2012

Proyecto Final de Carrera.
Jonatan Garcia Fernandez

MEMORIA

Contenido

Contenido 2

1. Resumen 5

2. Motivación y objetivos del proyecto 5

3. Propuesta técnica 6

3.1 Propuesta inicial 6

3.2 Área de conexión 6

3.3 Área de mantenimiento 7

3.4 Área de la agenda de los empleados 7

3.5 Área de facturación 7

3.6 Área de la explotación de la información 7

4. Evolución del proyecto 8

5. Planificación 8

6. Desarrollo y resultados del proyecto 10

6.1 Fase 1. Especificación y alcance de los requerimientos de la aplicación 10

6.2 Fase 2. Análisis. 10

6.3 Fase 3. Diseño de la arquitectura, modelo de datos y lógica de negocio 10

6.4 Fase 4. Diseño de la estructura de la aplicación 11

6.5 Fase 5. Implementación 11

6.6 Fase 6. Testeo y solución de posibles BUGS detectados en las pruebas. 11

6.7 Fase 7. Creación de la memoria y presentación del proyecto 11

6.8 Resultados 11

7. Análisis 11

7.1 Modelos de casos de uso y actores del sistema 11

7.2 Fichas de caso de uso 12

7.3 Prototipos de pantallas 16

7.3.1 Pantalla Menu Principal 17

7.3.2 Pantalla de Alta Clientes 17

7.3.3 Pantalla de Alta Proveedores 18

7.3.4 Pantalla de Alta Servicios 18

7.3.5 Pantalla de Alta Empleados 19

7.3.6 Pantalla de Login Administrador 19

7.3.7 Pantalla de Configuración de la agenda 19

7.3.8 Pantalla de Agenda 20

7.3.9 Pantalla de Entrada de movimientos 21

7.3.10 Pantalla de Alta Factura 21

7.3.11 Pantalla de consulta de facturas 22

8. Diseño 22

8.1 Diseño de clases 22

8.1.1 Clases de entidad 22

8.2 Diagrama de clases 23

8.3 Diagrama de casos de uso de toda la aplicación 24

8.4 Diagramas de secuencia 25

8.5 Diagrama entidad relación de la base de datos 27

9. Arquitectura del proyecto 28

9.1 ¿Qué es J2EE? ¿Qué nos aporta J2EE versus a Java tradicional? 29

9.2 Servidor de aplicaciones JBOSS. 29

9.3 Servidor web Apache Tomcat 29

9.4 Patrón de diseño MVC (Modelo, vista y controlador) 29

9.5 Framework Hibernate 30

9.5.1 Ventajas. 30

9.5.2 Inconvenientes. 30

9.6 Framework Struts2 30

9.7 Tiles 31

9.8 Patrón Data Access Object (DAO) 31

9.9 Internacialización i18n 31

9.10 Base de datos 31

10. Implementación 32

10.1 Herramientas de desarrollo 32

10.2 Decisiones. Ejemplo de creación del mantenimiento de Clientes 33

10.3 Estructura del proyecto 38

11. Manual de instalación 40

11.1 Requerimientos del sistema 40

11.2 Preparar entorno de ejecución 40

12. Conclusiones 41

13. Links y referencias de consulta 42

13.1 Hibernate 42

13.2 Struts 2 42

1. Resumen

De entre las diferentes áreas para realizar el proyecto, mi elección ha sido el área de aplicaciones web J2EE. Los motivos por lo que me he decantado por elegir esta área para el desarrollo del proyecto final de carrera han sido de orden de mayor a menor prioridad los siguientes motivos:

- *Gran interés en conocer la tecnología para ampliar mis conocimientos en el mundo de la programación a nivel profesional. Cada día más empresas utilizan esta tecnología y esto es proporcionalmente directo a ofertas de trabajo.*
- *J2EE es una tecnología que simplifica el desarrollo de aplicaciones informáticas basándose en la multicapa.*

Una vez seleccionada el área restaba plantear una temática que encajase con mis expectativas así como con los objetivos didácticos exigidos por el área J2EE.
El desarrollo del proyecto está enfocado en la gestión de una Peluquería, aunque cabe destacar que se podría implantar en cualquier otro sector de servicios. Los objetivos son disponer de una aplicación que permita mantener diferentes entidades: empleados, proveedores, clientes, servicios y productos con el objetivo de poder crear facturas a los clientes por los servicios ofrecidos, así como disponer de una agenda para poder realizar reserva de hora a los clientes bajo petición expresa de los mismos.

En este mismo documento encontraremos todo aquello referente a la memoria del proyecto PeluSoft.

2. Motivación y objetivos del proyecto

EL proyecto de final de carrera es la cúspide de la carrera de Ingeniería Técnica en informática de gestión. En ella se engloba y se pone en práctica todos los conocimientos adquiridos en las diferentes asignaturas cursadas durante el transcurso de toda la carrera.

En este proyecto se ponen en prácticas trabajos tales como la definición de requerimientos, análisis, implementación y test, dicho en otras palabras se ponen en prácticas todas las fases necesarias para el desarrollo de un proyecto en el mundo real.

Mi gran motivación personal era ampliar mis pocos conocimientos en el mundo Java así como en la inicialización de aplicaciones web con tecnología multicapa que te ofrece la plataforma de desarrollo J2EE. A parte de ser una motivación, también lo considero como un gran reto ya que la curva de aprendizaje ha sido exponencial y en muy poco tiempo. Además de motivación y de un reto también sentía curiosidad, ya que en el mundo empresarial se utiliza mucho esta plataforma de desarrollo por algo debía ser que yo desconocía, ahora una vez finalizado el proyecto creo que empiezo a entender el porqué.

Los objetivos planificados era conseguir una primera versión estable de una aplicación gestión de peluquerías, digo una primera versión ya que en un futuro se podría ampliar para adquirir más práctica con los diferentes frameworks o incluso nuevos.

3. Propuesta técnica

En este apartado se presenta la primera propuesta técnica que se consensuo con el consultor de la asignatura.

3.1 Propuesta inicial

Diseño y desarrollo de una aplicación cliente/servidor para el sector de las peluquerías.

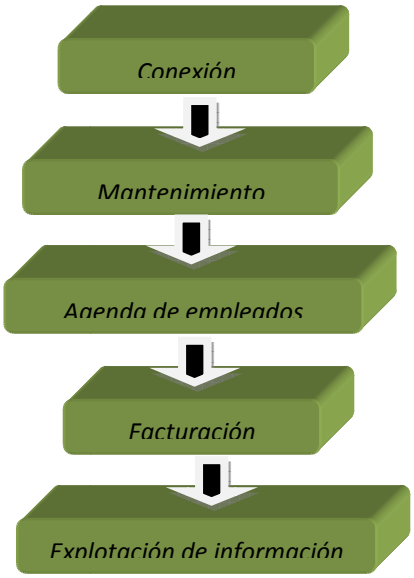
La principal funcionalidad de la aplicación es la de poder crear facturas de los servicios o artículos ofrecidos a sus cliente.

A continuación daremos una visión general de las principales funcionalidades del producto final.

Intentaremos englobar estas funcionalidades en diferentes áreas, nunca aisladas, sino que relacionadas entre ellas.

- ***Área de conexión***
- ***Área de mantenimiento***
- ***Área de agenda de empleados***
- ***Área de facturación***
- ***Área de explotación de información***

La interconexión entre estas áreas podría seguir el siguiente esquema aunque evidentemente no tiene porque seguir este orden de ejecución, exceptuando la conexión al sistema que si debe ir en primer lugar.



3.2 Área de conexión

Para poder acceder a la aplicación es necesario disponer de un usuario y un password, previamente creados por un usuario con el rol de administrador. Existirán 2 tipos de usuario el usuario administrador

y el usuario empleado, el administrador a diferencia del empleado es que podrá gestionar al resto de usuarios.

3.3 Área de mantenimiento

Toda aplicación necesita de unos datos maestros para poder realizar operaciones con ellos. En nuestro caso se mantendrá la siguiente información:

- *Empleados*
- *Clientes*
- *Artículos*
- *Servicios*

La aplicación de gestión de peluquerías pretende gestionar los servicios ofrecidos a un cliente registrándolos en el sistema.

Los usuarios podrán gestionar los clientes, productos y servicios con el fin de registrar el servicio ofrecido al cliente.

3.4 Área de la agenda de los empleados

Los empleados serán los encargados de registrar todos los servicios con los clientes así como de gestionar los stocks de los productos realizando para ello una entrada de movimiento para abastecer el stock del producto.

Una peluquería puede disponer de 1 empleado o n empleados y cada uno de ellos llevar su propia agenda. En esta área trataremos de gestionar la agenda de los empleados para conocer que disponibilidad tienen para atender a los clientes.

3.5 Área de facturación

Todo lo anteriormente no tendría sentido si la aplicación no pudiese registrar los servicios ofrecidos al cliente.

La aplicación deberá permitir a los empleados seleccionar entre los diferentes productos y servicios dados de alta, los utilizados para realizar el servicio del cliente y contabilizar el importe total para poder cobrarlo.

3.6 Área de la explotación de la información

Toda aplicación debe poder disponer de unos mecanismos para poder explotar la información que se va produciendo en la aplicación. Esta información se puede extraer ya sea por consultas por pantalla o por listados.

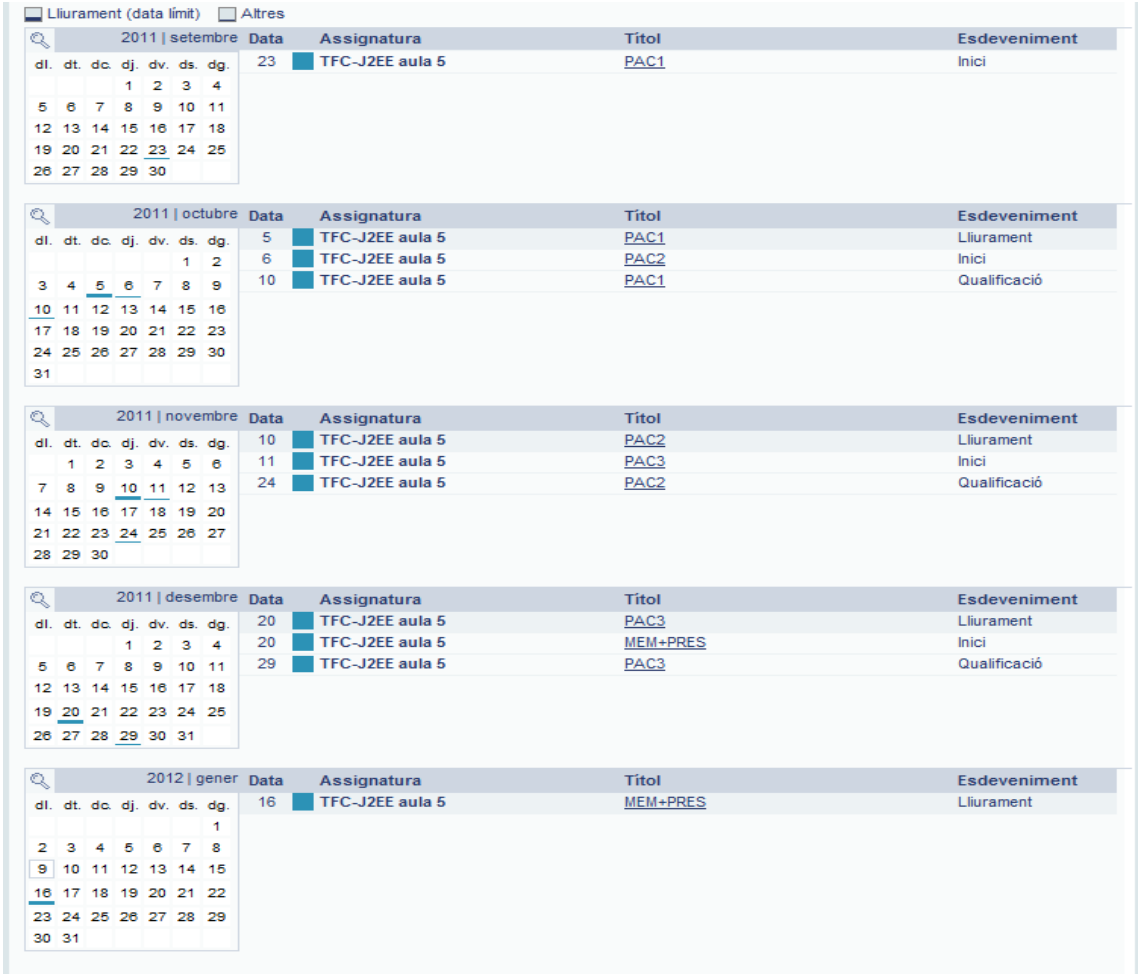
4. Evolución del proyecto

Como todo software se ha de seguir una metodología para conseguir las metas marcadas, en el caso de este proyecto se han seguido las siguientes fases, en el orden indicado:

- *Especificación y alcance de los requerimientos de la aplicación.*
- *Análisis.*
- *Diseño de la arquitectura, modelo de datos y lógica de negocio.*
- *Diseño de la estructura de la aplicación.*
- *Implementación.*
- *Testeo y solución de posibles BUGS detectado en las pruebas.*

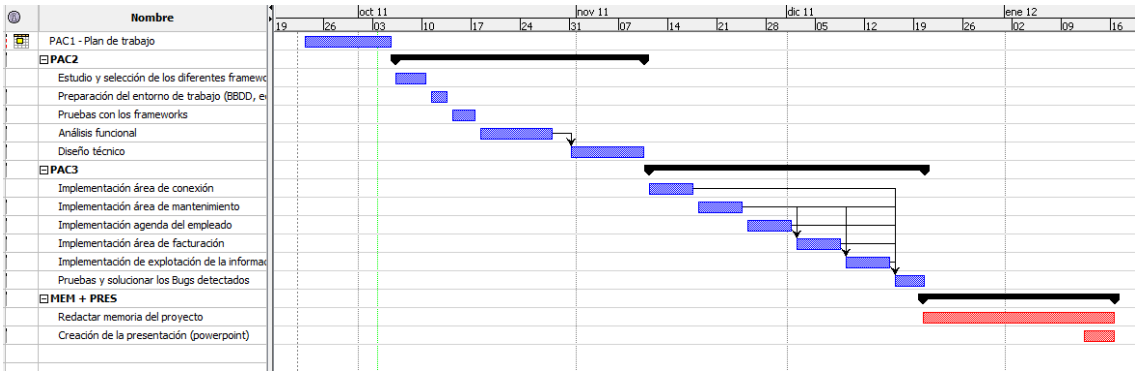
5. Planificación

Al principio de la asignatura se estipulo el siguiente calendario empezando el TFC el día 23-09-2011 y finalizando el día 16-01-2012 un total de un poco menos de 4 meses.

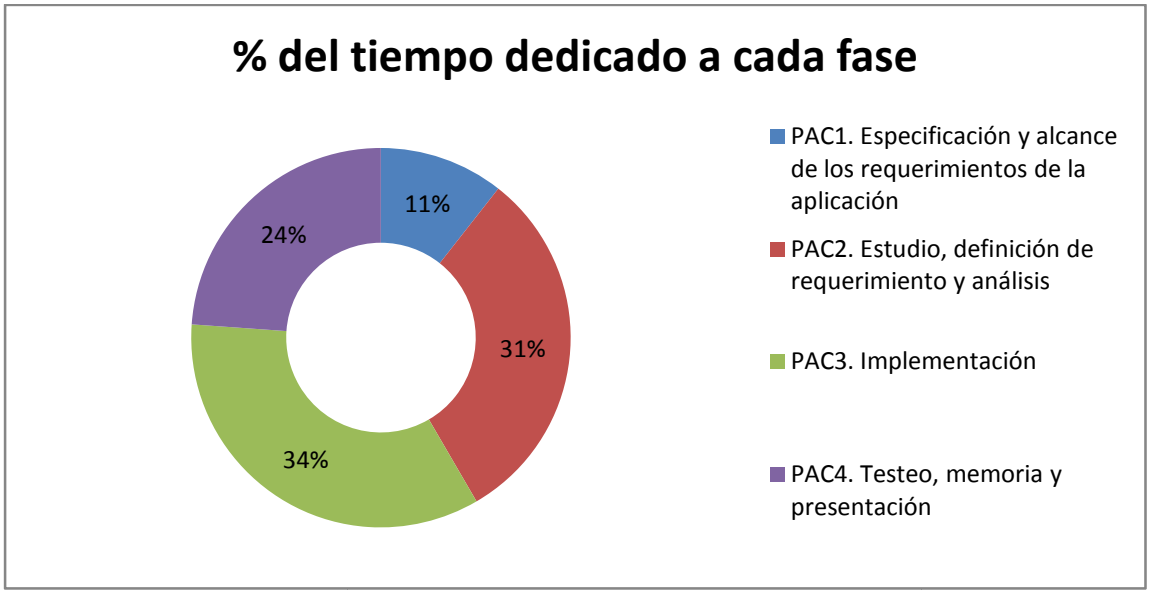


Es de vital importancia una buena planificación y más en este caso que el calendario está muy ajustado. Para ello que mejor que realizar un diagrama de grantt con todas las fases.

Nombre	Duración	Inicio	Terminado	Predecesores
PAC1 - Plan de trabajo	9 days	23/09/11 8:00	5/10/11 17:00	
PAC2	**26 days**	**6/10/11 8:00**	**10/11/11 17:00**	
Estudio y selección de los diferentes framewo	3 days	6/10/11 8:00	10/10/11 17:00	
Preparación del entorno de trabajo (BBDD, e	3 days	11/10/11 8:00	13/10/11 17:00	
Pruebas con los frameworks	2 days	14/10/11 8:00	17/10/11 17:00	
Análisis funcional	9 days	18/10/11 8:00	28/10/11 17:00	
Diseño técnico	9 days	31/10/11 8:00	10/11/11 17:00	6
PAC3	**28 days**	**11/11/11 8:00**	**20/12/11 17:00**	
Implementación área de conexión	5 days	11/11/11 8:00	17/11/11 17:00	
Implementación área de mantenimiento	5 days	18/11/11 8:00	24/11/11 17:00	
Implementación agenda del empleado	5 days	25/11/11 8:00	1/12/11 17:00	
Implementación área de facturación	5 days	2/12/11 8:00	8/12/11 17:00	10
Implementación de explotación de la informac	5 days	9/12/11 8:00	15/12/11 17:00	10
Pruebas y solucionar los Bugs detectados	3 days	16/12/11 8:00	20/12/11 17:00	9;10;11;12;13
MEM + PRES	**20 days**	**20/12/11 8:00**	**16/01/12 17:00**	
Redactar memoria del proyecto	20 days	20/12/11 8:00	16/01/12 17:00	
Creación de la presentación (powerpoint)	2 days	5/01/12 8:00	6/01/12 17:00	



En la siguiente gráfica se muestra el % porcentaje dedicado en cada fase.



6. Desarrollo y resultados del proyecto

Según la planificación y consensuadas las fases del desarrollo de software sólo cabía esperar si la evolución y resultados eran los esperados en cada fase. En este capítulo se explicara con claridad fase por fase lo esperado y lo realmente conseguido, consiguiendo de esta forma tener claros que se esperaba y que se ha conseguido.

A modo resumen y como apunte general se ha conseguido el 100% de lo esperado.

6.1 Fase 1. Especificación y alcance de los requerimientos de la aplicación

En esta fase el objetivo era definir el alcance del proyecto, así como la definición de lo que debía hacer la aplicación junto con sus limitaciones. Se cumplió el objetivo, además en esta fase se comento la tecnología que se iba a poner en práctica para desarrollar la aplicación web.

6.2 Fase 2. Análisis.

Se requería analizar la problemática, desglosando un todo en n partes más pequeñas y para cada una de estas partes analizar la situación inicial del problema y como consiguiente diseñar un documento técnico de desarrollo de dichas partes. Se cumplió el objetivo.

6.3 Fase 3. Diseño de la arquitectura, modelo de datos y lógica de negocio

Esta fase tiene una relación muy directa con la fase 2, además de diseñar y selección de la base de datos. En esta fase también se puso en práctica la utilización de los diferentes frameworks del mercado. Se cumplió el objetivo

6.4 Fase 4. Diseño de la estructura de la aplicación

¿Como se organizará la aplicación, cuantas pantallas existirán? A todas estas preguntas y más se les da respuesta en esta fase.

6.5 Fase 5. Implementación

Esta es la fase de programación de la aplicación, en la que se pone en práctica todo lo analizado anteriormente. Es una fase crítica del proyecto, ya que se puede haber realizado perfectamente las fases anteriores pero si esta fase no se finaliza por completo el fracaso estará asegurado. Cabe la posibilidad de que en esta fase se pueda realizar correcciones de las fases previas. Se cumplió el objetivo.

6.6 Fase 6. Testeo y solución de posibles BUGS detectados en las pruebas.

Una vez desarrollado el producto, se debe realizar un mínimo de pruebas unitarias y de cadena, por regla general siempre suelen aparecer algún error (BUGS), esta fase es la destinada para detectar todos los posibles problemas antes de pasar el producto a producción. Es una fase importante en el ciclo de desarrollo ya que es la última oportunidad de detectar posibles problemas antes de entregarlo al cliente.

6.7 Fase 7. Creación de la memoria y presentación del proyecto

Actualmente estamos en esta fase, en ella se crean 2 documentos uno este mismo documento, y otro la presentación del proyecto (explicado en puntos anteriores de este mismo documento).

6.8 Resultados

Los productos obtenidos al finalizar el TFC son los siguientes:

- *Una **aplicación web**, que permite al usuario realizar unos mantenimientos propiamente dichos de clientes, proveedores, productos, servicios. Una entrada de stocks de los productos. Una creación de facturas con los servicios y productos utilizados en un servicio al cliente, y por último y no menos importante una gestión de la agenda de reservas de citas de los posibles clientes que desean que se les realice un servicio de peluquería en un día y hora señalado.*
- *Una **memoria**, es este mismo documento, donde se recoge todo aquello esencial relacionado con el proyecto y de vital importancia y de futura consulta en un futuro para entender el proyecto así como si se diese el caso de ampliar o mejorar el funcionamiento del mismo.*
- *Una **presentación,** es un PowerPoint que ofrece de una forma muy esquematizada una perspectiva general de lo realizado el proyecto de final de carrera.*

7. Análisis

7.1 Modelos de casos de uso y actores del sistema

*En todo software informático existen los actores. Un **actor** es un conjunto de papeles de una entidad exterior en relación el con el sistema software considerado, podemos decir que un actor es la visión que el software tiene de una entidad exterior. Un actor puede ser una persona, o incluso un proceso. En el caso de una persona, una misma puede tener el papel de más de un actor.*

*EL **caso de uso** es una documentación gráfica sobre una interacción entre el software y un actor o más, siendo esta interacción una función autónoma dentro del software.*

A continuación se detallan los diferentes actores que intervendrán en PeluSoft:

- ***Empleado:*** *Es el usuario registrado en la aplicación y actor principal en el sistema.*
- ***Administrador:*** *Es un empleado con más privilegios. Las diferentes entre administrador y empleado es que el administrador puede realizar el mantenimiento de empleados y proveedores.*

En el siguiente punto se numeran todas las funcionalidades de cada actor.

Actor	*Funcionalidad*
Empleado	*Crear/modificar/consultar y eliminar productos* *Crear/modificar/consultar y eliminar servicios* *Crear/modificar/consultar y eliminar clientes* *Crear facturas* *Recibir citas del cliente e introducirlas en el sistema* *Dar de baja y modificar citas de la agenda de cualquier empleado*
Administrador	*Las mismas que las del empleado y además:* *Crear/Consultar/modificar y eliminar empleados* *Crear/Consultar/modificar y eliminar proveedores* *Configurar la agenda*

7.2 Fichas de caso de uso

En este apartado se definen los casos de usos más relevantes de la aplicación. Se obvian los casos de uso típicos como pueden ser los de mantenimiento.

Caso de uso	*Definir cliente*
Resumen de la funcionalidad	*Da de alta una persona como cliente*
Actores	*Empleado*
Casos relacionados	*Ninguno*
Precondición	*La persona no está de alta en la aplicación como cliente*
Postcondición	*La persona está data de alta en la aplicación como cliente*
Secuencia normal *La aplicación solicita al usuario los datos del cliente.* *El usuario introduce los datos.* *El sistema valida que el cliente no esté dado de alta en el sistema y lo guarda en la base de datos.*	
Excepciones *El usuario ya está dado de alta en el sistema. Se activa el caso de uso consultar cliente.*	
Alternativa de proseo y alternativa *Ninguna*	

Caso de uso	*Consultar cliente*
Resumen de la funcionalidad	*Consulta un cliente una persona dada de alta como cliente*

Actores	*Empleado*
Casos relacionados	*Ninguno*
Precondición	*La persona está de alta en la aplicación como cliente*
Postcondición	*La aplicación muestra los datos del cliente*
Secuencia normal *La aplicación solicita al usuario los datos para realizar la consulta del cliente.* *El usuario introduce uno o varios datos que puedan identificar a un cliente.* *El sistema busca en la base de datos según los criterios informados y muestra todos los datos del cliente. Cuantos más datos se informe más se acotará la búsqueda y más rápida será.*	
Excepciones *El usuario no está dado de alta en el sistema. La aplicación no devolverá ningún dato. Se activa el caso de uso Definir cliente.*	
Alternativa de proseo y alternativa *Ninguna*	

Caso de uso	***Actualizar cliente***
Resumen de la funcionalidad	*Modifica los datos de un cliente*
Actores	*Empleado*
Casos relacionados	*Consultar cliente*
Precondición	*La persona está de alta en la aplicación como cliente*
Postcondición	*El cliente está actualizado con los nuevos datos informados.*
Secuencia normal *Ejecutar la secuencia normal del caso de uso Consultar cliente.* *El usuario introduce aquellos datos que desea ampliar o modificar del cliente.* *El sistema valida los datos, y los graba en la base de datos.*	
Excepciones *El usuario no está dado de alta en el sistema. La aplicación no devolverá ningún dato. Se activa el caso de uso Definir cliente.*	
Alternativa de proseo y alternativa *Ninguna*	

Caso de uso	***Seleccionar cliente***
Resumen de la funcionalidad	*Seleccionar un cliente de entre todos*
Actores	*Empleado*
Casos relacionados	*Ninguno*
Precondición	*El cliente a seleccionar deberá estar dado de alta en el sistema*
Postcondición	*Cliente seleccionado, para poder operar con él, por ejemplo para hacerle una factura.*
Secuencia normal *Introducir el código del cliente* *El sistema verifica si el código existe.* *El sistema muestra el nombre del cliente.*	
Excepciones *El usuario no está dado de alta en el sistema. La aplicación no devolverá ningún dato.*	
Alternativa de proseo y alternativa	

Ninguna

El caso de uso seleccionar es lo mismo para productos y servicios.
Para los actores empleados y proveedores se utilizarán los mismos casos de uso, la diferencia que para definir/consultar/modificar un empleado o proveedor el actor que lo puede hacer es el empleado con rol de administrador.
El caso de uso definir empleado, y definir proveedor tendrá como casos relacionados el de Login de empleado.

Caso de uso	***Login de empleado***
Resumen de la funcionalidad	*Identificarse en la aplicación como administrador para poder definir empleados o proveedores.*
Actores	*Empleado*
Casos relacionados	*Ninguno*
Precondición	*Debe existir un empleado con el check de administrador marcado.*
Postcondición	*Se habilitara la opción de definir empleados y proveedores.*
Secuencia normal *El administrador introduce el usuario Administrador (predefinido) y el password.* *El sistema detecta que el usuario es administrador y habilita las opciones de definir empleados y proveedores.*	
Excepciones *El usuario no está dado de alta en el sistema.* *El usuario está dado de alta pero no es administrador.* *El usuario está dato de alta, es administrador pero la password no es la correcta.*	
Alternativa de proseo y alternativa *Ninguna*	

Caso de uso	***Entrar una factura***
Resumen de la funcionalidad	*Crear una factura a un cliente por los servicios realizados.*
Actores	*Empleado*
Casos relacionados	*Seleccionar cliente* *Seleccionar proveedor* *Seleccionar producto* *Seleccionar servicio*
Precondición	*El empleado se ha logineado correctamente en el sistema.* *La factura no está dada de alta en el sistema.* *Debe existir el cliente, producto/s, servicio/s utilizados en la factura.*
Postcondición	*La factura se ha creado en el sistema para posteriormente cobrarle al cliente.*
Secuencia normal *El empleado introduce los datos necesarios para la creación de la factura, introduciendo una cabecera y una o varias líneas de productos y servicios utilizado en el servicio.*	
Excepciones *Para loginearse, las explicadas en el caso de uso de Login de empleado.*	

No existe el cliente, producto/s o servicio/s.
Alternativa de proseo y alternativa *Ninguna*

Caso de uso	***Imprimir una factura o tiquet***
Resumen de la funcionalidad	*Imprimir en papel una factura.*
Actores	*Empleado*
Casos relacionados	*Entrar una factura*
Precondición	*La factura está dada de alta en el sistema.* *Existe una impresora conectada al sistema.*
Postcondición	*La factura se ha imprimido.*
Secuencia normal *El empleado localiza la factura.* *El empleado imprime la factura.*	
Excepciones *Errores varios a la hora de imprimir (incidencia técnica de la impresora)*	
Alternativa de proseo y alternativa *Ninguna*	

Caso de uso	***Configurar parámetros de la agenda***
Resumen de la funcionalidad	*Configurar los parámetros de la agenda*
Actores	*Empleado*
Casos relacionados	*Login Administrador*
Precondición	*El empleado que realizará la acción debe ser administrador.*
Postcondición	*La agenda está configurada y se podrá utilizar.*
Secuencia normal *El sistema solicita que el empleado se identifica como administrador.* *El usuario introduce los datos de la configuración y guarda los cambios.*	
Excepciones *El empleado no es administrador.*	
Alternativa de proseo y alternativa *Ninguna*	

Caso de uso	***Anular cita***
Resumen de la funcionalidad	*Anular una cita previamente registrada*
Actores	*Empleado*
Casos relacionados	*Consultar agenda, definir cita*
Precondición	*La agenda está configurada y la cita está registrada.*
Postcondición	*La cita está anulada para ese día y horas. Se podrán realizar más citas para ese día y esas horas.*
Secuencia normal *El empleado recibe una petición del cliente de una anulación de un servicio.* *El empleado consulta en la agenda la cita.* *El empleado anula la cita en el sistema y se lo notifica al cliente.*	
Excepciones	

La cita no estaba registrada en el sistema, en este caso no será necesario realizar ningún acción.
Alternativa de proseo y alternativa *Ninguna*

Caso de uso	***Consultar agenda***
Resumen de la funcionalidad	*Consultar la agenda*
Actores	*Empleado*
Casos relacionados	*Ninguna*
Precondición	*La agenda está configurada*
Postcondición	*La aplicación muestra los datos de la agenda para un día en concreto.*
Secuencia normal *El empleado introduce una fecha* *El sistema muestra las citas de la agenda de todos los empleados que se muestren en la agenda.*	
Excepciones *No está configurada la agenda.*	
Alternativa de proseo y alternativa *Ninguna*	

Caso de uso	***Definir cita***
Resumen de la funcionalidad	*Crear una cita en la agenda.*
Actores	*Empleado*
Casos relacionados	*Consultar agenda*
Precondición	*La agenda está configurada.*
Postcondición	*La cita está reservada para ese día y horas. No se podrán realizar más citas para ese día y esas horas.*
Secuencia normal *El empleado recibe una petición del cliente para un servicio.* *El empleado consulta la agenda para un día en concreto para conocer la disponibilidad de todos los empleados para ese día* *El empleado registra la cita en el sistema y se lo notifica al cliente.*	
Excepciones	
Alternativa de proseo y alternativa *Ninguna*	

7.3 Prototipos de pantallas

En este apartado se presentan todos los prototipos que compondrán la aplicación PeluSoft.
Entre ellas se presentan las típicas pantallas de mantenimiento y algunas más complejas como puede ser la entrada de una factura.
Como particularidad se ha decidido obviar pantallas de consulta aprovechando para ello las mismas pantallas de mantenimiento.
Se ha decidido usar colores en los campos con el siguiente propósito:

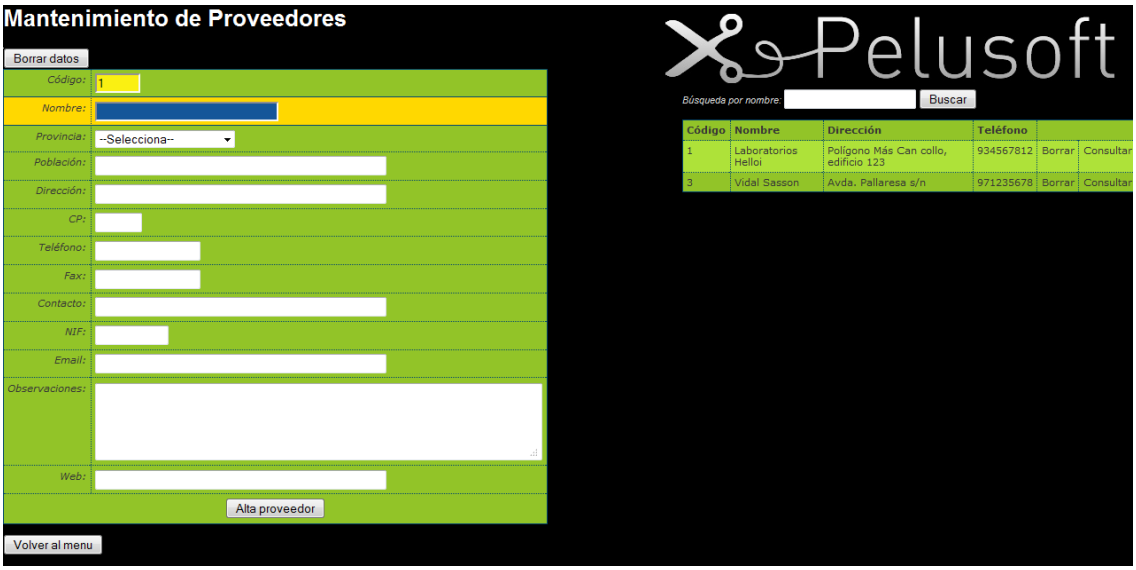
- *Color amarillo (PK): Aquellos campos que identifican inequívocamente a la entidad que representa.*
- *Color azul: Aquellos campos que son obligatorios informar para poder crear un registro.*

Para realizar las consultas para posteriormente realizar alguna acción sobre la entidad, será necesario informar el nombre de la entidad y pulsar sobre buscar y las coindicencias aparecerán en el grid de resultado (las consultas se hacen con un like, con lo cual no es necesario informar el nombre completo).

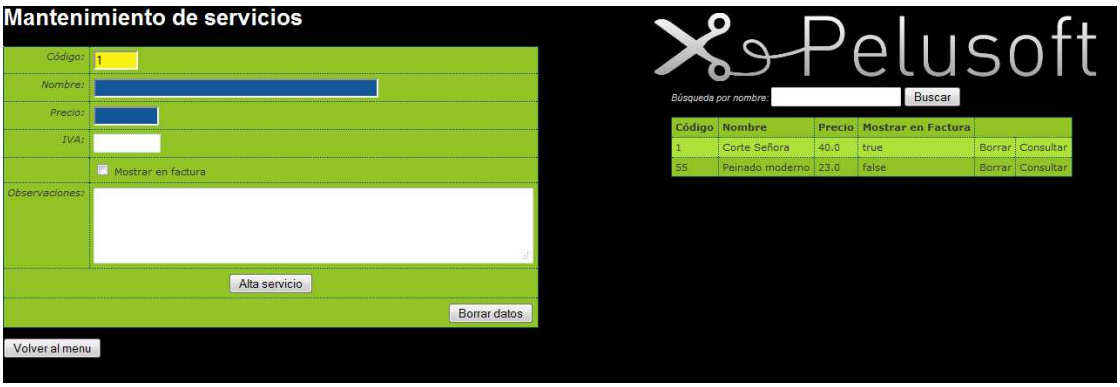
7.3.1 Pantalla Menu Principal

7.3.2 Pantalla de Alta Clientes

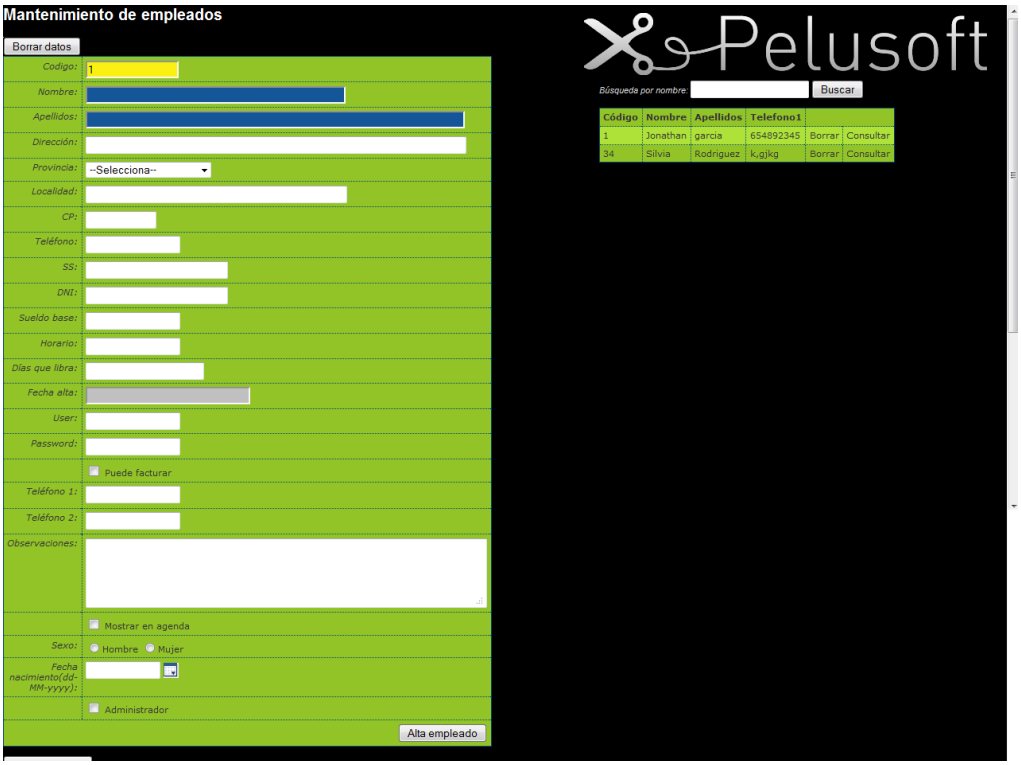
7.3.3 Pantalla de Alta Proveedores



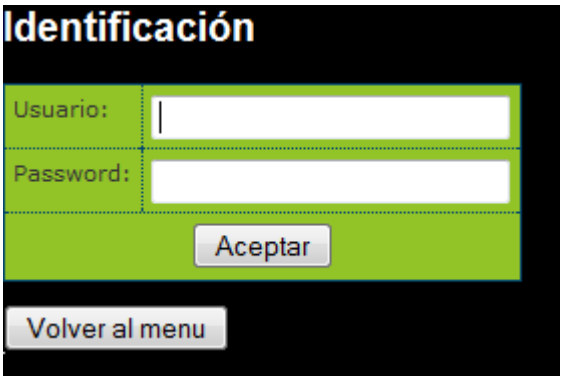
7.3.4 Pantalla de Alta Servicios



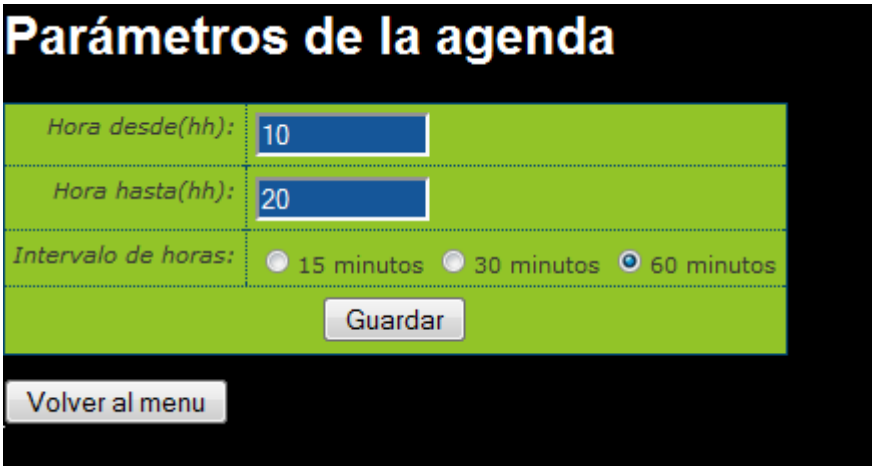
7.3.5 Pantalla de Alta Empleados



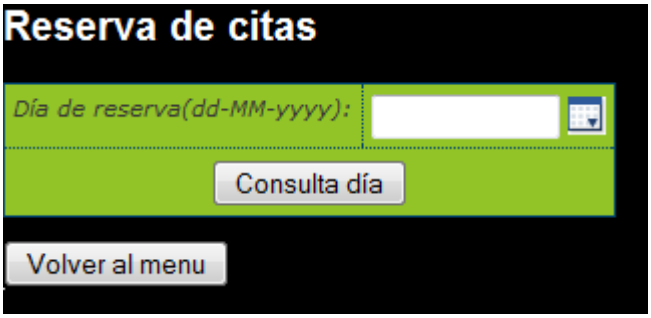
7.3.6 Pantalla de Login Administrador

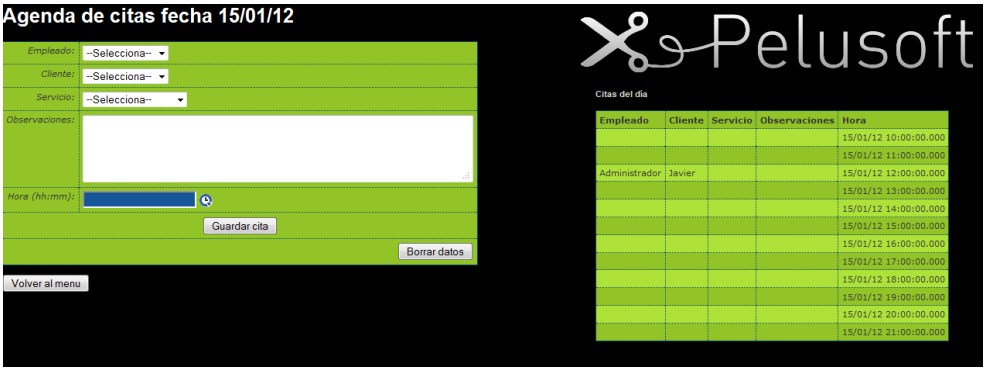


7.3.7 Pantalla de Configuración de la agenda



7.3.8 Pantalla de Agenda



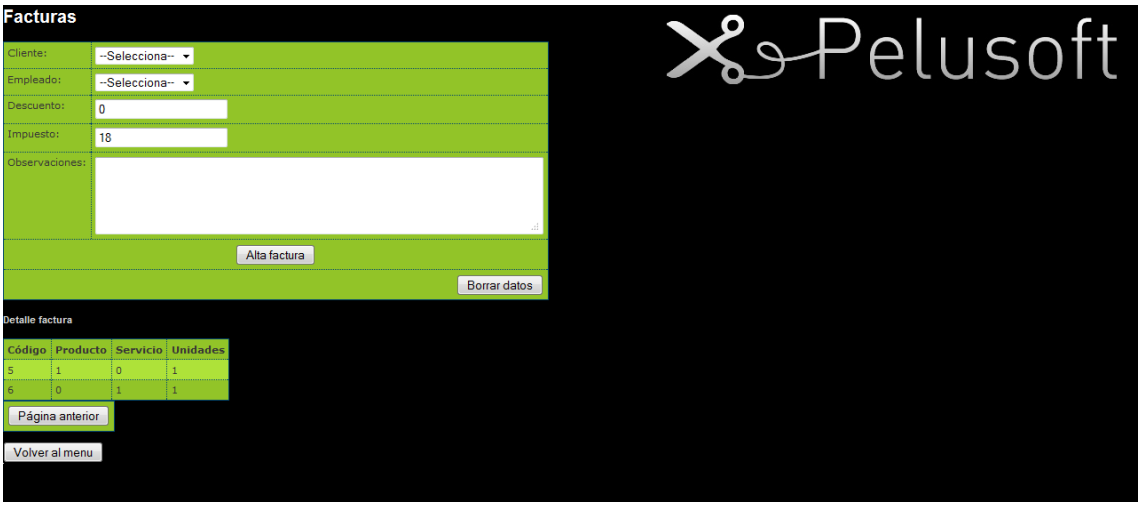


7.3.9 Pantalla de Entrada de movimientos

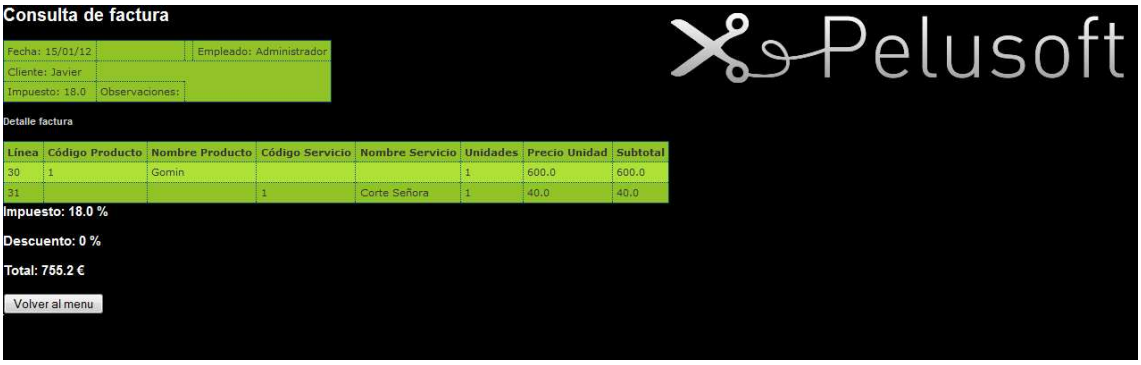
7.3.10 Pantalla de Alta Factura

7.3.10.1.1 Pantalla selección de productos y servicios

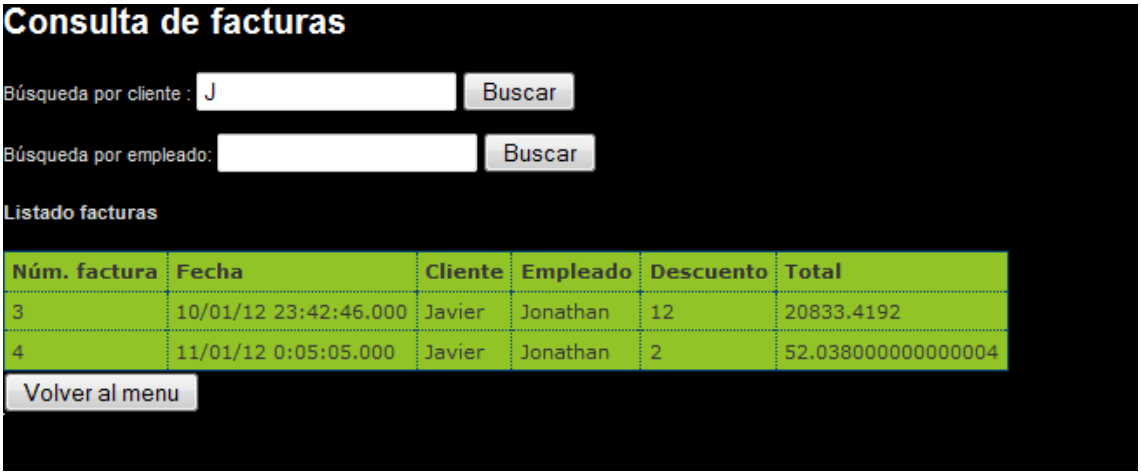
7.3.10.1.2 Pantalla selección de alta de factura



7.3.10.1.3 Pantalla selección resultante de la factura



7.3.11 Pantalla de consulta de facturas



8. Diseño

Siguiendo el ciclo de construcción de un software, una vez realizado el análisis el siguiente paso es diseñar la aplicación. En este apartado se verá el proceso de diseño de clases, el diseño del modelo relacional de la base de datos y la arquitectura empleada.

8.1 Diseño de clases

8.1.1 Clases de entidad

Para realizar un buen diseño es necesario identificar las clases entidad o beans. Las clases entidad son aquellas que no implementan ninguna interface, son clases planas con propiedades y con sus respectivos métodos getters y setters. Estas clases son estrictamente necesarias para el buen funcionamiento del mapeo objeto-relacional, de hecho son las necesaria para el mapeo, que usará el framework Hibernate para guardar el contenido de un objeto en una tabla de una base de datos relacional y al revés para cargar el contenido de una base de datos en objetos.

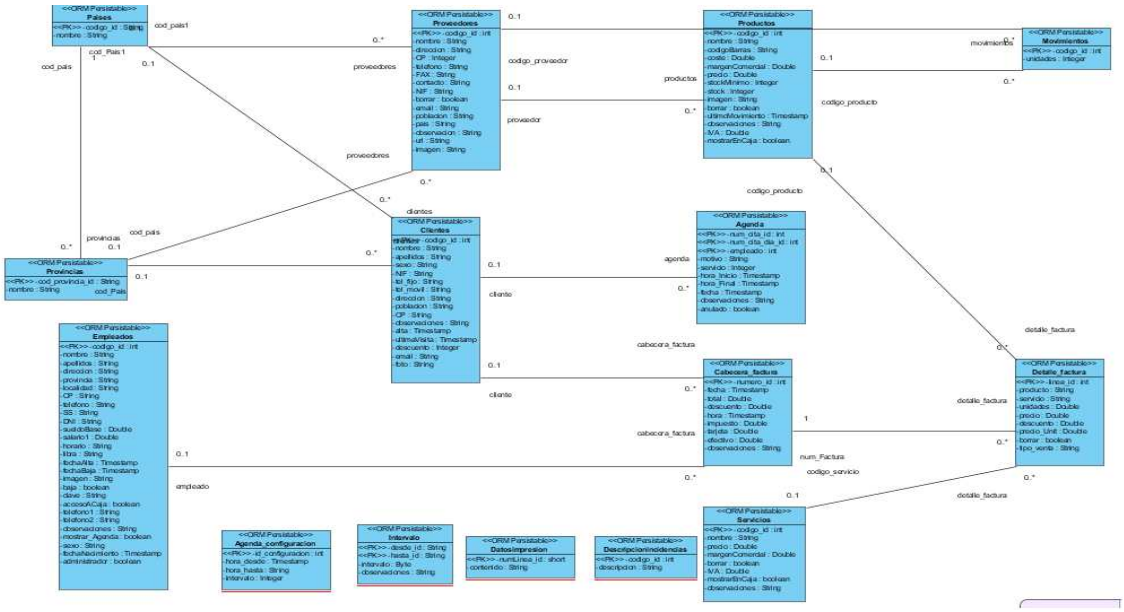
Los objetos que instancian estas clases se llaman POJO's (Plain Old Java Object).

A partir de los casos de uso se puede llegar a la siguiente lista de clases entidad:

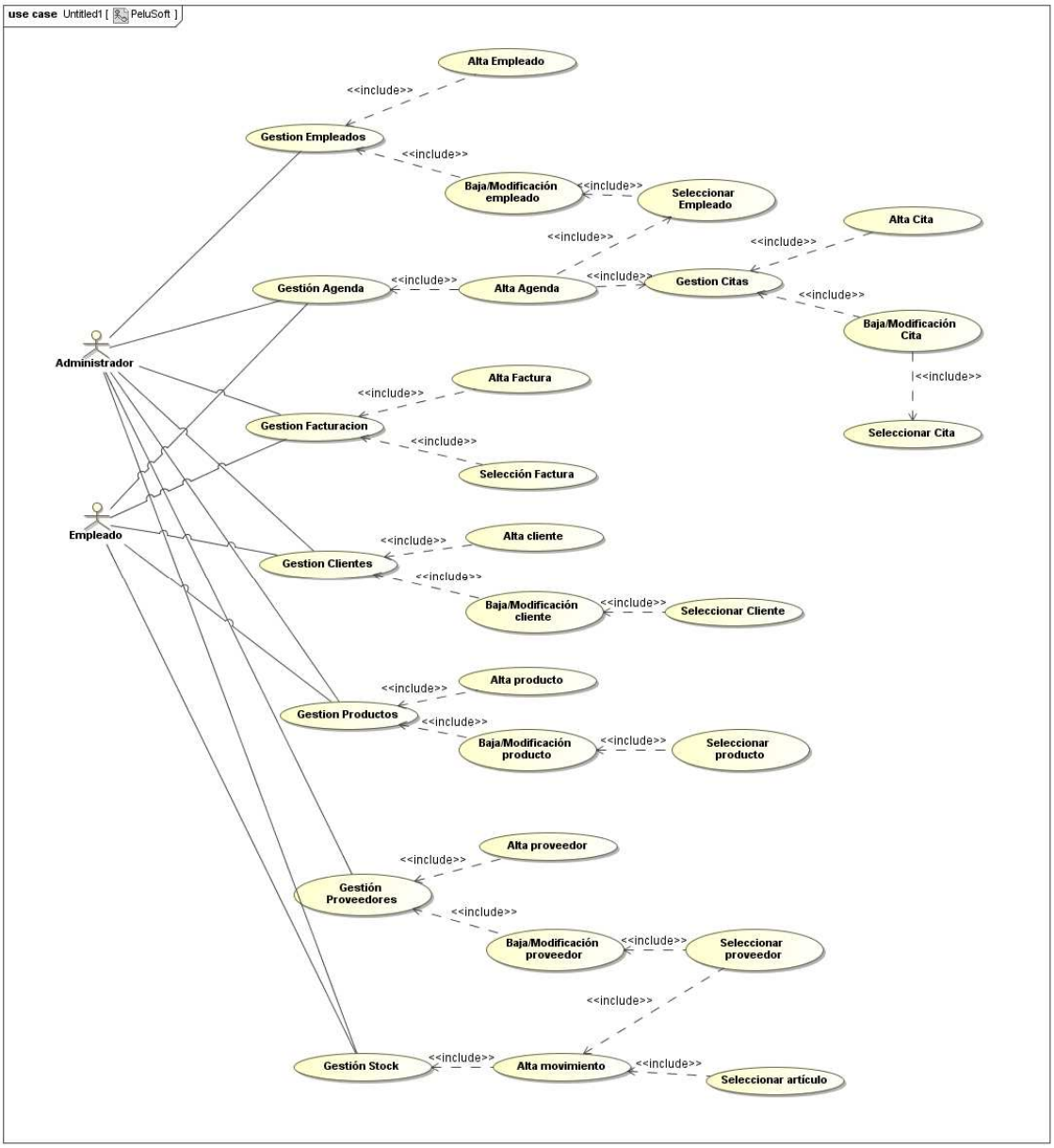
Clases Entidad
Empleado
Cliente
Proveedor
Producto
Servicio
Factura
Agenda
AgendaConfiguracion
Provincia

8.2 Diagrama de clases

En el diagrama de clases se describen todos los atributos de las clases entidad, y se diseñan las relaciones entre ellas.



8.3 Diagrama de casos de uso de toda la aplicación



8.4 Diagramas de secuencia

Se obvian el resto de diagramas de secuencia de mantenimientos por ser obvios.

Diagrama de secuencia del proceso de generación de una factura.

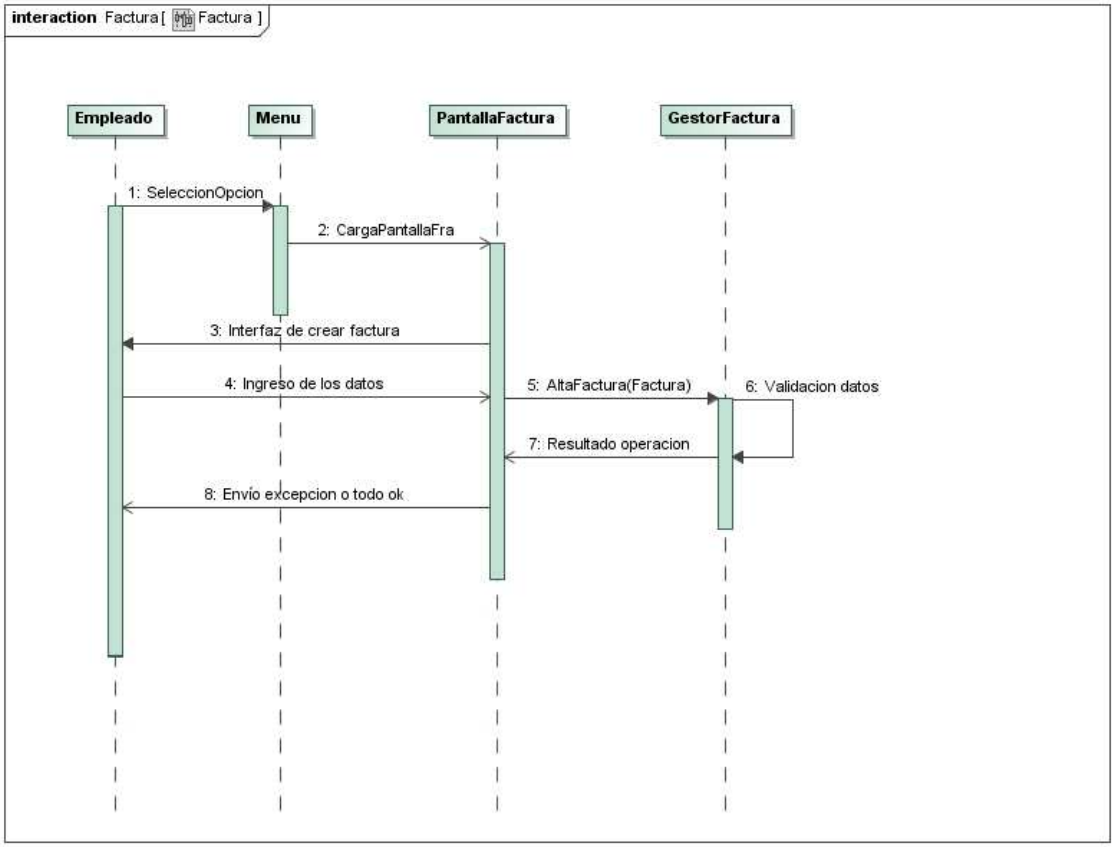
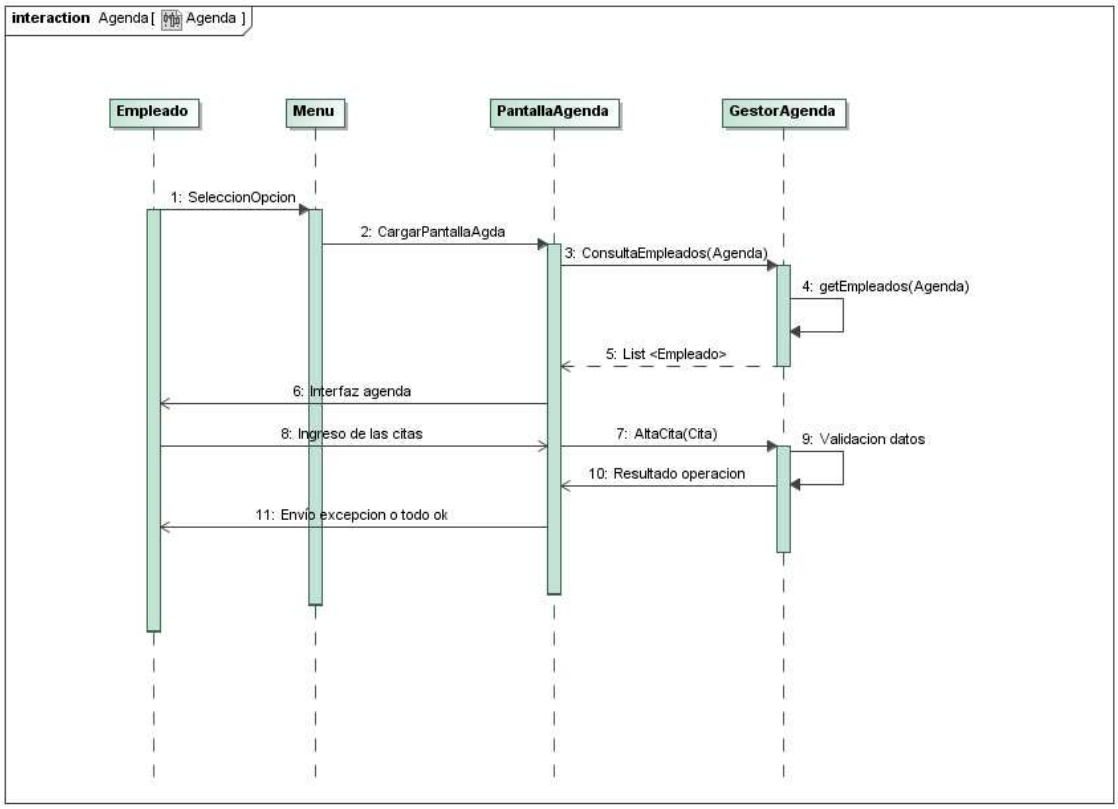
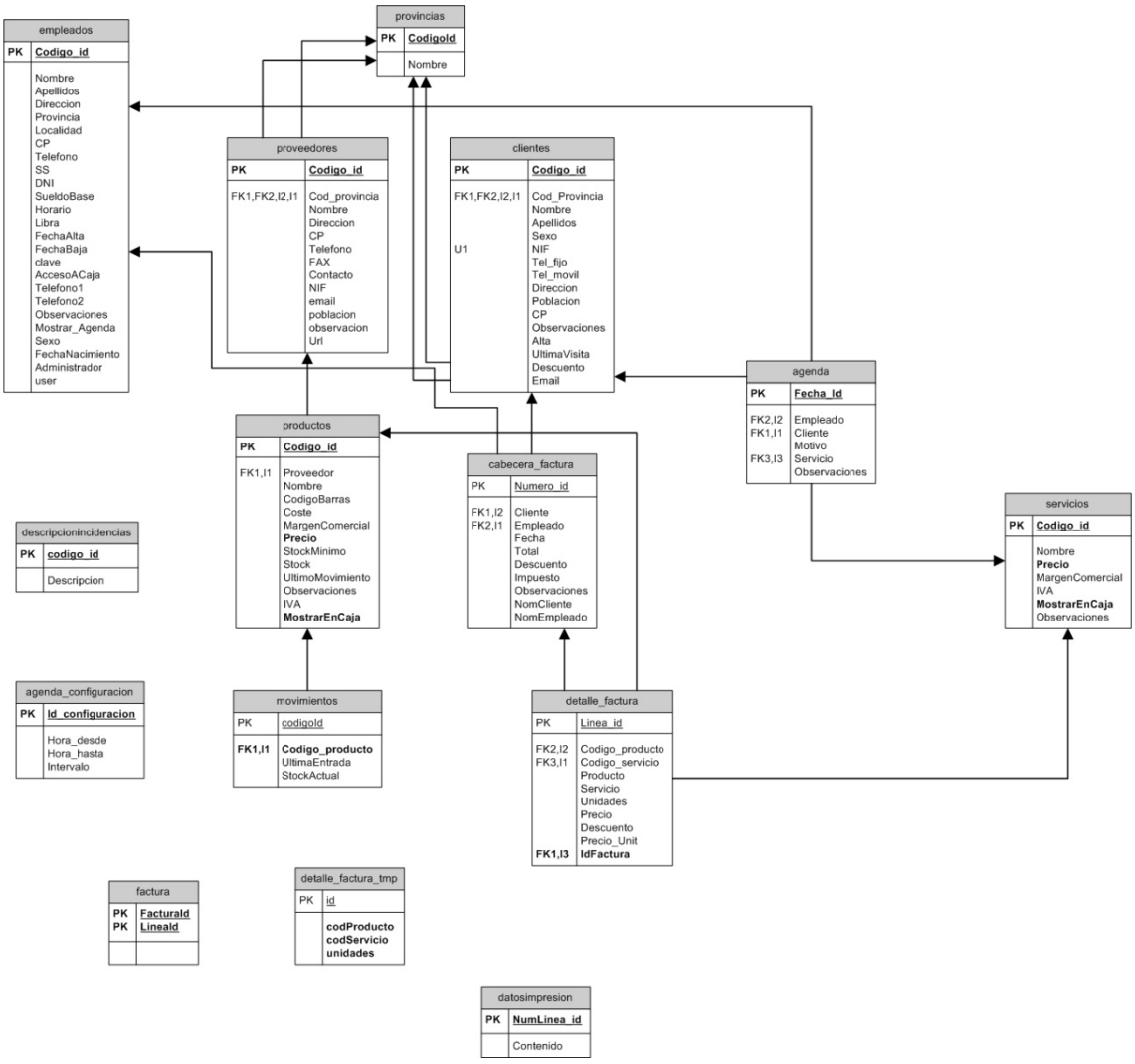


Diagrama de secuencia del proceso de introducir una cita en la agenda.



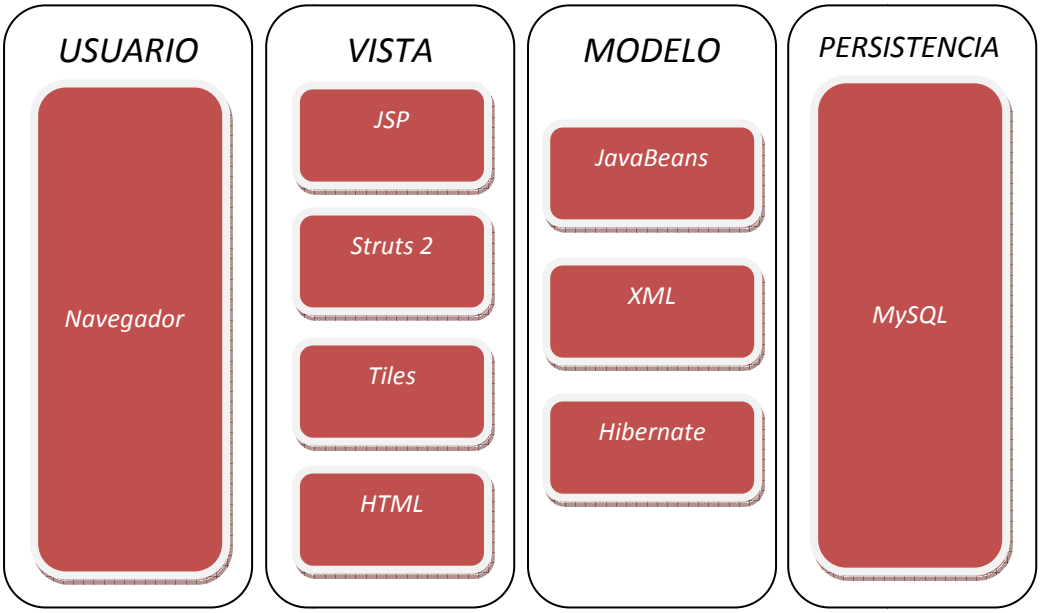
8.5 Diagrama entidad relación de la base de datos



9. Arquitectura del proyecto

Para desarrollar la aplicación PeluSoft se utilizará la tecnología J2EE, para entender en qué consiste tal tecnología, organizaré este apartado en los diferentes subapartados:

1. ***¿Qué es J2EE? ¿Qué nos aporta J2EE versus a Java tradicional?***
2. ***Servidor de aplicaciones JBOSS.***
3. ***Servidor web Apache Tomcat***
4. ***Patrón de diseño MVC (Modelo , vista y controlador)***
5. ***Framework Hibernate***
6. ***Framework Struts2***
7. ***Tiles***
8. ***Patrón Data Access Object (DAO)***
9. ***Internacionalización i18n***
10. ***Base de datos***



9.1 ¿Qué es J2EE? ¿Qué nos aporta J2EE versus a Java tradicional?

J2EE es una plataforma abierta y estándar para el desarrollo y despliegue de aplicaciones empresariales multicapa con n-niveles, distribuidas y basadas en componentes.

Las ventajas que nos ofrece J2EE:

- ***Alta disponibilidad.*** *Los servicios que nos ofrecen no pueden dejar de funcionar. Se han de proporcionar mecanismos para asegurar que no dejen de funcionar y, si lo hacen, que el tiempo de interrupción sea mínimo.*
- ***Seguros.*** *Se ha de garantizar que no hayan accesos no autorizado a los servicios y se ha de establecer políticas de acceso para los diferentes tipos de usuario de estos servicios.*
- ***Fiables.*** *Los servicios han de ser lo máximo de fiables y libres de errores. Se han de ofrecer mecanismos de detección y diagnosis de de errores.*
- ***Escalables.*** *Si aumenta la carga del sistema, ha de ser fácil añadir servidores o bien ampliar los que ya tenemos para dar la misma calidad de servicio sin tener que modificar las aplicaciones existentes.*
- ***Mantenibles****. Se ha de poder añadir servicios al sistema y modificar los existentes fácilmente.*
- ***Portables.*** *Se ha de poder cambiar de plataforma de una manera no traumática. Un cambio de plataforma no puede implicar volver a implementar todos los servicios.*
- ***Rápidos de desarrollar y de desplegar.*** *Se ha de poder desarrollar y ofrecer servicios a los usuarios del sistema de manera ágil y rápida. Hoy en día, el famoso time-to-market es vital para las empresas.*
- ***Fácilmente integrables con los sistemas existentes.*** *Raramente se desarrollarán servicios nuevos partiendo de cero. Lo normal será integrar los nuevos servicios o desarrollos ya existentes, y se ha de poder realizar fácilmente.*

9.2 Servidor de aplicaciones JBOSS.

De entre todos los servidores de Aplicaciones, JBOSS ha sido el seleccionado ya que es freeware y es unos de los más usados.

9.3 Servidor web Apache Tomcat

Como servidor web utilizaré Tomcat 6.0 de la familia Apache Software Foundation ya que soporta tanto servlets como páginas JSP.

9.4 Patrón de diseño MVC (Modelo, vista y controlador)

El patrón MVC (Modelo-Vista-Controlador), una arquitectura que busca reducir el acoplamiento dividiendo las responsabilidades en 3 capas claramente diferenciadas:

- El **modelo**, que hace referencia a los datos que maneja la aplicación y las reglas de negocio que operan sobre ellos y que se traducen en Struts 2 en las **acciones**.
- La **vista**, encargada de generar la interfaz con la que la aplicación interacciona con el usuario. En Struts 2 equivale a los resultados (**Páginas JSP**)
- El **controlador**, que comunica la vista y el modelo respondiendo a eventos generados por el usuario en la vista, invocando cambios en el modelo, y devolviendo a la vista la información del modelo necesaria para que pueda generar la respuesta adecuada para el usuario. El controlador se implementa en Struts 2 mediante el filtro `FilterDispatcher`

9.5 Framework Hibernate

Hibernate es un entorno de trabajo que tiene como objetivo facilitar la persistencia de objetos Java en bases de datos relacionales y a la vez la consulta de estas bases de datos para obtener objetos.

Dicho de otras palabras Hibernate nos proporciona una forma ágil y rápida de relacionar la base de datos con clases java, consiguiendo aislar de tal manera que da igual contra que base de datos vamos a trabajar.

Es sabido que un diagrama entidad relación no tiene porque ser igual que un diagrama de clases y de esta parte es la que se encarga Hibernate de relacionar una entidad relacional con un objeto, a través de mapeos.

9.5.1 Ventajas.

- Permite trabajar sobre base de datos relacionales haciendo servir toda la potencia de la orientación a objetos. Esta característica permite reducir la complejidad del desarrollo con los consiguientes aumentos de productividad, fiabilidad y mantenibilidad.
- Incluye numerosas mejoras de rendimiento respecto a otras soluciones como ahora la carga de datos bajo demanda (que evita cargar datos que finalmente no se consultan), la optimización trasparente de consultas SQL o el uso de una memoria cache.
- Permite cambiar de dialectos SQL (y, por lo tanto, de fabricante de bases de datos) sin haber de modificar el código.

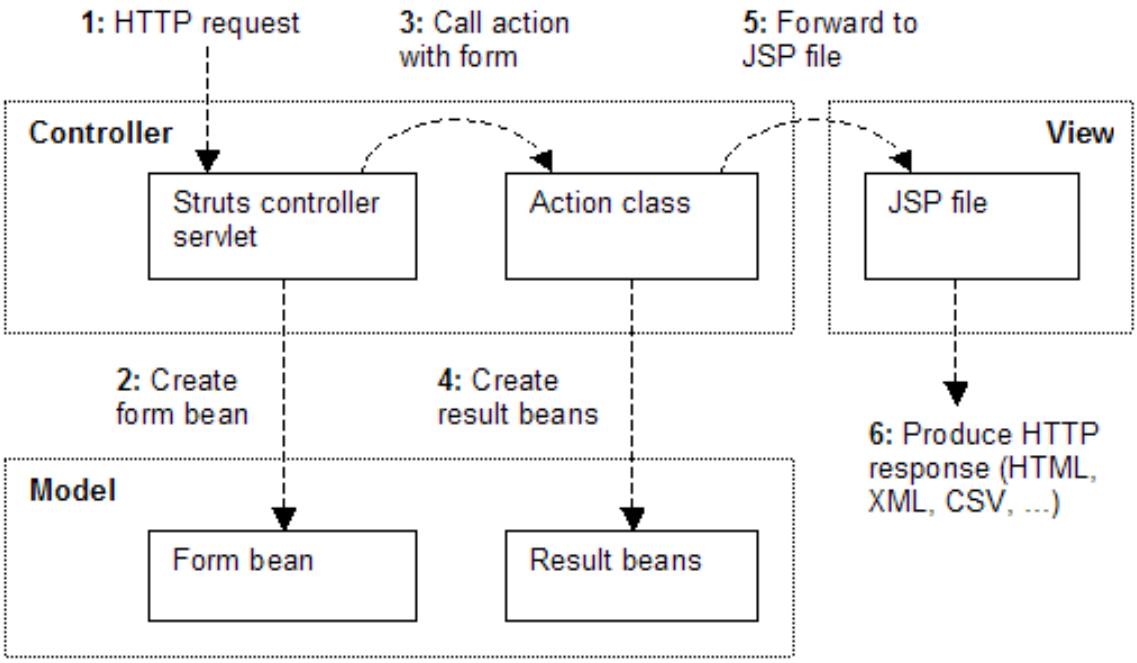
9.5.2 Inconvenientes.

El principal inconveniente de Hibernate es la necesidad de un proceso inicial de aprendizaje tanto por lo que es el modelo de programación como la interface específica que nos ofrece.

9.6 Framework Struts2

El framewrok Struts 2 está basado en el patrón MVC (Modelo-Vista-Controlador).

Diagrama de flujo de control de Struts2.



9.7 Tiles

Es un framework de Apache que facilita la construcción de interficies de usuario en aplicaciones web, ayudando de esta forma a desarrollar con más facilidad la capa de presentación de la plataforma J2EE. Este framework nos permite tener partes de la web común como son la cabecera y el pie, y otra dinámica que sería el cuerpo de la página web.

9.8 Patrón Data Access Object (DAO)

El patrón DAO se ocupa de almacenar y recoger datos de una base de datos. Utilizando este patrón hacemos que nuestra aplicación sea lo más independiente posible de una base de datos en concreto, de cómo se accede a los datos o incluso de si hay o no base de datos detrás. Nuestra aplicación debe conseguir los datos o ser capaz de guardarlos en algún sitio, pero no tiene por qué saber de dónde los está sacando o dónde se guardan.

9.9 Internacialización i18n

La aplicación soportará más de un idioma gracias al servicio que nos otorga la internacionalización i18n a través de que los literales de la aplicación están escritos en ficheros de propiedades. De esta forma simplemente leyendo un fichero u otro el idioma de la aplicación cambia sin necesidad de tocar el código fuente de la aplicación.

9.10 Base de datos

El sistema gestor de base de datos utilizado para guardar los datos de la aplicación (persistencia) será MySQL.

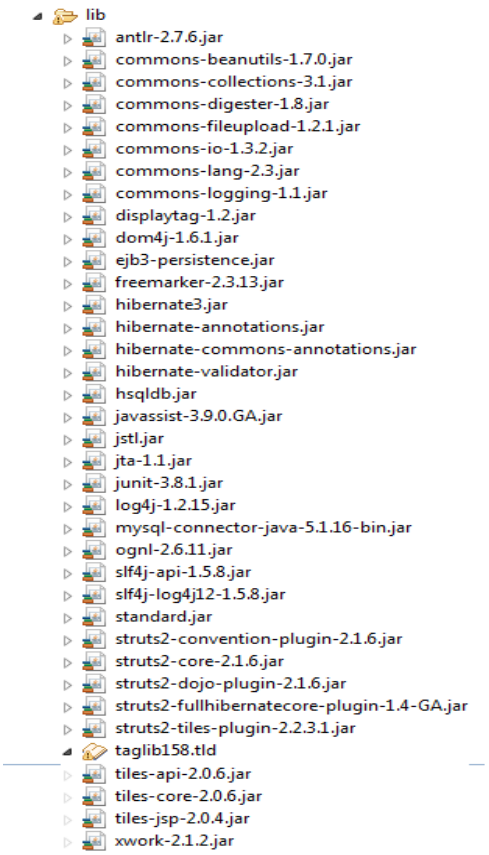
10. Implementación

La fase de implementación es la fase más práctica de todas. En ella se pone entredicho todo lo aprendido y todo lo analizado anteriormente para construir la aplicación.

10.1 Herramientas de desarrollo

Las herramientas utilizadas en todas las fases del desarrollo de la aplicación:

- ***Eclipse Helios:*** *Es el entorno de desarrollo IDE escogido para realizar la programación en entorno Java. Para trabajar con este entorno y con los diferentes frameworks es necesario descargarse una sería de librerías.*



- ***JDK 6.0 (Java Development Kit):*** *Las herramientas necesarias de programación es la JDK en la que se incluye el JRE (Java Runtime Evironment).*
- ***Apache Tomcat v 6.0:*** *Servidor web*
- ***JBOSS 5.1:*** *Servidor de aplicaciones*

- ***Navegador web:*** *Mozilla Firefox versión 8.0.1*
- ***MySQL 1.2.17:*** *El sistema gestor de base de datos seleccionado es MySQL.*
- ***Microsoft Office Visio 2003:*** *Software utilizado para crear los diagramas de clases entidad y el diagrama de clases UML.*
- ***Pencil:*** *En la fase de diseño, realicé el enmaquetado de las pantallas. Pencil fue la aplicación escogida ya que la había utilizado en otras asignaturas (TDP) de la carrera.*
- ***Open Project:*** *Aplicación que se usa para crear diagramas de Grantt para hacer un seguimiento de cumplimientos de fechas.*
- ***Magic Draw UML:*** *Para realizar los diferentes diagramas UML.*

10.2 Decisiones. Ejemplo de creación del mantenimiento de Clientes

En este capítulo se explica el patrón de diseño MVC (Modelo, vista, controlador) utilizado para la creación de los mantenimientos de tablas maestras (Empleado, Cliente, Proveedor, Servicio …).

Para hacer más entendible este capítulo me basaré en explicar el desarrollo necesario para el mantenimiento de clientes.

A continuación se presenta la estructura multicapa. En la capa modelo disponemos de las clases entidad con sus atributos y sus métodos getters y setters.

```
import java.util.Date;
import java.util.HashSet;
import java.util.Set;

/**
 * Clientes generated by hbm2java
 */
@SuppressWarnings("serial")
public class Cliente implements java.io.Serializable {

    private int codigoId=1;
    public Provincia provincias;
    private String nombre;
    private String apellidos;
    private String sexo;
    private String nif;
    private String telFijo;
    private String telMovil;
    private String direccion;
    private String poblacion;
    private String cp;
    private String observaciones;
    private Date alta;
    private Date ultimaVisita;
    private Integer descuento;
    private String email;
    private Set<Agenda> agendas = new HashSet<Agenda>(0);
    private Set<Factura> cabeceraFacturas = new HashSet<Factura>(
            0);

    public int getCodigoId() {
        return this.codigoId;
    }

    public void setCodigoId(int codigoId) {
        this.codigoId = codigoId;
    }
```

…. El contenido del fichero es más grande pero no es necesario ponerlo todo en la memoria.

En la capa controlador disponemos de diferentes clases para trabajar con la lógica de negocio.

***ClienteAction.java**. El contenido del fichero es más grande pero no es necesario ponerlo todo en la memoria.*

```
package actions;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;

import com.opensymphony.xwork2.ActionContext;
import com.opensymphony.xwork2.ActionSupport;
import com.opensymphony.xwork2.ModelDriven;

import modelos.Cliente;
import modelos.Provincia;
import dao.ClienteDAO;
import dao.ClienteDAOImpl;
import dao.ProvinciaDAO;
import dao.ProvinciaDAOImpl;


@SuppressWarnings("serial")
public class ClienteAction extends ActionSupport implements ModelDriven<Cliente> {

    private ClienteDAO clienteDAO = new ClienteDAOImpl();
    private Cliente cliente= new Cliente();
    private List<Cliente> clienteList = new ArrayList<Cliente>();
    private ProvinciaDAO provinciaDAO = new ProvinciaDAOImpl();
    private List<Provincia> provinciaList = new ArrayList<Provincia>();
    private String namesearch;

    public String getName() {
        return namesearch;
    }

    public void setName(String name) {
        this.namesearch = name;
    }

    public Cliente getModel() {
        return cliente;
    }
```

ClienteAddAction.java

```
package actions;
import com.opensymphony.xwork2.ActionSupport;
import com.opensymphony.xwork2.ModelDriven;
import modelos.Cliente;
import dao.ClienteDAO;
import dao.ClienteDAOImpl;

@SuppressWarnings("serial")
public class ClienteAddAction extends ActionSupport implements ModelDriven<Cliente> {

    private ClienteDAO clienteDAO = new ClienteDAOImpl();
    private Cliente cliente= new Cliente();

    public Cliente getModel() {
        return cliente;
    }

    public String addCliente()
    {
        boolean resultado=false;
        resultado=clienteDAO.saveCliente(cliente);
        if (resultado==true) {
            return SUCCESS;
        } else {
            addActionError("No ha sido posible guardar el cliente. Verifice si el nif pertenece a otro cliente.");
            return INPUT;
        }
    }

    public Cliente getCliente(){
        return cliente;
    }

    public void setCliente(Cliente cliente) {
        this.cliente  = cliente;
    }

}
```

ClienteDAO.java

```
package dao;

import java.util.List;

import modelos.Cliente;

public interface ClienteDAO {

    public boolean saveCliente(Cliente cliente);
    public List<Cliente> listClientes();
    public List<Cliente> listByName(String name);
    public boolean deleteCliente(Cliente cliente);
    public Cliente loadCliente(int idCliente);
}
```

ClienteDAOImpl.java

```
package dao;

import java.util.List;

import org.hibernate.Query;
import org.hibernate.Session;
import org.hibernate.Transaction;

import modelos.Cliente;
import modelos.Provincia;

public class ClienteDAOImpl implements ClienteDAO{
    private Transaction transaction;


    public boolean saveCliente(Cliente cliente) {

        Session session = HibernateUtil.getSessionFactory().openSession();
        transaction=session.beginTransaction();
        try {
            Provincia provincia = (Provincia)session.get(Provincia.class, cliente.getProvincias().getId());
            cliente.setProvincias(provincia);
            //Guardar la fecha actual como fecha de alta si esta vacía
            if (cliente.getAlta()==null){
                java.util.Date date = new java.util.Date();
                cliente.setAlta(date);
            }
            session.saveOrUpdate(cliente);
            transaction.commit();
            return true;
        } catch (Exception e) {
            transaction.rollback();
            e.printStackTrace();
            return false;
        }
    }
```

Y por último en la capa Vista tenemos las páginas jsp, ya sea para pedir información al usuario o para mostrarle resultados, digamos que sería la capa de comunicación con el usuario.

*Todas estas 3 capas se han de comunicar de una forma. Como he comentado anteriormente disponemos de una clase entidad llamada Cliente (**Modelo**) y a su vez del fichero Cliente.hbm.xml en el que se indica el mapeo entre objeto y tabla relacional para que el framework Hibernate sepa ligar un atributo de la clase con un campo de la tabla. En la siguiente presento un ejemplo del fichero Cliente.hbm.xml el fichero es más grande pero considero que no es necesario incluirlo en este documento.*

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!-- Generated 10-nov-2011 0:07:43 by Hibernate Tools 3.4.0.CR1 -->
<hibernate-mapping>
    <class name="modelos.Cliente" table="clientes" catalog="peluqueria">
        <id name="codigoId" type="int">
            <column name="Codigo_id" />
            <generator class="assigned" />
        </id>
        <many-to-one name="provincias" class="modelos.Provincia" update="true" insert="true" fetch="select">
            <column name="Cod_Provincia" />
        </many-to-one>
        <property name="nombre" type="string">
            <column name="Nombre" length="30" />
        </property>
        <property name="apellidos" type="string">
            <column name="Apellidos" length="50" />
        </property>
        <property name="sexo" type="string">
            <column name="Sexo" length="10" />
        </property>
        <property name="nif" type="string">
            <column name="NIF" length="25" />
        </property>
        <property name="telFijo" type="string">
            <column name="Tel_fijo" length="13" />
        </property>
        <property name="telMovil" type="string">
            <column name="Tel_movil" length="13" />
        </property>
        <property name="direccion" type="string">
            <column name="Direccion" length="100" />
        </property>
        <property name="poblacion" type="string">
            <column name="Poblacion" length="30" />
        </property>
```

*Por otro lado tenemos una página JSP (**Vista**) para interactuar con el usuario de esta forma el usuario podrá manipular la información del Cliente, consultando, borrando y creando clientes en la aplicación. Esta página JSP dispara o ejecuta una acción cuando se pulsa sobre el botón SUBMIT y aquí es donde tiene lugar la capa **Controlador** a través del framework Struts2 en la que debe existir un fichero llamado struts.xml en el que se describen las acciones que se deben realizar.*

```
<s:form action="addCustomer" method="post" validate="true" onReset="location.href='Cliente'">
```

Struts.xml

```
<!-- ACCIONES SOBRE CLIENTES -->
    <action name="addCustomer" class="actions.ClienteAddAction" method="addCliente">
        <result name="success" type="redirect">listCustomer</result>
        <result name="input">/Cliente.jsp</result>
    </action>
    <action name="listCustomer" method="listClientes" class="actions.ClienteAction">
        <result name="success">/Cliente.jsp</result>
        <result name="input">/Cliente.jsp</result>
    </action>
    <action name="deleteCustomer" method="deleteCliente" class="actions.ClienteAction">
        <result name="success" type="redirect">listCustomer</result>
        <result name="input">/Cliente.jsp</result>
    </action>
    <action name="loadCustomer" class="actions.ClienteAction" method="loadCliente">
        <result name="success">/Cliente.jsp</result>
        <result name="input">/cliente.tiles</result>
    </action>
    <action name="Cliente" class="actions.ClienteAction" method="loadDates">
        <result name="success">/Cliente.jsp</result>
        <result name="input">/Cliente.jsp</result>
    </action>
    <action name="searchbynameCliente" method="listClientesbyName" class="actions.ClienteAction">
        <result name="success">/Cliente.jsp</result>
        <result name="input">/Cliente.jsp</result>
    </action>
```

Para validar la información introducida por el usuario Strut2 proporciona un mecanismo muy cómodo de realizar dichas validaciones. Para realizar estas validaciones es necesario crear un archivo xml con el mismo nombre de la clase java que se encarga de manipular la acción, más el literal "-validation.xml" es decir es necesario crear el fichero ClienteAddAction-validation.xml y dentro de él se deben crear tantos tags como campos deseamos validar y definir las validaciones que deseamos realizar, por ejemplo se suele usar para verificar que un atributo que sea obligatorio se informe o por ejemplo para validar que un tipo de dato sea el esperado.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE validators PUBLIC          "-//OpenSymphony Group//XWork Validator 1.0.2//EN" "http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">
<validators>
    <field name="Cliente.codigoId">
        <field-validator type="required">
            <message key="errors.required" />
        </field-validator>
    </field>
    <field name="Cliente.nombre">
        <field-validator type="requiredstring">
            <message key="errors.required" />
        </field-validator>
        <field-validator type="stringlength">
            <param name="maxLength">30</param>
            <message>Se ha sobrepasado el tamaño máximo del campo. Tamaño máximo 30</message>
        </field-validator>
    </field>
    <field name="Cliente.apellidos">
        <field-validator type="requiredstring">
            <message key="errors.required" />
    </field-validator>
        <field-validator type="stringlength">
            <param name="maxLength">50</param>
            <message>Se ha sobrepasado el tamaño máximo del campo. Tamaño máximo 50</message>
        </field-validator>
    </field>
    <field name="Cliente.sexo">
        <field-validator type="required">
            <message key="errors.required" />
        </field-validator>
        <field-validator type="int">
            <message key="errors.invalid" />
        </field-validator>
        <!-- Valores posibles v o m. Debera ser Numerico 1: Hombre 2: Mujer-->
        <field-validator type="regex" short-circuit="true">
            <param name="expression">[1-2]</param>
            <message>Valores posibles 1 o 2</message>
        </field-validator>
```

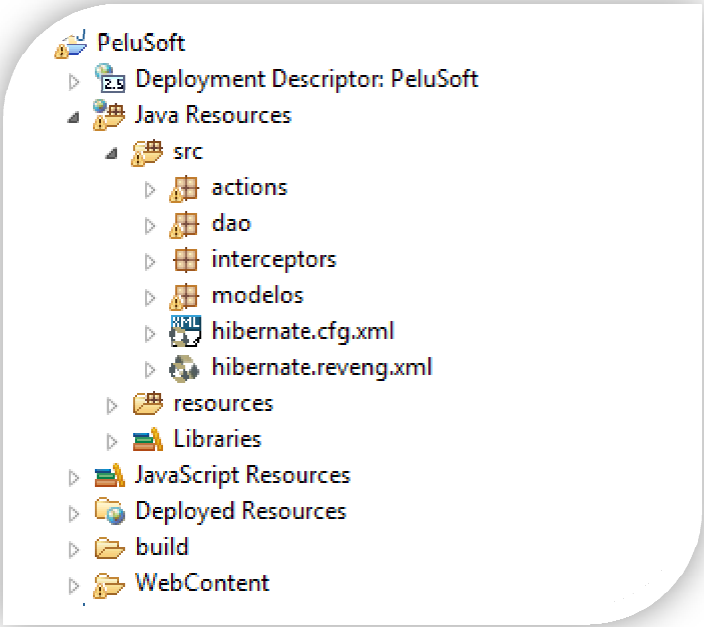
Si la validación de los datos no es correcta se redirige a la misma página jsp, por lo contrario si es correcta los datos informados Struts2 redirige la ejecución del programa al método y clase definidos en el fichero xml según la acción invocada desde la página JSP, en el caso de una creación del cliente la clase controladora sería ClienteAddAction.java, y esta a su vez invoca al método add de la clase ClienteDAOImpl.java que usa la interface ClienteDAO.java y el método utilizando la tecnología que ofrece el framework Hibernate graba el registro en la base de datos.

Para eliminar un cliente, listar clientes, consulta cliente el proceso muy parecido lo único que difiere de la alta es que la clase controladora no es ClienteAddAction.java sino que es ClienteAction.java y que en vez de llamar al método Add se llama al método correspondiente según la acción a realizar.

No hay que olvidar los interceptores, que no es más que un control que ofrece Struts2 que es de gran utilidad en el proyecto PeluSoft para verificar que se ha configurado la agenda antes de intentar reservar una cita. Los interceptores son clases que se ejecutan antes de que se ejecute otra acción.

10.3 Estructura del proyecto

Este es el aspecto del explorador de aplicaciones de Eclipse.



En la siguiente tabla se muestra los packages junto con el contenido de cada uno:

Package	***Contenido***
Src/modelos	*En este paquete están todas las clases entidad.*
Src/dao	*En este paquete hay 2 tipos de contenidos:* • ***Interficies****: Clases de java que interpretan una interface con las funciones básicas (delete, list, load, ...) que se pueden realizar sobre una clase entidad. Sus nombre son nombre de la entidad con la que trabaja + "DAO". Ejemplo: ClienteDAO.java.* • ***Clases DAOImpl****: Clases que implementan las interficies anteriormente explicadas. Sus nombres son nombre de la interficie que implementa + Impl. Ejemplo: ClienteDAOImpl.java.*

Src/actions	*En este paquete hay 2 tipos de contenidos:* • ***Clases actions**: Clases actions controladoras. Sus nombres son nombre de la entidad con la que trabaja + "Action". Ejemplo: ClienteAction.java y ClienteAddAction.java* • ***Ficheros xml de validación**: En estos ficheros están definidas las validaciones que se deben realizar en los datos introducidos por el usuario. La aplicación verifica las validaciones y si todo va bien sigue ejecutando la aplicación.*
Src/interceptors	*En la aplicación sólo se ha implementado un interceptor, para verificar que antes de reservar una cita se comprueba si laos parámetros de la agenda se han informado.*
Resources	*En esta carpeta se ha decidido guardar el fichero AppplicationResources.properties donde están todos los textos en español de la aplicación ya que se usa i18n.*
WebContent	*Aquí es donde están guardadas todo el entorno web es decir todas las páginas JSP.*

11. Manual de instalación

11.1 Requerimientos del sistema

Será necesario que en las máquinas que se desee instalar el producto dispongan del siguiente software instalado:

- *JRE (Java Runtime Evironment). http://www.oracle.com/technetwork/java/javase/downloads/index.html*
- *Base de datos MySQL.*
- *Servidor de aplicaciones y contenedor Web Tomcat, JBoss versión 5.1.0 GA. http://sourceforge.net/projects/jboss/files/JBoss/JBoss-5.1.0.GA*

11.2 Preparar entorno de ejecución

En este apartado se detallará los pasos necesarios para preparar el entorno del sistema para poder ejecutar con éxito la aplicación PeluSoft:

- *Será necesario configurar las siguientes variables de entorno del sistema operativo:*
  - ***JAVA_HOME:*** *C:\Program Files\Java\jdk1.6.0_25*
  - ***CLASSPATH:*** *C:\Program Files\Java\jre6\lib*

- **PATH:** C:\Program Files\Java\jdk1.6.0_25\bin;%JAVA_HOME%\bin

*En el paquete de instalación se entrega además de este mismo documento 2 scripts de sql, uno para crear el esquema de la base de datos (**PeluSoftSchema.sql**) y otro para incorporar los datos necesarios para poder ejecutar la aplicación (**PeluSoftData.sql**). Se deberán ejecutar estos 2 scripts desde el entorno de MySQL primero el de creación del esquema y luego el de creación de los datos.*

Además de lo anteriormente indicado se entrega un fichero llamado PeluSoft.war para desplegar el proyecto donde está el código fuente de la aplicación así como todas las librerías externas (jar) necesarias.

*Para poder acceder a los puntos de menu de la aplicación en los que solicita usuario y password, por defecto se distribuye el **usuario Admin con el password 1234.***

*Será necesario modificar el fichero **PeluSoft\src\hibernate.cfg.xml** (mirar texto de color rojo) indicando el password y usuario de acceso al SGBD MySQL.*

```
<property name="hibernate.connection.password">1234</property>
<property name="hibernate.connection.pool_size">1</property>
<property name="hibernate.connection.url">jdbc:mysql://localhost/peluqueria</property>
<property name="hibernate.connection.username">root</property>
```

Por último sólo quedará realizar el despliegue de la aplicación para ello seguiremos los siguientes pasos:

- *Copiar el fichero PeluSoft.war dentro de la carpeta de instalación de JBoss **…\jboss-5.1.0.GA\server\default\deploy***
- *Ejecutar **run.bat** ubicado en la carpeta de instalación de Jboss **..\jboss-5.1.0.GA\bin**, consiguiendo de esta manera arrancar el servidor JBoss*
- *Ahora sólo queda disfrutar de la aplicación arrancando un navegador web e indicando como dirección:*

  http://localhost:8080/PeluSoft

12. Conclusiones

Cuando empecé con este proyecto sólo tenía unos conocimientos básicos de Java y nada de J2EE. Ahora he logrado entender lo que siempre escuchaba y leía por todos sitios y que desconocía por absoluto, me estoy refiriendo a J2EE.

Con los conocimientos adquiridos con este proyecto, puedo seguir formándome para ampliar los conocimientos.

Ha sido una experiencia gratificante a nivel personal y espero que a nivel profesional también lo sea.

Como nota negativa, hubiese deseado disponer de más tiempo para el desarrollo del proyecto, por ejemplo la fase de implementación sólo fueron 5 semanas.

13. Links y referencias de consulta

En este apartado se indican los links o referencias que se han usado para la realización de las pruebas de los diferentes frameworks.

13.1 Hibernate

http://ricardopons.wordpress.com/2010/09/24/trabajando-con-hibernate-y-eclipse-helios-parte-ipreparando-el-entorno/

http://yaqui.mxl.uabc.mx/~larredondo/distribuidas/Hibernate.htm

13.2 Struts 2

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

http://mundogeek.net/archivos/2009/02/08/struts-2/

http://viralpatel.net/blogs/2009/12/tutorial-create-struts-2-application-eclipse-example.html

http://www.hachisvertas.net/blog/01/2009/01/05/primeros-pasos-con-struts2

http://www.youtube.com/watch?v=Blc5knV9tsU&list=PLBC52C9CEDC2F824A&index=1&feature=plpp_video

http://www.youtube.com/watch?v=qqUFaS1-RGE&list=PLBC52C9CEDC2F824A&index=2&feature=plpp_video

http://www.youtube.com/watch?v=XmxWAe4KKaw&list=PLBC52C9CEDC2F824A&index=3&feature=plpp_video