quiere crear una factura a partir de un presupuesto.
Actores:	**Gestor clientes**
Casos de uso relacionados:	Presupuestos
Precondición:	Se ha seleccionado un concepto/objeto del listado.
Postcondición:	Se genera la factura del concepto/objeto seleccionado por el usuario.
Descripción:	El gestor puede generar la factura de cualquier concepto/objeto seleccionándolo previamente en el listado.
Alternativas de proceso y excepciones:	
Acciones disponibles:	
Observaciones:	

Tabla 23. Especificación caso de uso Generar factura

Caso de uso núm. 1.17:	“Generar PDF”
Resumen de la funcionalidad:	Exporta a formato PDF un concepto/objeto.
Papel del usuario:	El gestor desea generar en PDF un concepto/objeto seleccionado o consultado.
Actores:	**Gestor de clientes**
Casos de uso relacionados:	Facturas, Presupuestos
Precondición:	Se ha seleccionado el concepto/objeto o se está consultando.
Postcondición:	Se exporta a PDF correctamente.
Descripción:	El gestor puede generar en PDF cualquier concepto/objeto seleccionándolo previamente de la lista o accediendo a su consulta.
Alternativas de proceso y excepciones:	
Acciones disponibles:	
Observaciones:	

Tabla 24. Especificación caso de uso Generar PDF

2.5. Análisis de requisitos no funcionales

Éstos son requisitos que especifican criterios que pueden usarse para juzgar la operación del sistema en lugar de sus comportamientos específicos, es decir, de sus requisitos funcionales. Por tanto, se refieren a todos los requisitos que ni describen información a guardar, ni funciones a realizar.

2.5.1 Calidad

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

- **Integridad y seguridad.** Es necesario asegurar que personas ajenas al programa no puedan acceder a él. Para aquellos usuarios que tengan privilegios de acceso hay que corroborar que sólo tengan cobertura en aquellas partes del aplicativo donde los permisos de su rol tengan vigencia. Además los datos del programa deben 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.

- **Interoperabilidad.** Será posible unir el programa a otros, por ejemplo mediante Web Services, sin grandes cambios en el sistema.

2.5.2 Carga

Se establece como requisito no funcional lograr unos parámetros de carga razonables. Por una parte se decide que pueden conectarse un mínimo de 2 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 y 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 GestDifran.

2.5.3 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 software libre para lograr este propósito.

2.5.4 Requerimientos tecnológicos

Se debe tener en consideración que la aplicación debe ser accesible desde cualquier sistema operativo y navegador web. El sistema usará el protocolo http para conectar los clientes y el servidor. Por tanto, será necesario una máquina donde situar el servidor y un navegador web en aquellos equipos donde se sitúen los clientes. El navegador podrá funcionar con cualquier resolución igual o superior a 800x600 píxeles.

Para lograr el hito de independizar el aplicativo de plataformas o propietarios se hace uso del lenguaje UML (Unified Modeling Languaje) en el diseño y lenguajes como Java, Frameworks como Struts 2, o soluciones como JavaScript, CSS, etc. en la implementación.

2.5.5 Interfaces

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 interfaces amigables y simples.

2.6 Especificación de las pantallas del sistema

En la siguiente imagen podemos observar el diagrama de transición de pantallas simplificado de la aplicación.

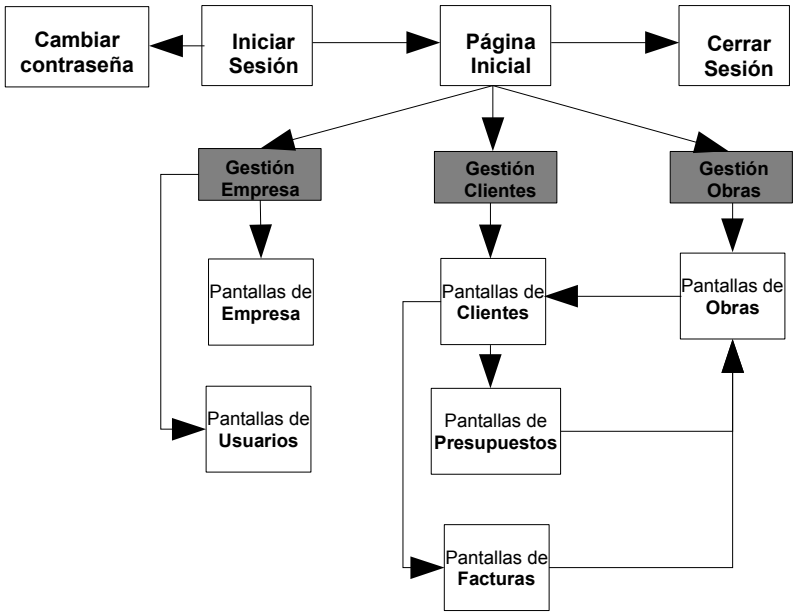

Figura 6. Diagrama de pantallas del sistema

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 realizan 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.
- Cuerpo: Contenido que consultará, modificará el usuario.
- Pie: Imagen para distinguir el fin de página.

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

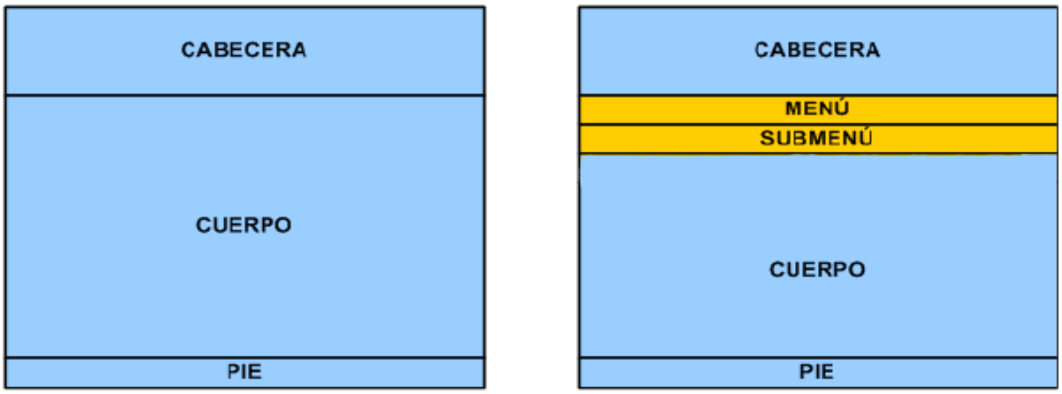

Figura 7. Ejemplos de estructura de pantallas

La sección cuerpo estará compuesta por texto, formularios, listados, imágenes, botones...

Capítulo 3. Diseño técnico

Según lo detallado, se ha procedido a la creación de una aplicación web para facilitar una herramienta de trabajo en la administración y gestión de facturación de la empresa Serralleria Difran. De esta manera, los usuarios del sistema podrán acceder a la aplicación desde cualquier lugar que disponga de conexión a Internet.
Por lo tanto, se ha diseñado e implementado la aplicación GestDifran siguiendo las especificaciones definidas por la plataforma J2EE. Los principales motivos de esta elección para el desarrollo de la aplicación son:

- Abstracción al desarrollador de las tareas de bajo nivel, ya que se encuentran cubiertas por las propias APIs de la plataforma J2EE.
- Existencia de una gran variedad de Frameworks que incrementan las utilidades o tareas de bajo nivel.
- Java como lenguaje de programación. Enfocado claramente a la programación orientada a objetos (POO) que favorece en gran medida la reutilización del código fuente y la portabilidad.
- Fomenta el uso del patrón de diseño MODELO – VISTA – CONTROLADOR (MVC), el cual permite separar la parte lógica de la presentación en una aplicación web.

Una vez recogidas las especificaciones es necesario definir el escenario bajo el cual funcionará la aplicación. Aprovecharemos también para conocer las herramientas mediante las cuales ha sido posible la implementación y en qué parte del escenario esta implementación ha cobrado sentido.

3.1 Arquitectura

El escenario que se pinta en la siguiente imagen es el ejemplo de dos usuarios que se conectarán a la aplicación GestDifran 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 se prevee en un futuro próximo que disponga 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.

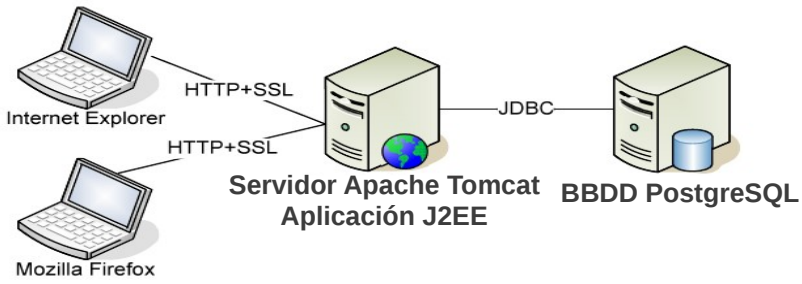

Figura 8. Arquitectura GestDifran

Se ha optado 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.

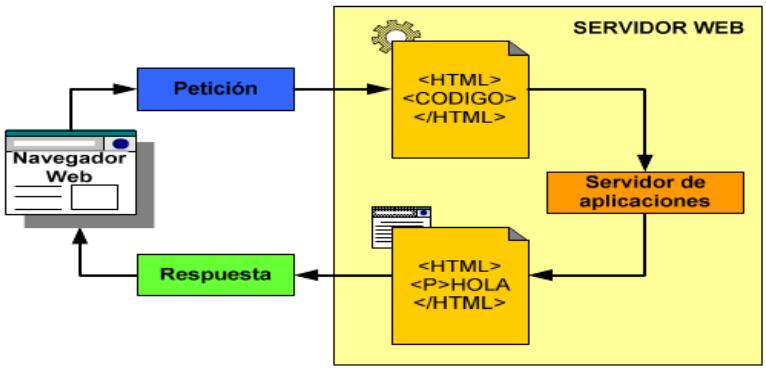

Figura 9. Flujo entre Cliente y Servidor

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 de información que proceda de la base de datos.

3.2 Apache Tomcat

El primer pilar de la aplicación web que analizamos es el servidor. Es el 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, de código abierto e implementado en Java, lo que permite que pueda funcionar en cualquier sistema operativo que disponga de máquina virtual Java.

La versión utilizada en este proyecto es Apache Tomcat 7.0.27.

3.3 Aplicación Java

En el servidor encontraremos el código Java referente a la aplicación GestDifran. Se ha definido bajo una estructura de aplicación dinámica web e implementado haciendo uso de las librerías J2EE.

3.3.1 Módulo web en J2EE

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

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 JSP serán compilados a posteriori, mientras el programa esté en ejecución.

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

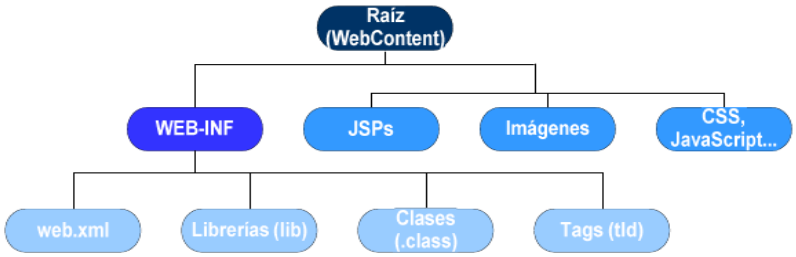

Figura 10. Arquitectura carpetas aplicación web GestDifran

En la carpeta WEB-INF también hayamos el web.xml. Éste es el archivo mediante el cual se define el deploy de la aplicación, es decir es el archivo base del aplicativo y en el cual se han definido una serie de parámetros.

3.3.2 Struts 2, Hibernate, Spring, iText,...

Se ha utilizado un seguido de tecnologías, válidas para escenarios web, para la construcción de la aplicación. Todas juntas integradas forman un complemento ideal para el desarrollo del aplicativo, cada una de ellas se ha utilizado en su mayor potencia para implementar una parte concreta de la aplicación.

3.3.3 IDE 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.

3.3.4 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 se necesita estructurar y englobar la aplicación dentro de un archivo de extensión .war.

Para realizar un empaquetado war, solución adoptada en este proyecto, es 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 configuran 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 despliega el paquete y crea el sistema de archivos según su metodología para desplegar la aplicación.

3.3.5 Java SE Development Kit (JDK)

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

3.4 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 necesitad 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 período de adaptación será menor.

3.5 Secure Sockets Layer (SSL)

La seguridad en aplicaciones Web es un punto importante a tener en cuenta. Para evitar que la información de clientes, usuarios, facturas, presupuestos, etc. 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.

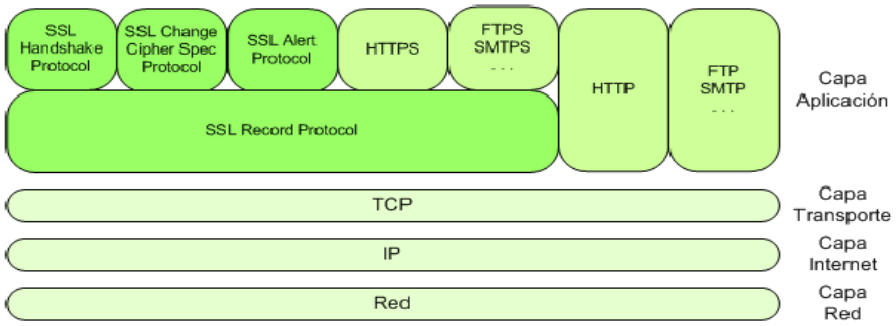

Figura 11. Protocolo SSL

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 la ventana del navegador aparecerá una URL iniciada con https y un icono con un candado.

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 GestDifran está previsto hacer 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 *keystore* y se sitúa en la carpeta *conf* de Apache.

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.

Esta seguridad está prevista realizarla en un futuro próximo y antes que la aplicación hay sido instalada en el servidor.

3.6 Base de Datos PostgreSQL

Para la creación y administración de la base de datos, se utiliza la herramienta propia del PostgreSQL, como es pgAdminIII.

Como norma general, PostgreSQL ofrece una integridad de datos más fuerte que MySQL. Además, numerosos tipos de datos son igualmente a su disposición (tablas, XML, etc). Puede incluso crear a sus propios tipos de datos, o crear sus propias funciones para construir peticiones específicas.

Módulos externos le permiten disponer de tipos de datos preparados para expertos, y PostgreSQL le ofrece igualmente un motor de indexación potente y evolutivo, que le permite acelerar las peticiones más frecuentes a sus aplicaciones...

Este tipo de SGBDR está particularmente adaptado a los cálculos científicos y estadísticas, a aplicaciones financieras o a la gestión fina de las transacciones.

3.7 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 modular 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.

- **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.

- **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 podemos observar el ciclo que realiza una aplicación basada en el modelo-vista-controlador.

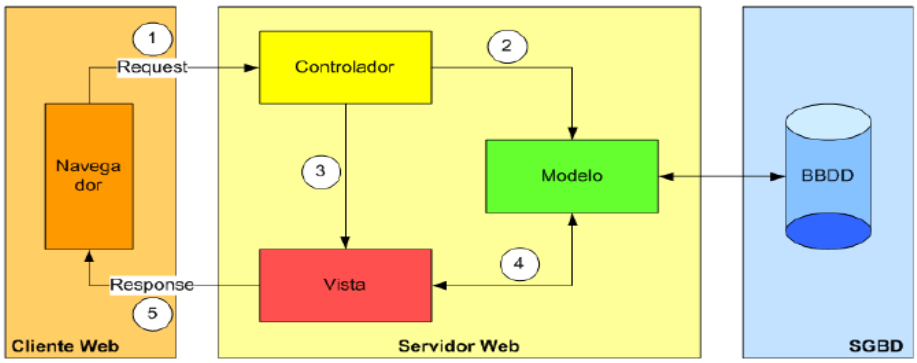

Figura 12. Arquitectura Modelo-Vista-Controlador (MVC)

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 qué tecnología se usa para cada propósito. Además trabajar con este método provoca que exista un número más elevado de archivos.

3.8 Struts 2

Struts 2 es, como el nombre sugiere, la nueva versión del popular Framework de desarrollo web en Java Apache Struts. Sin embargo, contrariamente a lo que cabría esperar, el código de ambos tiene poco que ver, dado que Struts 2 no se basa en el código de Struts 1.x, sino en el de otro Framework de desarrollo web en Java llamado WebWork, un Framework que el creador de Struts consideraba superior a Struts 1.x en varios aspectos.

Para nuestro proyecto, aplicación GestDifran hemos optado por usar la versión Struts 2.2.3.

Struts 2 es un Framework de presentación, dentro de las capas en las que se divide una aplicación en la arquitectura JEE, el cual implementa el controlador del patrón de diseño MVC (Modelo-Vista-Controlador), y que podemos configurar de varias maneras; además proporciona algunos componentes para la capa de vista. Por si fuera poco, proporciona una integración perfecta con otros Frameworks para implementar la capa del modelo (como Hibernate y Spring).

Para hacer más fácil presentar datos dinámicos, el Framework incluye una biblioteca de etiquetas web. Las etiquetas interactúan con las validaciones y las características de internacionalización del Framework, para asegurar que las entradas son válidas, y las salidas están localizadas. La biblioteca de etiquetas puede ser usada con JSP, FreeMarker o Velocity;

también pueden ser usadas otras bibliotecas de etiquetas como JSTL y soporta el uso de componentes JSF.

Además permite agregarle funcionalidades, mediante el uso de plugins, de forma transparente, ya que los plugins no tienen que ser declarados ni configurados de ninguna forma. Basta con agregar al classpath el jar que contiene al plugin, y esto es todo.

El objetivo de Struts 2 es hacer que el desarrollo de aplicaciones web sea fácil para los desarrolladores. Para lograr esto, Struts 2 cuenta con características que permiten reducir la configuración gracias a que proporcionan un conjunto inteligente de valores por defecto. Además, hace uso de anotaciones y proporciona una forma de hacer la configuración de manera automática si usamos una serie de convenciones (y si hacemos uso de un plugin especial).

Por supuesto, podríamos utilizar JSP y servlets para crear una aplicación utilizando el MVC: en una aplicación con JSP y servlets clásica se suelen utilizar páginas JSP para la vista, un servlet como controlador (patrón front controller) y POJOs (Plain Old Java Objects) para el modelo. El modelo, a su vez, suele estar dividido en dos subcapas siguiendo el patrón DAO (Data Access Object).

Sin embargo, no tiene sentido reinventar la rueda cuando podemos recurrir a un framework, como Struts 2, para que se encargue de obtener y transformar los parámetros de las peticiones del cliente, validar los datos, buscar qué acción se debe realizar en el modelo según la petición, buscar la página a mostrar al usuario según la respuesta del modelo, proporcionar los datos del modelo a la vista, ocuparse de la internacionalización, etc.

Lo cierto es que Struts 2 (o más bien el interceptor `param` de Struts 2) nos tiene preparada una sorpresa, y es que basta con crear getters y setters en nuestra clase java con los nombres de los campos de los formularios para que estos se rellenen auto mágicamente, incluso aunque las propiedades no sean cadenas y sea necesaria una sencilla conversión. Este es el trabajo del interceptor `param`, como hemos dicho.

Los interceptores son unos de los elementos más importantes de Struts 2. Se trata de objetos que ejecutan código antes y después de una cierta acción, de forma que podamos añadir nueva funcionalidad no relacionada directamente con el modelo de forma modular. Tenemos interceptores para logging, para validación, para capturar excepciones, etc.

Las acciones son las encargadas de definir cual será la clase que se encargará de procesar la petición solicitada y cual será la respuesta dependiendo del resultado. Estas acciones se registran o bien dentro de struts.xml (si optamos por la configuración mediante XML) o bien con anotaciones en las clases (si optamos por la configuración mediante anotaciones).

La lógica general puede resumirse en el siguiente esquema:

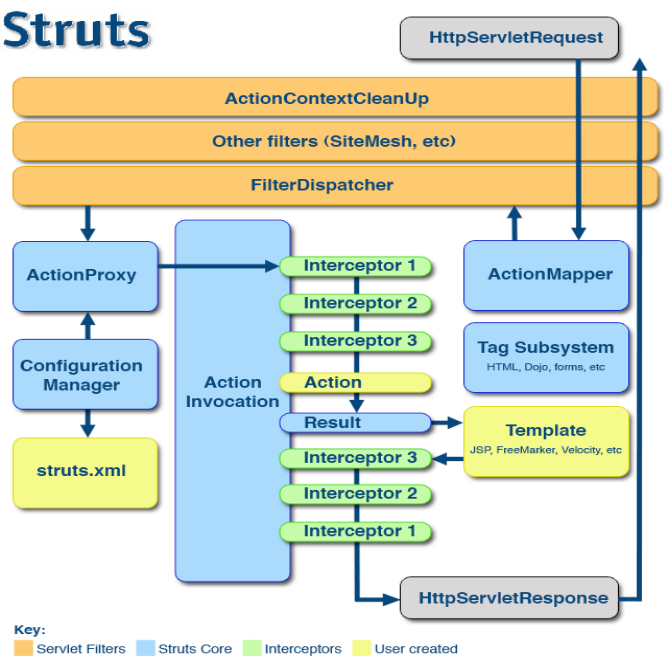

Figura 13. Esquema lógico general de Struts2

En definitiva, las principales características y ventajas que ofrece son:

- Diseño simplificado: Unos de los principales problemas que tenía el Framework Struts 1 era el uso de clases abstractas, cosa que cambia en la versión 2 en la cual se hace uso de Interfaces. Esto le provee una mayor facilidad para la extensión y la adaptación, ya que las interfaces son más fáciles de adaptar que las clases abstractas. Otro cambio es que se busca que las clases sean lo más simples posible con lo que los actions se convierten en POJOs, elementos que además estarán poco acoplados. Los POJOs son clases que cuentan con getter y setter para poder recibir valores desde páginas y cuentan con algunos métodos en los cuales pondremos la lógica de negocio.
- Simplificación de los actions: Como hemos dicho los actions son POJOs. Cualquier clase java con un método execute puede actuar como un Action. Así no se hace necesario implementar ninguna interfaz. Además se introduce la inversión de control en la gestión de los actions.
- Desaparecen los ActionForms: se ven reemplazados por simples JavaBeans que son usados para leer las propiedades directamente. Lo usual es que el propio Action actúe de JavaBean, con lo que se facilita el desarrollo. Además se ha mejorado la lectura de parámetros con el objetivo de no tener únicamente propiedades de tipo String.
- Test simplificados: como los actions engloban la lógica de negocio y los JavaBeans, es más sencillo hacer test unitarios.
- Fácil selección de opciones por defecto: casi todos los elementos de configuración tienen definidos un valor por defecto que se pueden parametrizar, lo que facilita la elección de acciones por defecto.
- Results mejorados: a diferencia de los Action Forwards, los results de Struts 2 son más flexibles a la hora de poder definir múltiples elementos de la vista. Además desaparece la complejidad de utilizar ActionForward, ya que se sustituye por la devolución de Strings.
- Mejoras en Tags: Struts 2 permite añadir capacidades utilizando tags que permite hacer páginas consistentes sin añadir código. Los tags presentan más opciones, están orientados a los results y pueden ser cambiados de forma sencilla. Además se añaden tags de marcado (markup) los cuáles son editables usando templates FreeMarker. Esto significa que podemos hacer que un mismo tag se comporte de forma diferente sin tener que hacer ninguna tarea de programación.
- Se introducen anotaciones: las aplicaciones en Struts 2 pueden usar anotaciones como alternativa a XML y configuraciones basadas en properties.

- Arranque rápido: muchos cambios se pueden hacer sin necesidad de reiniciar el contenedor web.
- Parametrización del controlador: Struts 1 permitía parametrizar el procesador de peticiones a nivel de módulo. Struts 2 lo permite a nivel de action.
- Se puede combinar fácilmente con Spring para permitir la inyección de objetos en nuestra aplicación desde el fichero de configuración de Spring, lo que aumenta la portabilidad y sobre todo la integración de Struts 2 con otras APIs.
- Permite configurar cadenas de filtros, interceptores y validadores de una manera relativamente diferente a la proporcionada por la primera versión, una de las ventajas es el empleo de clases java estándar.
- Soporte de Ajax: se incluye un theme AJAX que nos permite hacer aplicaciones interactivas de forma más sencilla.

En realidad, Struts 2 está más enfocado a tratar la capa vista y la capa controlador, dejando espacio a otras tecnologías para tratar la capa modelo.

3.9 Configuración

Los ficheros de configuración son la base de la construcción. En este punto se puede ver qué archivos requiere GestDifran para implementar el modelo-vista-controlador.

3.9.1 web.xml

El pilar de cualquier aplicación Web, el deployment descriptor conocido sencillamente como el archivo “web.xml”. Es el descriptor de despliegue donde se añade el filtro que sirve Struts 2 a modo de punto de entrada a la aplicación: `StrutsPrepareAndExecuteFilter`. Todas las llamadas que se hagan a nuestra aplicación web pasan a través de este filtro. En versiones anteriores de Struts 2 se usaba un filtro llamado `FilterDispatcher`, sin embargo este ha sido marcado como deprecated y por lo tanto ya no debe ser usado.
`StrutsPrepareAndExecuteFilter` es en realidad el controlador de nuestra aplicación: nosotros sólo tendremos que ocuparnos de la vista y el modelo, llamados resultados y acciones en la terminología de Struts 2.

3.9.2 Configuración de Struts 2

El siguiente paso es realizar la configuración de Struts 2. Únicamente se necesita declarar qué objetos prestarán servicio a las acciones, resultados e interceptores de la aplicación. Este proceso de declaración consiste fundamentalmente en especificar qué clase Java implementará la interfaz necesaria. Casi todos los componentes arquitectónicos de Struts 2 se definen como interfaces. En realidad, el marco de trabajo ofrece implementaciones para casi todos los componentes que el desarrollador necesita utilizar.
De una forma típica, se necesita únicamente implementar acciones y vincularlas a los resultados e interceptores implícitos.
Existen dos métodos para la declaración de la arquitectura: a través de archivos de configuración basados en XML, o a través de anotaciones Java.
Independientemente de si los componentes de la aplicación Struts 2 están declarados en XML o bien en anotaciones, el marco de trabajo las traduce a los mismos componentes en tiempo de ejecución.

3.9.2.1 Arquitectura declarativa basada en XML

Struts 2 permite utilizar archivos XML para describir los componentes arquitectónicos deseados para Struts 2. En general, los documentos XML consisten en elementos que representan los componentes de la aplicación.
El marco de trabajo utiliza un archivo específico como punto de entrada en esta descripción más amplia. Este punto de entrada es el archivo “`struts.xml`”, que se coloca en el paquete

raíz de la aplicación. En este archivo se especifica la relación entre la URL de una petición, la clase Java que ejecutará la acción de la petición, y la página JSP que se encargará de mostrar la respuesta a la petición del usuario.

Dentro de este archivo se configuran algunas constantes que modifican, de una manera mínima, el comportamiento el filtro de Struts 2. También se configuran paquetes, que son conjuntos de acciones que podemos organizar debajo de un mismo namespace. Dentro de los paquetes podemos definir un conjunto de acciones, cada una de las cuales responderá a una petición. Y dentro de las acciones podemos definir resultados, que son la páginas o vistas a las que se enviará al usuario dependiendo del resultado de una acción.

Si bien es posible declarar todos nuestros componentes en "`struts.xml`", con frecuencia los desarrolladores utilizan este archivo únicamente para incluir archivos secundarios XML para modular sus aplicaciones, y es así exactamente como se ha llevado a cabo en este proyecto.

En última instancia, corresponde al desarrollador elegir un mecanismo concreto para declarar la arquitectura. Lo más importante es comprender los conceptos de la arquitectura declarativa de Struts 2. Una vez comprendidos, pasar de un mecanismo basado en XML a uno basado en anotaciones Java debería ser una tarea trivial.
En este proyecto, se usa XML en la aplicación, por varias razones. En primer lugar, creo que la versión XML es la más adecuada para el aprendizaje del marco de trabajo. Los archivos XML son, probablemente, los más conocidos entre nosotros los programadores; de forma más importante aún, es el tipo de archivos que ofrece una notación más centralizada de los componentes de una aplicación. Todo ello facilita el estudio de los materiales en el momento de aprendizaje del marco de trabajo. En segundo lugar, las anotaciones están en plena evolución en estos momentos, Los desarrolladores de Struts 2 van encaminados hacia un sistema de configuración cero que opta por la convención frente a la configuración , en el que las anotaciones cumplirán la función de mecanismo de anulación elegante allí donde no se sigue ninguna convención. Muchos desarrolladores ya utilizan este sistema, pero en estos momentos no todos creemos que sea el mejor modo de aprender a trabajar con el marco de trabajo.

3.9.2.2 Arquitectura declarativa basada en anotaciones Java

Las anotaciones (una característica relativamente novedosa del lenguaje Java) permiten añadir metadatos directamente a los archivos de origen de Java. Uno de los objetivos más loables de las anotaciones Java es el de ofrecer soporte a herramientas que puedan leer metadatos de una clase Java y puedan hacer algo útil con dicha información.
Al igual que los elementos equivalentes en XML, estas anotaciones contienen metadatos que el marco de trabajo utilizará para crear los componentes de tiempo de ejecución de la aplicación.
Muchos consideran el mecanismo basado en anotaciones una solución más elegante que el mecanismo XML. Para empezar, el mecanismo de anotaciones se combina en buena parte con una deducción de la información basada en convenciones. En otras palabras: parte de la información que debe especificarse de forma explícita en los elementos XML y puede deducir-se automáticamente a partir de la estructura de paquete Java a la que pertenecen las clases anotadas. Por ejemplo, no es necesario especificar el nombre de la clase Java, puesto que está claramente implícita en la ubicación física de las anotaciones. Muchos desarrolladores aprecian también el hecho de que las anotaciones eliminen parte del desorden de archivos XML que parece aumentar año tras año en la ruta de clases de aplicaciones Web.

Cuando usamos anotaciones para realizar la configuración de una aplicación con Struts 2, necesitamos hacer uso de un plugin, el plugin "`convention`" (el cual se encuentra en el jar "`struts2-convention-plugin-2.2.3`" que agregamos para la biblioteca de "`Struts2Anotaciones`"). Este plugin proporciona la característica de poder crear una aplicación Struts, siguiendo una serie de convenciones de nombres tanto para los `results`, los `Actions`, en fin, para todo lo relacionado con Struts. Esto sin necesidad de crear ningún archivo de configuración ni usar ninguna anotación.

3.10 Capa Modelo

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

3.10.1 Base de datos

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

Para llevar a cabo el diseño de la base de datos se toma 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... GestDifran contiene un total de 12 entidades: nif, perfil, usuario, empresa, cliente, obra, factura, medicion, producto, presupuesto, factura_producto y presupuesto_producto. El segundo paso es identificar las relaciones entre entidades. No solo eso, también es necesario conocer si es una relación uno a uno, uno a varios, varios a uno o varios a varios.

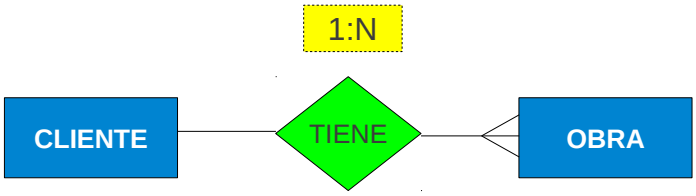

Figura 14. Relación entre entidades

Una vez decididas las entidades y conocidas sus relaciones es necesario detallar cada entidad. Para ello se especifican 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.

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

3.10.1.1 Entidades BBDD

Se detallan a continuación las diferentes entidades de base de datos de la aplicación GestDifran y sus correspondientes atributos:

Entidad NIF
Contiene la información de los diferentes números de identificación fiscal.

NIF			
Atributo	Dominio	Clave	Nulo
id_nif	serial	Primaria	No

serie	character (8)		No
letra	character (1)		No

Tabla 25. Tabla nif

Entidad PERFIL
Contiene toda la información de los diferentes perfiles de usuarios del sistema.

PERFIL			
Atributo	Dominio	Clave	Nulo
id_perfil	integer	Primaria	No
descripcion	character (20)		No

Tabla 26. Tabla perfil

Entidad USUARIO
Contiene la información detallada de cada usuario del sistema.

USUARIO			
Atributo	Dominio	Clave	Nulo
id_usuario	serial	Primaria	No
nif	integer	Secundaria	No
nombre	character (50)		No
apellido1	character (50)		No
apellido2	character (50)		SI
domicilio	character (250)		No
codigo_postal	character (5)		No
poblacion	character (25)		No
provincia	character (25)		No
telefono	character (9)		Si
mobil	character (9)		Si
fax	character (9)		Si
email	character (50)		Si
login	character(25)		No
passw	character (5)		No
activo	boolean		No
perfil	integer	Secundaria	No
f_alta	date		No
f_modificacion	date		SI
f_baja	date		SI
observaciones	character (250)		Si

Tabla 27. Tabla usuario

Entidad EMPRESA
Contiene toda la información referente a la empresa Serralleria Difran

EMPRESA			
Atributo	Dominio	Clave	Nulo
id_empresa	serial	Primaria	No

nif	integer	Secundaria	No
razon_social	character (50)		No
domicilio	character (250)		No
codigo_postal	character (5)		No
poblacion	character (25)		No
provincia	character (25)		No
telefono	character (9)		Si
mobil	character (9)		Si
fax	character (9)		Si
email	character (50)		Si
sucursal	character (100)		No
entidad	character (4)		No
oficina	character (4)		No
dc	character (2)		No
cuenta	character (10)		No
f_modificacion	date		Si
observaciones	character (250)		Si

Tabla 28. Tabla empresa

Entidad CLIENTE
Contiene toda la información de los clientes de la empresa.

CLIENTE			
Atributo	Dominio	Clave	Nulo
id_cliente	serial	Primaria	No
nif	integer	Secundaria	No
nombre_razon	character (250)		No
domicilio	character (250)		No
codigo_postal	character (5)		No
poblacion	character (25)		No
provincia	character (25)		No
telefono	character (9)		Si
mobil	character (9)		Si
fax	character (9)		Si
email	character (50)		Si
observaciones	character (250)		Si
activo	boolean		No
f_alta	date		No
f_modificacion	date		Si
f_baja	date		Si

Tabla 29. Tabla cliente

Entidad OBRA
Contiene toda la información de las obras de los clientes de la empresa.

OBRA			
Atributo	Dominio	Clave	Nulo
id_obra	serial	Primaria	No

cliente	integer	Secundaria	No
descripcion	character (250)		No
direccion	character (250)		No
poblacion	character (50)		No
codigo_postal	character (5)		No
provincia	character (50)		No
activa	boolean		No
f_alta	date		No
f_modificacion	date		Si
f_baja	date		Si
observaciones	character (250)		Si

Tabla 30. Tabla obra

Entidad MEDICION
Contiene toda la información de los diferentes tipos de mediciones para los conceptos.

MEDICION			
Atributo	Dominio	Clave	Nulo
id_medicion	integer	Primaria	No
descripcion	character (20)		No

Tabla 31. Tabla medicion

Entidad FACTURA
Contiene la información de las facturas realizadas a clientes.

FACTURA			
Atributo	Dominio	Clave	Nulo
id_factura	serial	Primaria	No
cliente	integer	Secundaria	No
obra	integer	Secundaria	Si
referencia	character (250)		Si
fecha	date		No
f_modificacion	date		Si
base	double		No
iva	double		No
total	double		No
cobrada	boolean		No
observaciones	character (250)		Si

Tabla 32. Tabla factura

Entidad PRESUPUESTO
Contiene la información de los presupuestos realizados a clientes.

PRESUPUESTO			
Atributo	Dominio	Clave	Nulo
id_presupuesto	serial	Primaria	No
factura	integer	Secundaria	Si
cliente	integer	Secundaria	No

obra	integer	Secundaria	Si
referencia	character (250)		Si
fecha	date		No
f_modificacion	date		Si
base	double		No
iva	double		No
total	double		No
aceptado	boolean		--
observaciones	character (250)		Si

Tabla 33. Tabla presupuesto

Entidad PRODUCTO
Contiene la información de los productos a presupuestar o facturar.

PRODUCTO			
Atributo	Dominio	Clave	Nulo
id_producto	serial	Primaria	No
cantidad	doble		No
medicion	integer	Secundaria	No
precio	double		No
importe	double		No
descripcion	character(250)		No

Tabla 34. Tabla producto

Entidad FACTURA_PRODUCTO
Contiene la relación entre factura y producto, almacenando así todos los productos para una factura.

FACTURA_PRODUCTO			
Atributo	Dominio	Clave	Nulo
id	serial	Primaria	No
id_factura	integer	Secundaria	No
id_producto	integer	Secundaria	No

Tabla 35. Tabla factura_producto

Entidad PRESUPUESTO_PRODUCTO
Contiene la relación entre factura y producto, almacenando así todos los productos para una factura.

PRESUPUESTO_PRODUCTO			
Atributo	Dominio	Clave	Nulo
id	serial	Primaria	No
id_presupuesto	integer	Secundaria	No
id_producto	integer	Secundaria	No

Tabla 36. Tabla presupuesto_producto

3.10.1.2 Diseño de la base de datos (Modelo relacional)

El último paso del diseño del modelo consiste en dibujar el diagrama de entidad relación.

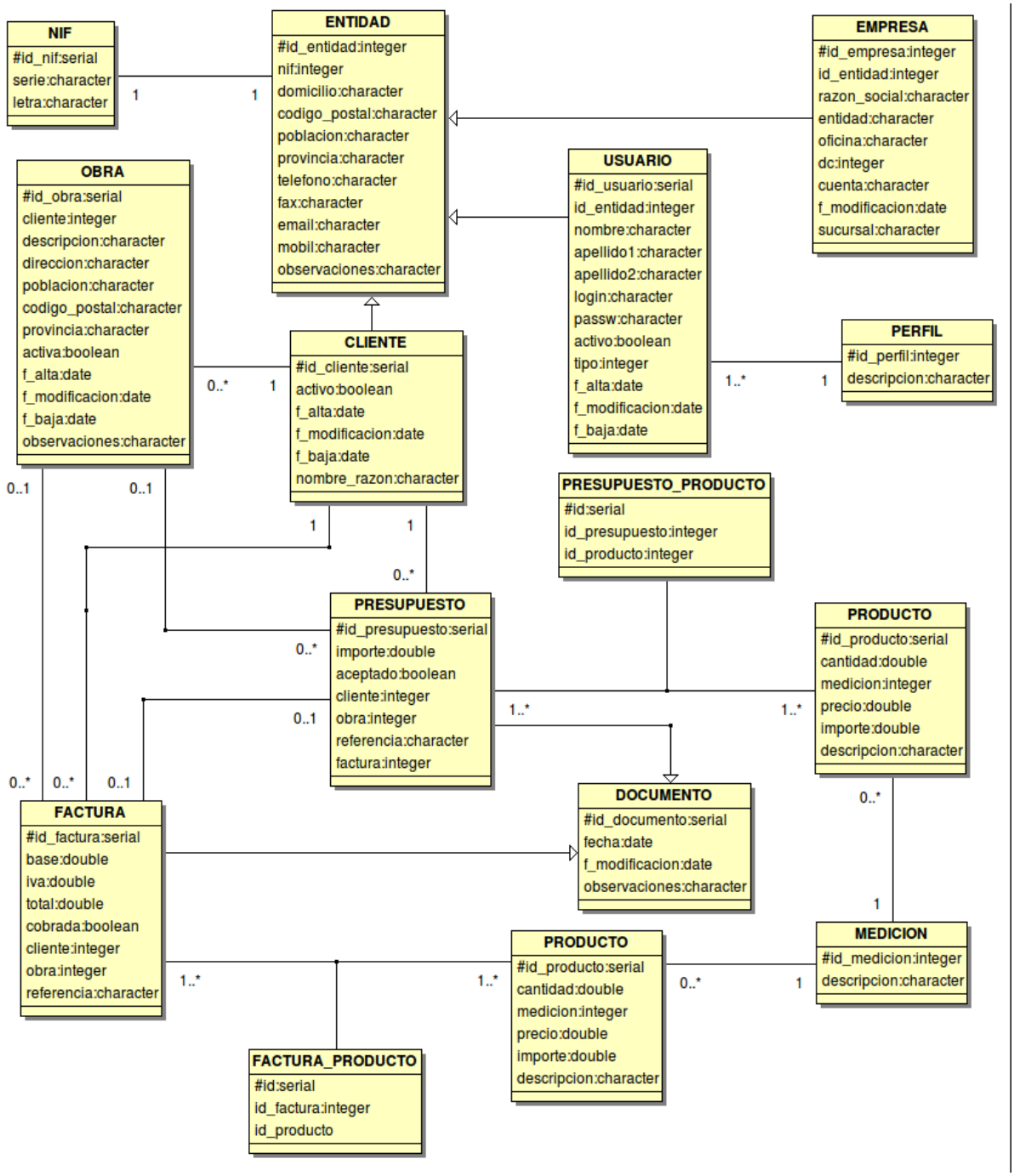

Figura 15. Modelo E/R

3.10.1.3 Script de la Base de Datos

Una vez finalizado el diseño es necesario trasladarlo a la base de datos PostgreSQL. En este proyecto se ha optado porque cada entidad se convierta en una tabla, con sus atributos definidos y sus relaciones estipuladas.

El script de la base de datos “gestdifran” lo podemos encontrar adjunto en el archivo

comprimido “sgutierrezmel_producto.zip” bajo el nombre de “DML_gestdifran.sql”.

3.10.2 Spring

¿Por qué utilizamos Spring con Struts 2? La confusión nace del hecho de que Spring es un amplio marco de trabajo que contiene soluciones para muchos aspectos diferentes de una aplicación J2EE. Spring cuenta incluso con su propio marco de trabajo para aplicaciones Web MVC, pero no es esa la parte que hemos necesitado para este proyecto, sino que nos hemos centrado en la parte de Spring que permite la inyección de dependencias. Muchos desarrolladores de Struts 2 consideran este servicio de gestión de recursos como parte esencial de una aplicación Web bien construida.

Como ya se ha mencionado, Spring tiene múltiples usos, pero uno de sus usos más populares es la gestión de la creación de objetos y la inyección de dependencias en dichos objetos a medida que son creados. En lugar de usar código para adquirir nuestros recursos, utilizaremos metadatos, situados por lo general en un archivo “`applicationContext.xml`”, para conocer las dependencias de los objetos que gestiona. Spring ofrece varias vías para inyectar dependencias en los objetos gestionados, pero un método común consiste en utilizar métodos `setter` expuestos por el objeto gestionado.

La idea básica es que necesitamos dar una oportunidad a Spring para que gestione los objetos que son creados por Struts 2. Una de las vías para permitir a Spring realizar esta gestión consiste en ofrecer una extensión Spring del `ObjectFactory` de Struts 2, la clase que crea cada uno de los objetos utilizados en el marco de trabajo.
En primer lugar, es necesario descargar y añadir el plug-in de Spring a nuestra aplicación. Una vez tengamos colocado este plug-in, Spring tiene la oportunidad de gestionar la creación de cualquier objeto creado por el marco de trabajo.
Se puede encontrar el plug-in de Spring en el registro de plug-ins de Struts 2, en http://cwiki.apache.org/S2PLUGINS/home.html. Se necesita también el JAR Spring, `spring.jar`, que lo podemos encontrar en www.springframework.org. Con estos dos plug-in añadidos al directorio de bibliotecas, únicamente se necesita crear el propio contenedor Spring. Dado que estamos construyendo aplicaciones Web, utilizamos un escuchador de contexto de aplicación Spring. Este escuchador viene integrado en el JAR de Spring y se configura con el siguiente fragmento de código en el archivo “`web.xml`”:

```
<listener>
      <listener-class>org.springframework.web.context.ContextLoaderListener
      </listener-class>
</listener>
```

Ahora, estaría completamente listo para empezar a administrar los objetos con Spring. Pero, Spring no se limitará a empezar a gestionar todos los objetos. Es necesario indicar a Spring que debe intervenir. Para que Spring administre los objetos, es necesario declarar dichos objetos como beans de Spring en un archivo de configuración Spring. Por defecto, el contenedor Spring creado por `Context-LoaderListener` busca metadatos en un archivo XML en `/WEB-INF/applicationContext.xml`.

Y ya que nos hemos molestado en configurar el plug-in de Spring, haremos bien en sacar todo el partido posible del mismo. En realidad, el uso de Spring para gestionar la API de Persistencia es fácil. Spring viene equipado con paquetes dedicados para hacer de la gestión de la API de Persistencia de Java una tarea fácil y potente. Para pedirle a Spring que haga todo esto por nosotros lo haremos a través del archivo “`applicationContext.xml`”.

En definitiva, para el desarrollo de la aplicación GestDifran me ha interesado principalmente el contenedor de Inyección de Dependencias de Spring (anteriormente llamado Inversión de Control).
Este contenedor permite mediante la definición de un XML (mediante anotaciones en sus versiones más recientes), configurar las clases y sus dependencias. De esta manera se consigue un desacoplamiento total de las clases.

3.10.3 Patrón DAO

En las aplicaciones que implementan persistencia de datos, el programador se puede encontrar con problemas de incompatibilidades como podrían ser: diferentes tipos de BBDD, diferentes implementaciones según el fabricante, etc.
Para solucionar la problemática de las incompatibilidades de las bases de datos, se definió el patrón de diseño DAO. Un objeto DAO encapsula la información y se pone en contacto con el sistema de persistencia, sea cual sea, para gestionar la transferencia de información.
Para realizar esto, de forma completamente transparente al resto del sistema, los objetos DAO hacen uso de las interfaces de Java, definiendo en la interfaz del DAO solo los métodos necesarios para el transporte de datos hacia y desde la BBDD.

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 ver en la siguiente imagen.

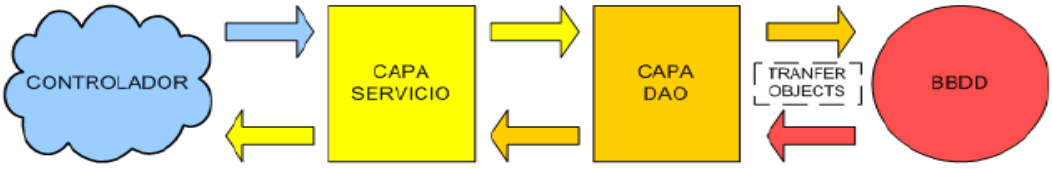

Figura 16. Arquitectura patrón DAO

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.

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 Servicio es donde 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.

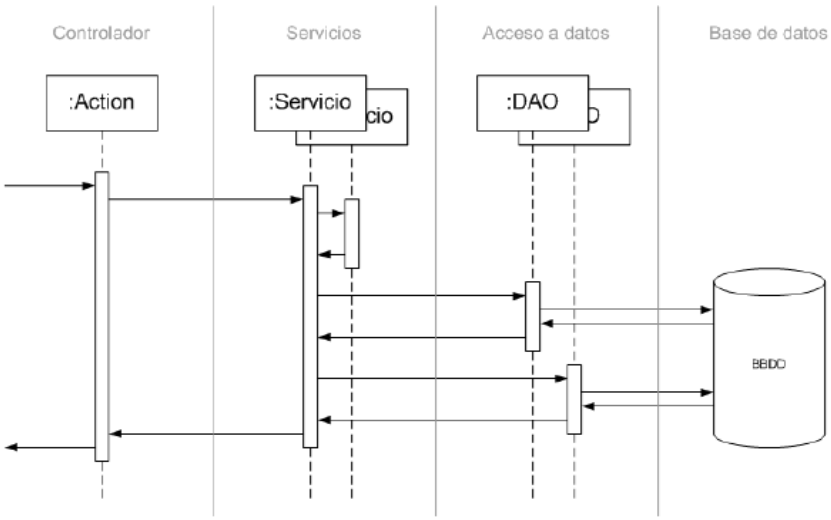

Figura 17. Acceso a BBDD mediante Servicios y DAO

En nuestro proyecto, se ha creado una interfaz que todos nuestros DAOs implementan y una implementación por defecto que todos extenderán.

3.10.4 Hibernate

Una de las problemáticas que surge a la hora de trabajar con la base de datos es que ésta sigue un modelo relacional y en cambio el código Java está orientado a objetos. Hibernate es una tecnología que nos permite 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 esta tabla y un fichero XML donde se informe la correspondencia entre atributos de la tabla y propiedades de la clase.

Figura 18. Mapeo Hibernate

La clase Java está formada por un seguido de propiedades, constructores y sus correspondientes métodos getters y setters para almacenar y recuperar las propiedades.

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 qué entidad de la base de datos relaciona a qué objeto Java. No obstante, también existe la opción de realizar el mapeo de clases mediante anotaciones, directamente en nuestra clase Java. Es de esta manera como se ha llevado a cabo en nuestro proyecto.

Posteriormente hay que etiquetar todos los atributos de la entidad empezando por 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. Un parámetro importante es *lazy* que nos permitirá importar los datos de tablas relacionadas convertidas ya en objetos de forma transparente.

3.11 Capa Vista

Para llevar a cabo la capa visual se han utilizado diversas herramientas y se toma como base las Java Server Pages (JSP).

3.11.1 Java Server Pages (JSP)

En una aplicación Web clásica, estas tareas de vista equivalen por lo general a la creación de una página HTML que es devuelta al cliente. Las opciones inteligentes por defecto del marco de trabajo se ajustan perfectamente a este uso. Por defecto, el marco de trabajo utiliza un tipo de resultados que funciona con páginas JSP para mostrar estas páginas de respuesta. Todo lo que se necesita saber es que un resultado es el elemento en el que almacenamos la ubicación de nuestra página JSP. En tanto el proyecto se mantenga dentro del curso JSP correcto, esto es todo lo que necesitamos saber.

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.

Struts 2 también proporciona una API de etiquetas que potencian la funcionalidad necesaria para crear de forma dinámica páginas Web robustas mediante el aprovechamiento de la renderización condicional y la integración de datos del modelo de dominio de la aplicación ubicado en `ValueStack`. Struts 2 viene equipado con muchos tipos diferentes de etiquetas. Desde un punto de vista organizativo, las etiquetas pueden dividirse en cuatro categorías: etiquetas de datos, etiquetas de control de flujo, etiquetas UI y etiquetas mixtas.

Las etiquetas de Struts 2 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.

Cuando las etiquetas de Struts 2 resultan insuficientes el código Java cobra protagonismo. Es una solución menos elegante porque complica la lectura de la JSP pero resulta muy efectiva. El código java es rápidamente reconocible porque se limita entre etiquetas <% %> para que el servidor lo pueda detectar.

3.11.2 Struts 2 Taglibs

Struts 2 proporciona una librería de etiquetas que facilitan, entre otras, las tareas de validación e internacionalización. Estas etiquetas, cuyo TLD podemos encontrar en META-Inf/struts-tags.tld, en el jar de struts2-core, pueden utilizarse tanto con JSP como con Velocity o FreeMarker.

Dentro de la JSP usaremos las etiquetas de Struts para mostrar el valor correspondiente a la propiedad del objeto. Indicamos que usaremos la biblioteca de etiquetas (taglib) "`struts-tags`". Por convención usaremos el prefijo "**`s`**" para hacer referencia a esta biblioteca:

```
<%@ taglib prefix="s" uri="/struts-tags" %>
```

Dentro de esta biblioteca se encuentran contenidas todas las etiquetas de Struts 2. Tiene etiquetas control de flujo (`if`, `iterator`), de datos (`a`, `bean`, `i18n`), de formulario (`checkbox`, `form`, `label`, `submit`), para el manejo de componentes ajax (`autocompleter`, `tree`).

3.11.3 Tiles

Un punto importante en una aplicación Web es siempre su diseño. Puesto que estamos hablando 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 2, 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 aprovecharlo tantas veces como queramos.

Para construir una nueva pantalla simplemente hará falta realizar una nueva jsp donde informar, otra vez con etiquetas Tiles, que plantilla usamos y que jsp queremos situar en cada sección.

3.11.4 Cascading Style Sheets (CSS)

Otro aspecto que se ha adoptado a nivel gráfico es el uso de una hoja de estilo externa. 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 GestDifran podemos comprobar que cualquiera de las últimas versiones de los exploradores más extendidos es válido para trabajar:

- Mozilla Firefox 11.0
- Internet Explorer 9
- Opera 11.62
- Google Chrome
- Safari 5.1

3.11.5 JavaScript

El JavaScript es un lenguaje de programación muy simple pensado para añadir pequeñas funcionalidades a nivel de cliente (explorador). En GestDifran 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 2.

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 GestDifran es modificar los datos de una factura o un presupuesto. Esto se realiza pulsando un botón y por motivos de seguridad se ha decidido que el usuario corrobore que realmente quiere llevar a cabo esa acción mediante la aparición de un mensaje. Para implantarlo se decide llamar al método *confirmSave()* de JavaScript y así evitar que esa validación llegue al servidor y sea necesario crear una nueva pantalla.

3.11.6 jQuery

El mejor resumen de lo que es jQuery lo podemos encontrar en el lema de su propia página web: “La librería JavaScript para escribir menos y hacer más”. Ampliando algo más esta definición, dejémoslo en que es una forma de convertir el desarrollo de la parte de cliente de una aplicación web en algo mucho más divertido, rápido y sencillo, facilitando la interacción con los elementos del árbol de documento, el manejo de eventos, el uso de animaciones, etc.

Validar un formulario sin tener la necesidad de recargar el documento completo es claramente una practica más que recomendable. Para lograr esto JavaScript es la mejor opción, porque sacando algún que otro método es el más simple, y si utilizamos para su implementación jQuery, bueno, no hay mucho más que pensar.

Para jQuery hay varios plugin para la validación de formularios, pero en este proyecto se ha hecho uso de *Validate* que es uno de los más recientes.

Validate es un plugin completamente gratuito, que nos permite validar nuestros formularios de manera muy simple, y tiene un peso de apenas 9 KB, muy buen tamaño para no aumentar los tiempos de descarga.

El plugin se puede configurar a un nivel muy importante, no solo podemos personalizar el mensaje de error, sino el tipo de dato a validar, y todo con apenas unas líneas de código.

Otro plugin de jQuery que hemos utilizado para esta aplicación ha sido Tablesorter que ha permitido crear tablas ordenables a partir de una tabla HTML estándar. Haciendo clic sobre las cabeceras de cada columna para ordenar.

3.12 Capa Controlador

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

Comenzamos hablando un poco de los componentes que forman a Struts 2. El corazón de Struts 2 es un filtro, conocido como el `StrutsPrepareAndExecuteFilter`. Este es el punto de entrada del Framework. A partir de él se lanza la ejecución de todas las peticiones que involucran al Framework.

Las principales responsabilidades del `StrutsPrepareAndExecuteFilter` son:

- Ejecutar los **`Actions`**, que son los manejadores de las peticiones.
- Comenzar la ejecución de la cadena de interceptores (de la que hablaremos en un momento).
- Limpiar el **`ActionContext`**, para evitar fugas de memoria.

Struts 2 procesa las peticiones usando tres elementos principales:

- Interceptores
- Acciones

- Resultados

3.12.1 Interceptores

Los interceptores son clases que siguen el patrón interceptor. Estos permiten que se implementen funcionalidades cruzadas o comunes para todos los `Actions`, pero que se ejecuten fuera del `Action` (por ejemplo validaciones de datos, conversiones de tipos, población de datos, etc.).

Los interceptores realizan tareas antes y después de la ejecución de un `Action` y también pueden evitar que un `Action` se ejecute (por ejemplo si estamos haciendo alguna validación que no se ha cumplido).

Sirven para ejecutar algún proceso particular que se quiere aplicar a un conjunto de `Actions`. De hecho muchas de las características con que cuenta Struts 2 son proporcionadas por los interceptores.

Si alguna funcionalidad que necesitamos no se encuentra en los interceptores de Struts podemos crear nuestro propio interceptor y agregarlo a la cadena que se ejecuta por default.

De la misma forma, podemos modificar la cadena de interceptores de Struts, por ejemplo para quitar un interceptor o modificar su orden de ejecución.

Algunos de los interceptores más importantes que vienen integrados y pre-configurados en Struts 2 son: `Alias, Chaining, Checkbox, Conversion Error, Create Session, Execute and Wait, file Upload, Logging, Parameters, Prepare, Servlet Configuration, Roles, Timer, Validation, Workflow.`

Por ejemplo, pueden ser muy interesantes el interceptor `Execute and Wait` que envía al usuario a una página de espera intermedia mientras el `Action` se ejecuta en Background, y el interceptor `Validation` que proporciona a los `Actions` soporte para validaciones de datos.

Cada interceptor proporciona una característica distinta al `Action`. Para sacar la mayor ventaja posible de los interceptores, un `Action` permite que se aplique más de un interceptor. Para lograr esto Struts 2 permite crear pilas o stacks de interceptores y aplicarlas a los `Actions`. Cada interceptor es aplicado en el orden en el que aparece en el stack. También podemos formar pilas de interceptores en base a otras pilas.

Una de estas pilas de interceptores es aplicada por default a todos los `Actions` de la aplicación (aunque si queremos, también podemos aplicar una pila particular a un `Action`).

3.12.2 Acciones

Las acciones o `Actions` son clases encargadas de realizar la lógica para servir una petición. Cada URL es mapeada a una acción específica, la cual proporciona la lógica necesaria para servir a cada petición hecha por el usuario.

Estrictamente hablando, las acciones no necesitan implementar una interfaz o extender de alguna clase base. El único requisito para que una clase sea considerada un `Action` es que debe tener un método que no reciba argumentos que regrese ya sea un `String` o un objeto de tipo `Result`. Por default el nombre de ese método debe ser “`execute`” aunque podemos ponerle el nombre que queramos y posteriormente indicarlo en el archivo de configuración de Struts.

Cuando el resultado es un `String` (lo cual es lo más común), el `Result` correspondiente se obtiene de la configuración del `Action`. Esto se usa para generar una respuesta para el usuario, lo cual veremos en el próximo apartado.

Los Actions pueden ser objetos java simples (POJOs) que cumplan con el requisito anterior, aunque también pueden implementar la interface "com.opensymphony.xwork2.Action" o extender una clase base que proporciona Struts 2, "com.opensymphony.xwork2.ActionSupport" (lo cual nos hace más sencilla su creación y manejo).

La interfaz Action proporciona un conjunto de constantes con los nombres de los Results más comunes. Esta interfaz luce de la siguiente forma:

```
public interface Action
{
    public static final String SUCCESS = "success";
    public static final String NONE = "none";
    public static final String ERROR = "error";
    public static final String INPUT = "input";
    public static final String LOGIN = "login";

    public String execute() throws Exception;
}
```

La clase "ActionSupport" implementa la interfaz "Action" y contiene una implementación del método "execute()" que regresa un valor de "SUCCESS". Además proporciona unos cuantos métodos para establecer mensajes, tanto de error como informativos, que pueden ser mostrados al usuario.

3.12.3 Resultados

Después que un Action ha sido procesado se debe enviar la respuesta de regreso al usuario, esto se realiza usando results. Este proceso tiene dos componentes, el tipo del result y el result mismo.

El tipo del result indica cómo debe ser tratado el resultado que se le regresará al cliente. Por ejemplo un tipo de Result puede enviar al usuario de vuelta una JSP (lo que haremos más a menudo), otro puede redirigirlo hacia otro sitio, mientras otro puede enviarle un flujo de bytes (para descargar un archivo, por ejemplo).

Un Action puede tener más de un result asociado. Esto nos permitirá enviar al usuario a una vista distinta dependiendo del resultado de la ejecución del Action. Por ejemplo, en caso de que todo salga bien, enviaremos al usuario al result "success", si algo sale mal lo enviaremos al result "error", o si no tiene permisos lo enviaremos al result "denied".

3.13 Arquitectura en capas de la aplicación

El siguiente diagrama muestra la arquitectura con la que se ha llevado a cabo el desarrollo de la aplicación GestDifran.

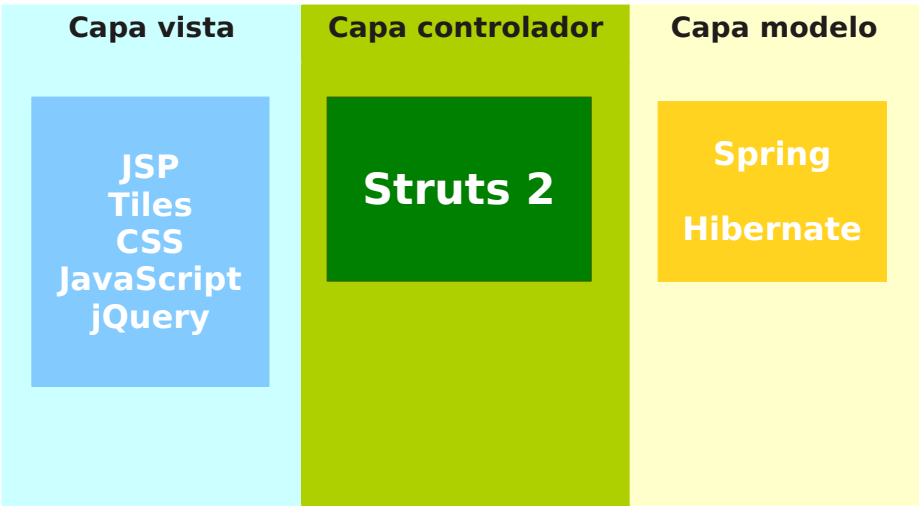

Figura 19. Arquitectura de la aplicación

3.14 iText. Generación de PDFs

Uno de los requerimientos de GestDifran es generar dinámicamente archivos pdf (facturas o presupuestos). Esto comporta la necesidad de extraer información del aplicativo para su envío por email o impresión.

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 que el usuario haga una petición, Struts 2 mediante un Action use la librería de iText y se devuelva una respuesta que logre abrir este tipo de archivos. Para entornos web como GestDifran 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.

3.15 Diagrama de clases

Esta es la estructura principal del proyecto donde dentro de la carpeta “resultados” encontramos todas las páginas jsp pertenecientes a la vista de la aplicación, dentro de la carpteta “css” las hojas de estilo externas, dentro de “js” los archivos Javascript y jQuery, dentro de “tiles” las plantillas definidas de la aplicación y dentro de la carpeta “src” las demás clases java (acciones, DAOs y POJOs).

Figura 20. Estructura proyecto

Se muestra el diagrama de clases principales de la aplicación estructurado en cuatro paquetes principales:

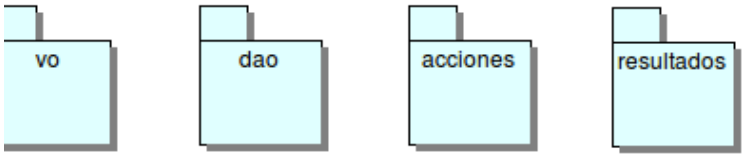

Figura 21. Diagrama de paquetes principales

Y que se detallan a continuación:

- **src.com.vo**: contiene todas las entidades de la aplicación, clases sencillas (POJO) sin ningún tipo de lógica. Este diagrama será exactamente igual al de la “Figura 15. Modelo E/R”.
- **src.com.dao**: contiene los objetos de acceso a base de datos (DAO). Que, como bien

hemos dicho, será una interfaz DAO genérica con los principales métodos usados para manipular nuestras entidades en la base de datos, una implementación de ésta y un DAO para cada entidad.

- **src.com.acciones**: contiene las clases de la lógica de negocio de la aplicación GestDifran, es decir, las acciones.
- **resultados**: contiene las clases que forman parte de la capa de control de la aplicación, capa implementada mediante jsp's.

Capítulo 4. Implementación

4.1 Evaluación

La evaluación del proyecto es un punto muy importante ya que de esta depende que la aplicación web contenga todas las necesidades del usuario y por tanto, el éxito de esta aplicación.

La usabilidad se refiere a la capacidad de un software de ser comprendido, aprendido, usado y ser atractivo para el usuario, en condiciones específicas de uso. Los puntos seguidos para conseguirla son los siguientes:

- La facilidad de comprensión de la estructuración de la aplicación, así como de las funcionalidades y contenidos que en esta pueden observarse.
- La sencillez de uso de la aplicación en las etapas iniciales.
- La rapidez con la que el usuario encuentra aquello que busca.
- La facilidad percibida en la navegación en términos de tiempo requeridos y tareas precisadas para obtener los resultados perseguidos.
- La capacidad del usuario del sistema para controlar aquello que hace y el lugar en el que se encuentra en cada momento.

Estos son algunos de los puntos seguidos para conseguir una aplicación con un alto grado de usabilidad, pero existen numerosas reglas y recomendaciones más.

Los sistemas que mejor se ajustan a las necesidades del usuario mejoran la productividad y la calidad de las acciones y las decisiones. El diseño centrado en el usuario resulta en productos de mayor calidad de uso, más competitivos en un mercado que demanda productos de fácil uso.

4.2 Pruebas

Las primeras pruebas realizadas han sido las pruebas de diseño. Había que realizar un formato único para los formularios así como un estilo único también para los listados y botones. Comprobar que quedara todo centrado y alineado.

Las pruebas referentes a la navegabilidad, enlaces rotos y a la accesibilidad son otro tipo de pruebas que se han ido realizando simultáneamente con el desarrollo de la aplicación. Tras realizar algún cambio importante en la misma, había que probar que hubiera quedado bien y que funcionara como se esperaba.

4.2.1 Validación CSS

El código de las hojas de estilo CSS ha sido comprobado a través del validador W3C con la versión 3 (http://jigsaw.w3.org/css-validator/).

4.2.2 Comprobación de enlaces rotos

Para una navegabilidad sin errores, se han comprobado todos los enlaces de la aplicación. Existe un gran numero de software que realiza esta tarea de forma gratuita. Para este caso se ha utilizado la siguiente página http://www.anybrowser.com/.

4.2.3 Pruebas contra la Base de Datos

Por último, la prueba más importante y que más tiempo nos ha llevado ha sido la de realizar pruebas con la base de datos “gestdifran”.
Para ello, lo primero que se ha hecho es realizar un monto de inserciones para algunas de las tablas de “gesdifran”. Un ejemplo de inserciones para la base de datos “gestdifran” lo podemos encontrar adjunto en el archivo comprimido “sgutierrezmel_producto.zip” bajo el nombre de “DDL_gestdifran.sql”.

Ya con estas inserciones se ha podido llevar a cabo todo tipo de comprobaciones para el correcto funcionamiento de la aplicación.

Capítulo 5. Valoración económica

Para realizar una valoración económica, lo primero que hay que considerar es el importe invertido en la adquisición de software y maquinaria. En este caso ha sido nulo.

Por otro lado, cabría considerar las horas invertidas en la realización del proyecto. Esto no sería muy objetivo, puesto que gran parte del tiempo lo he dedicado a adquirir conocimientos de todas las tecnologías integradas, por lo que es un atrevimiento por mi parte hacer una valoración, ni tan siquiera aproximada, del coste de este proyecto.

Capítulo 6. Conclusiones

6.1 Conclusiones personales

Desde el punto de vista personal, el proyecto ha constituido un gran reto para mí.

El proyecto final de carrera es la culminación de un largo trabajo, años y años de esfuerzo traducidos en una asignatura en la que se deben poner en práctica todos los conocimientos adquiridos.

Aun teniendo en cuenta la gran variedad de TFCs a escoger, consideré oportuno hacer un proyecto basado en la plataforma J2EE para, en cierto modo, poder ampliar mis humildes conocimientos técnicos sobre alguno de sus muchos marcos de trabajo.
Estos últimos meses han implicado un gran esfuerzo personal; días enteros de duro trabajo, largas noches a base de café, pocas horas de sueño... pero todo ello ha merecido la pena con creces. Extraigo una enriquecedora experiencia de esta asignatura.

6.2 Justificación y razonamiento de las decisiones adoptadas durante el proyecto

Hablando desde mi propia experiencia y siendo completamente sincera, he de decir que cuando comencé este TFC me dio un poco de pánico al darme cuenta que nunca había trabajado con J2EE y las diferentes tecnologías que lo componen. No me quedó otra alternativa que adentrarme en ellas y tomarme su aprendizaje como un reto a corto plazo para finalmente poder desarrollar un proyecto de tal envergadura.

Tras investigar sobre la plataforma J2EE y sus diferentes tecnologías decidí utilizar el marco de trabajo de aplicación Web Apache Struts 2, ya que es completamente novedoso y una tecnología punta. Éste me facilitaba una sencilla separación de tareas, en capas, implementando un patrón de diseño Modelo-Vista-Controlador(MVC).

Mediante Internet he recopilado muchos manuales, documentación, tutoriales y ejemplos de aplicaciones Web sencillas con Struts 2 que me han sido de mucha ayuda, sobretodo al inicio para poder realizar una buena configuración del proyecto.
Decidí también obtener el libro de Anaya Multimedia, “Struts 2”, el cual ha sido para mí un pilar imprescindible durante todo el TFC y aseguro que lo seguirá siendo.

A la hora de decidir qué Framework utilizar para la persistencia de datos en la aplicación lo tuve bastante claro, Hibernate. No fue porque ya había trabajado antes con él sino porque tenia siempre presente el poder utilizarlo, dado que había oído hablar maravillas de él y su actual importancia en el mundo de las conexiones de datos.
Integrar Hibernate en mi aplicación no me ha sido muy complicado, ya que hay muchos ejemplos en Internet que me han servido de mucha ayuda.
A la hora de mapear las entidades decidí hacerlo directamente con anotaciones y he de decir que no he encontrado ningún tipo de problema, me ha resultado bastante sencillo.

Seguidamente me tocaba decidir como iba a gestionar la creación de objetos e inyección de dependencias en dichos objetos a medida que los creaba. Tras investigar nuevamente, confirmé que esta vez el Framework Spring era ideal para esto. Aunque Spring tiene múltiples usos, el más popular es la manera de facilitar al desarrollador la creación de objetos e inyección de dependencias. Integrar este Framework con Struts 2 ha sido una satisfacción y me ha hecho las cosas más fáciles. Definitivamente, pienso que es un complemento ideal en mi aplicación y no lo substituiría.

A la hora de escoger la tecnología idónea para la vista de presentación de la aplicación me he decantado por JSP ya que era la más fácil de entender. No obstante, también decidí utilizar Tiles para crear plantillas personalizadas. Tenía en mente que la aplicación se dividiría en una cabecera estática donde mostrar el logotipo de la empresa, un menú para mostrar las diferentes funcionalidades disponibles, un cuerpo dinámico donde se irían visualizando las pantallas correspondientes a cada función y un pie de página estático. El plugin Tiles me ha resultado perfecto y a la vez muy sencillo para realizar las plantillas de la aplicación.

Las hojas de estilo en cascada externas han ido perfectas para establecer un estilo personalizado a la aplicación y a la vez separarlo de la página jsp. Dado que muchas de las pantallas tenían el mismo formato (título, tablas, botones, etc.), lo único que he tenido que hacer es crear clases de estilos y asociarlas a los elementos correspondientes en las jsp.

JavaScript me ha resultado de mucha ayuda a la hora de realizar comprobaciones en la parte cliente de la aplicación. Sobretodo lo he utilizado para recoger valores seleccionados en las tablas de los listados y éstos guardarlos en variables que serían enviadas a las acciones para seguidamente realizar operaciones con ellas. Dado que hay muchísima documentación en Internet sobre este lenguaje, no he tenido ningún tipo de problema en hacer uso de él.

Con jQuery he realizado únicamente la validación de campos para un nuevo usuario. He encontrado que es un método de validación más limpio y rápido que el del Interceptor Validate de Struts 2 ya que jQuery lo realiza dinámicamente y en la parte cliente.

Por otro lado, la generación de PDFs con iText ha resultado muy sencilla y me ha satisfecho bastante el resultado, aunque pienso que con un poco más de tiempo se podría mejorar muchísimo el aspecto.

Respecto a las funcionalidades, en un principio había previsto que tanto las facturas como los presupuestos se pudieran crear también de forma independiente a los clientes. De esta manera, el menú “Clientes” se llamaría “Ventas” y dentro de él habría un submenú con “Clientes”, “Facturas” y “Presupuestos”. Tras pensar en ello, me resultó que sería una funcionalidad un tanto repetitiva, ya que desde “Clientes” tan solo había que seleccionar el deseado y seguidamente presiona en “Facturas” donde se podría llevar a cabo toda la gestión de facturación sobre ese cliente. Se sobreentiende que una factura o presupuesto se realiza sobre un cliente, así que es intuitivo para el usuario ir directamente a “Clientes”.

6.3 Dificultades surgidas durante el proyecto y razonamientos seguidos para solventarlos

En un primer momento, mi decisión fue de realizar la configuración de Struts 2 mediante anotaciones, pero el hecho de no haber utilizado nunca esta tecnología me lo puso bastante difícil, ya que al ser una configuración tan novedosa no encontraba mucha información ni documentación al respecto y me tropezaba bastante. Finalmente, y con consejo de mi consultor, seguí la configuración mediante archivos XML. Éste me ha sido más fácil y me ha dado también la oportunidad de aprender algo sobre este otro lenguaje.

El mapeo de las relaciones con anotaciones en lugar de archivos XML me ha llevado algo más de tiempo, ya que no sabía exactamente cómo declararlas, pero al encontrar tanta documentación al respecto lo he logrado solventar con éxito.

Con Spring no he tenido ningún problema. Únicamente con el applicationContext.xml, al tener un error en la ruta que no me lo reconocía, pero logré darme cuenta a tiempo y todo funcionó correctamente.

Algo que todavía no entiendo y sigo haciendo comprobaciones es porqué en algunas páginas jsp no me reconoce el archivo .js donde hay definidas las funciones JavaScript. Por este problema, he tenido que integrar las funciones que necesitaba en la misma jsp en lugar de indicar la ruta a este archivo. Algo parecido me ha pasado con las validaciones en jQuery para un nuevo usuario.

6.4 Experiencias o conclusiones extraídas de este TFC

La experiencia vivida durante el desarrollo de este TFC ha sido para mí muy emocionante, provechosa y me ha motivado muchísimo.
Hasta ahora no tenía claro qué hacer una vez acabara la ITIG, ya que tirar únicamente hacia la Ingeniería del Software no me acababa de entusiasmar mucho. Necesitaba esta experiencia para darme cuenta lo que de verdad me gusta y se me da mejor, que es sin duda el desarrollo de aplicaciones Web.
Dado el tiempo tan limitado que hemos tenido para llevar a cabo este proyecto y teniendo presente la inexperiencia que tenia en J2EE, ya que nunca había utilizado ninguna de sus tecnologías, ni tan siquiera JavaScript, es todo un logro el haber finalizado (aunque todavía le faltan mucho retoques y mejoras) esta aplicación tan satisfactoriamente y con una arquitectura tan completa como es la integración de varias tecnologías, cada una aportando lo mejor de ellas y complementándolas entre sí.

Si tuviera que volver a hacer el proyecto, ahora que ya tengo grandes conocimientos de Struts 2, Spring para la inyección de dependencias e Hibernate, haría las pantallas de listados de forma diferente. Adquiriría conocimientos sobre jQuery y Json para poder implementar un JQGrid de manera que desde él se pudiera llevar a cabo muchas de las funciones sobre los listados.

Y sobretodo modificaría la forma en la que se crea y modifica una factura o presupuesto. Actualmente, por ejemplo, cuando se añaden los productos para una nueva factura éstos al mismo tiempo se van creando en la base de datos, de tal manera que si se cancela la operación desde el botón “Cancelar” éstos productos se eliminan de la base de datos pero si se presiona en alguna otra función, por ejemplo en “Obras”, estos productos permanecerán en la base de datos y no estarán asociados a ninguna factura. Y algo parecido sucede a la hora de modificar una factura o presupuesto. Por lo tanto, esto habría que corregirlo y una manera de hacerlo sería creando y/o eliminando los productos de manera temporal en una lista de productos para cuando se acepte finalmente la operación poder persistirlos en la base de datos. Así, nos curamos en que si se presiona sobre cualquier otra opción mientras estamos creando o modificando una factura o presupuesto, estos productos se hayan creado en la base de datos sin motivo.
Esta funcionalidad querría haberla corregido antes de entregar el proyecto pero me faltó tiempo. No obstante, ya que la aplicación servirá para la vida real tengo el compromiso de hacerlo.

Capítulo 7. Lineas futuras

Hasta ahora se han descrito todas las funcionalidades principales que en un primer momento la empresa Serralleria Difran necesita para llevar a cabo la gestión de facturación de su empresa.
No obstante, existen un añadido de funcionalidades que formarán parte de la aplicación en un futuro próximo.

Algunos de los clientes de la empresa realizan obras, que éstas a su vez tienen contratos asignados con partidas de trabajos a realizar en ella. Así que, será necesario disponer de una gestión más amplia de la facturación donde podamos gestionar los presupuestos y facturas por contratos de obras.

Está previsto que se puedan enviar por correo electrónico a los clientes tanto las facturas como los presupuestos.

Seria interesante poder almacenar todos los datos relativos a los trabajadores de la empresa, tales como sus contratos de trabajo, documentos de identidad, partes de trabajo, nóminas, etc.

Por último, sería de gran interés para la empresa el poder visualizar e imprimir tanto estadísticas de facturación como informes detallados.

Todo esto, conlleva a un abasto mayor del proyecto que será estudiado en un futuro próximo cuando la empresa Serralleria Difran lo solicite.

He de mencionar también que hay un seguido de funcionalidades que por tener tiempo limitado para realizar este proyecto no las he podido llevar a cabo.
En un principio, tenia pensado que los listados se visualizaran a través de un filtro donde el usuario pudiera escoger entre varias opciones (por ejemplo, en clientes poder visualizar únicamente los activos o inactivos, en facturas poder visualizar las de un mes y año concreto, etc.). Este añadido de funcionalidad me conllevaba a adquirir conocimientos más extensos sobre Ajax, Json, jQuery ya que rellenaban las tablas de forma dinámica sin tener que pasar la petición por el servidor.
Otra manera de hacerlo era mostrando un JQGrid de jQuery en la que ya tenía funcionalidades tales como añadir, eliminar, editar o filtrar. Ésta era perfecta para las pantallas de listados e incluso tenía un diseño más novedoso. Tras investigar y realizar pruebas finalmente no puede llevarlo a cabo ya que necesitaba más conocimientos sobre Json y jQuery que por falta de tiempo no podía adquirirlos.
Por lo tanto, esta perfección en los listados es una corrección futura de la aplicación, en la que primeramente se deberán adquirir los conocimientos oportunos y realizar pruebas iniciales.

Respecto a las validaciones, está previsto de realizarlas todas con jQuery, tal y como se han

hecho en la creación de un nuevo usuario. De esta manera, no se tendrá que enviar al servidor la petición cada vez que un campo sea incorrecto, la validación se hará en el cliente y dinámicamente.

Otro aspecto que me ha quedado pendiente, pero no por ello de menor importancia, ha sido el de controlar la seguridad de la aplicación mediante SSL. Éste no ha sido posible llevarlo a cabo por falta de tiempo y porque he tenido que priorizar otras tareas antes que esta para tener finalizada la aplicación a tiempo.

Y, por último, habría que controlar los intentos al introducir la contraseña en el acceso a la aplicación. Con esto, se establecería un máximo de tres intentos para introducir incorrectamente los datos de acceso y automáticamente el usuario quedar inactivado, teniendo que recurrir a un administrador del sistema para resolver el problema.

Capítulo 8. Glosario de términos

Actor Especifica un rol jugado por un usuario o cualquier otro sistema que interactúa con el sujeto.

Administrador Usuario del sistema con privilegios especiales encargado de gestionar la empresa y los usuarios, a parte de la gestión de facturación.

Cliente Persona física o jurídica que percibe los servicios de la empresa.

Empleado Usuario del sistema con privilegios limitados sobre el sistema, encargado de gestionar los clientes y las obras.

Factura Documento mercantil que refleja toda la información de una operación de compraventa.

GestDifran Se trata de la aplicación que implementa el proyecto.

Obra Construcción de una edificación.

Presupuesto Cómputo anticipado del coste de la compra de un producto o de un servicio.

Sistema La aplicación GestDifran.

Subsistema Cada uno de los componentes funcionales del sistema.

Usuario Persona que se conecta al sistema. Puede hacerlo con el rol de Administrador o Empleado.

Bean JavaBean, es un componente hecho en software que se puede reutilizar y que puede ser manipulado visualmente por una herramienta de programación en lenguaje Java.

Framework En español Marco de trabajo, son un conjunto de librerías que ayudan al desarrollo rápido de aplicaciones.

XML Acrónimo de Extensible Markup Language, metalenguaje extensible de etiquetas desarrollado por World Wide Web Consortium (W3C).

MVC Patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de negocio en tres componentes distintos.

Plugin Programa que puede anexar-se a otro para aumentar sus funcionalidades (generalmente sin afectar otras funciones ni afectar la aplicación principal).

Capítulo 9. Bibliografia

Donald Brown; Chad Michael Davis; Scott Stanlick, *Struts 2*, Madrid, Anaya Multimedia, 2009, Manning.

http://hop2croft.wordpress.com/2011/06/23/dont-repeat-the-dao-no-repitas-el-dao/

http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=desarrolloRapidoJava

http://www.genbetadev.com/java-j2ee/spring-framework-el-patron-dao-ii

http://www.mkyong.com/tutorials/struts-2-tutorials/

http://www.javatutoriales.com/2011/06/struts-2-parte-1-configuracion.html

http://mundogeek.net/archivos/2009/02/13/etiquetas-struts-2/

http://www.uv.es/grimo/teaching/SpringMVCv3PasoAPaso/index.html

http://docs.jboss.org/hibernate/core/3.6/reference/es-ES/html_single/#session-configuration

http://es.scribd.com/doc/454457/tutorialHibernate

http://www.slideshare.net/Emmerson_Miranda/hibernate-32-short-manual-9367150

http://www.dosideas.com/wiki/Consultas_Con_HibernateTemplate

http://www.htmlquick.com/es/tutorials.html

http://crismattweb.tripod.com/guiahtml/etiqueta.html

http://www.aprende-gratis.com/css/curso.php?lec=clases-identificadores

http://www.manualdecss.com/codigos-css/indice-codigos.html

http://java.dzone.com/articles/struts2-tutorial-part-47

http://www.roseindia.net/struts/struts/struts2.2.1/createpdf.html

http://mysticalpotato.wordpress.com/2010/12/13/plugin-de-jquery-ui-para-struts2-struts2-jquery

http://theproc.es/2011/2/9/13066/comenzando-con-jquery--tutorial-basico-para-comenzar-a-trabajar-con-jquery--parte-1-de-4-

http://www.um.es/atica/documentos/forja/js/jqueryPlugins/tablesorter/doc/ejemplo.html

http://codigojavaoracle.com/javascript/jquery-tablesorter-ii/

http://docs.jquery.com/Plugins/Validation

http://www.webtaller.com/construccion/lenguajes/index/javascript/

http://www.desarrolloweb.com/javascript/

http://www.bloogie.es/tecnologia/programacion/20-como-obtener-valores-de-un-formulario-con-javascript

http://www.findjar.com/index.x