J2EE - LA LIBRERIA VIRTUAL

AUTOR:TOMÁS SAAVEDRA FERNÁNDEZ
ITIG

CONSULTOR: JAVIER FERRÓ GARCIA

9 de Enero de 2006.

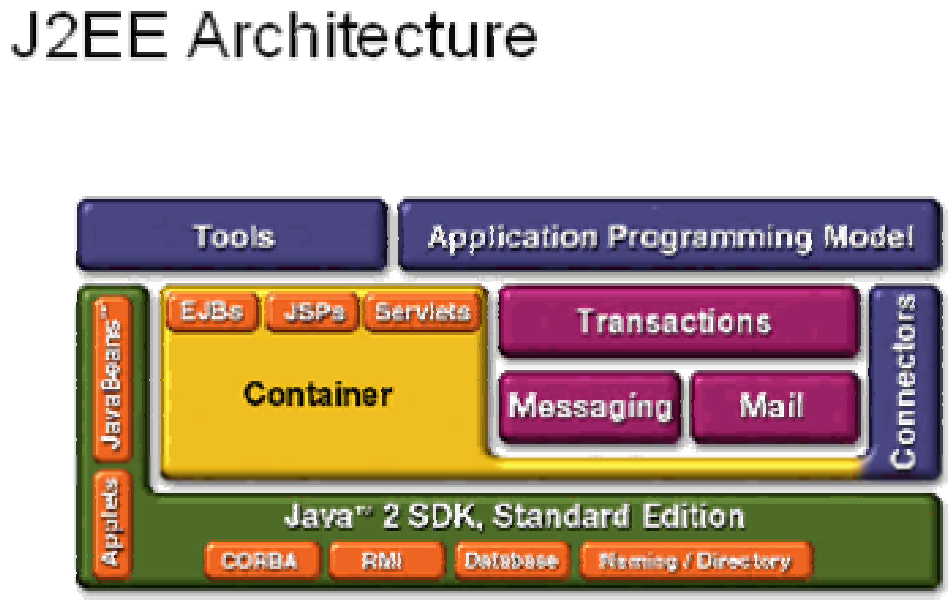
2.RESUMEN

Con este proyecto hemos logrado ofrecer a nuestros futuros clientes un producto de calidad, con un interfaz claro y sencillo y con unas funcionalidades muy definidas.

Se ha pretendido realizar un proyecto para un producto para realizar comercio electronico a traves de Internet, para ello no hemos decidimo por una “Librería Virtual”, donde nuestro clientes podran exponer todos los libros de su catalogo de ventas organizados por las propias categorias que el mismo ha creado, para que los futuros usuarios de esta Librería puedan desde cualquier parte del mundo y con un simple navegador web y una conexión a Internet, realizar cualquier pedido de libros que desea comprar.Tambien permitiremos que esta Librería pueda ser administrada (mantenimiento del catalogo de libros , de las nuevas categorias de la librería, asi como la consulta de los datos de cualquier cliente o pedido ya realizado) desde cualquier parte del mundo , contando igualmente con un navegador web y una conexión a Internet.

El producto que entregamos le garantizamos la extensibilidad y reutilización del modelo, aunque esto puede estar claramente restringuido por la propia funcionalidad del negocio que tengamos.

Todo esto ha sido desarrollado mendiante tecnologías de vanguarida con es el Kit de desarrollo de Sun Microsystem J2EE ,Jakarta Apache Tomcat y MySQL, todas ellas herramientas opensource de libre distribución y exceptas de royalties.



3.INDICE DE CONTENIDOS

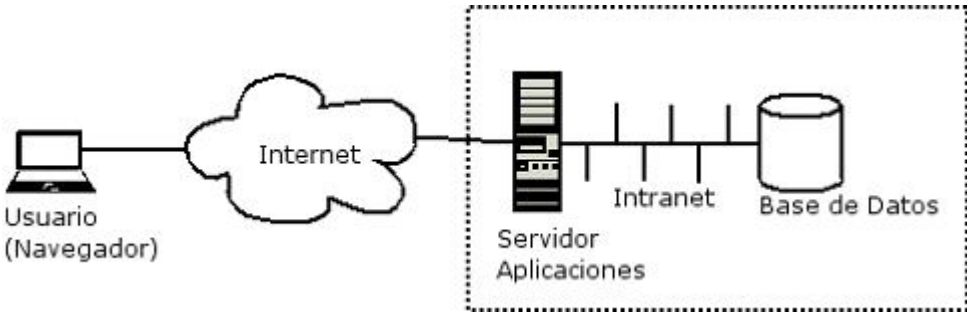
4.CUERPO DE LA MEMORIA

4.1 Capitulo 1,Introducción

4.1.1 Justificación del proyecto

Hoy en día la mayoría de los negocios tradicionales no puede limitar sus ventas y organización de la gestión cotidiana a un sistema presencial y físico , donde el cliente para poder acceder a los servicios que la empresa nos ofrece tenemos que desplazarnos físicamente a un establecimiento para efectuar cualquier compra o los mismos gestores de la empresa que deben de estar físicamente delante del ordenador de la empresa para realizar cualquier gestión cotidiana, esto ya actualmente esta superado con la implantación de las nuevas tecnologías, en concreto el uso de Internet y las enormes ventajas que nos proporciona.

¿Qué ventajas?, principalmente la compra a distancia , el comercio electrónico, ya no tenemos por que estar físicamente en el establecimiento para efectuar cualquier compra, desde nuestra casa o trabajo podemos consultar los productos que nos ofrecen y efectuar la compra, lógicamente todo esto debe de efectuarse con un precio inferior al que pueda tener el propio producto en el establecimiento y con cierto valor añadido (envoltorios para regalos, tarjetas de felicitación,ofertas puntuales,….), también con facilidad de recibir l compra en nuestro propio domicilio con un plazo de entrega muy corto y efectuar el pago con cualquiera de los métodos tradiciones o algunos que nos proporciona la propia Internet en un entorno seguro: tarjetas de créditos, trasferencias bancarias onlines, paypal y similares….



Para realizar todo esto podemos utilizar herramientas que estas disponibles hoy en día en el mercado como por ejemplo Oscommerce, Zen Cart, PhpShop y muchas otras, eso si ninguna de ellas son tan flexibles como para adaptarse a cualquier posible situación de nuestros clientes en este caso una Librería tradicional, ¿ Por que no ?, principalmente por su poca intuición y manejabilidad por parte del usuario que va a administrar la Librería y por un sistema de ventas demasiado “enredado” para los futuros clientes esta librería y por que también: el uso de tecnologías no demasiado apropiadas para la labor.
Es por ello que realizaremos una aplicación a medida de nuestro cliente, fácil y simple de usar para Él y sus futuros compradores y fácilmente adaptable a futuras necesidades de la empresa.

4.1.2 Objetivos del Trabajo Final de Carrera

Nuestro objetivo con este trabajo es plasmar en un producto final todo lo que hemos aprendido a lo largo de todas las asignaturas cursadas en nuestra carrera, realizando por ello un trabajo metodológico y por fases acordes a muchas de las asignaturas trabajadas como: Ingeniería del Software, Programación orientada a Objetos,Bases de datos I y II, Técnicas de desarrollo de software,…

Generales

Nuestro objetivo principal con este trabajo es intentar aprender el uso de las distintas técnicas que existen para implementar una solución empresarial de comercio electrónico, donde el principal usuario de esta solución será un individuo con pocos o nulos conocimientos informáticos que usa la red para su ocio personal, familiar y en este caso en concreto para efectuar compras a distancias , planteamos el ejemplo concreto de una librería online al estilo de otras como Amazon, La librería electrónica de El Corte Ingles y similares.
La principal herramienta con la que contara nuestro potenciales usuarios no será otra sino su navegador web (Internet Explorer, Opera, Firefox u otros).
Por supuesto también nos ocuparemos de la gestión interna de esta nuestra librerías, para ello también diseñaremos un completo sistema de administración de la misma utilizando nuestro navegador para su correcta gestión.

Específicos

Básicamente nuestra aplicación web se puede "manejar" desde dos puntos de vista totalmente distintos pero a la vez compartiendo la misma filosofía:

- Tienda virtual online, con muestra de catálogos, búsquedas y realización de compras y en su caso enlace con pasarela de pagos segura además de nuestro universalmente conocido "carrito de la compra".
- Administración privada de la tienda, con posibilidad de creación de catálogos, categorías, administración de pedidos , clientes,…

Por ello nuestro sistema tendrá la posilidad de identificar a los posibles usuarios para dependiendo de su perfil concreto realizar algunas de las dos tareas descritas anteriormente.

para lograr este objetivo hemos tenido que principalmente:

- ***Conocer el funcionamiento de un sistema de comercio electrónico.***
  La mejor manera de sabe lo que queremos y saber como funciona, y así lo hemos realizado, investigando en diversas web de comercio electrónico para conocer su interfaz y lógica de trabajo, también instalando y probando aplicaciones comerciales como las ya mencionadas y también haciendo un pequeño estudio de mercado (nada serio, a unos amigos con un par de negocios tradicionales) para ver como querían que se implantara un sistema virtual como el que vamos a desarrollar.
- ***Uso de tecnologías J2EE.***
  Entender como funciona los servidores de Internet y mas concretamente los servidores empresariales de comercio electrónico , así como los servidores de aplicaciones y diversos contenedores web que existe, ventajas y facilidad de uso de cada uno de ellos, aprender tecnología de Servlets y JSP y aplicar los conocimiento ya adquiridos de programación orientada a objetos y base de datos bajo JDBC para el lenguaje JAVA.
- ***Implementar una solución practica del producto***
  Todo ello con un claro objetivo : analizar,diseñar e implementar una aplicación para el mundo real donde plasmemos todo lo conseguido

4.1.3 Enfoque y método seguido.

Para conseguir nuestra aplicación final hemos decidido llevar a cabo nuestro trabajo siguiendo la metodología clásica del ciclo de vida un sistema informático, es decir llevar a cabo un trabajo por fases, retroalimentando cada una de ellas a la otra.

hemos dividido el proyecto en tres fases principales: Análisis de especificaciones, Diseño de la aplicación y Implementación.

cada una de ellas brevemente resumidas en estos apartados:

Análisis de requisitos de la aplicación.

- Elaboración de diagramas de casos UML gráficos y textuales
- Esquema de navegación web por la aplicación tanto el modo compra como en modo administración.
- Diseño preliminar de la interfaz grafica de usuario.
- Instalación librerías ,aplicaciones y entorno de desarrollo
- Recopilación de información.

Diseño de la aplicación.

- Diseño de la base de datos.
- Modelo Entidad/Relación.
- Generación de Base de datos,Tablas e índices.
- Juego de pruebas de la bases de datos.
- Especificaciones de actores de la aplicación.
- Diagramas de clases.

Implementación de la aplicación.

- Implementación clases de entidad
- Implementación de otras clases.
- Implementación de gestor de conexión y acceso a base de datos.
- Implementación interfaz grafica HTML y JSP
- Implementación servlets de gestión.
- Control de errores y excepciones.
- Optimización de código.
- Empaquetado de la aplicación.
- Pruebas finales.
- Elaboración memoria final y entrega.

4.1.4 Planificación del proyecto.

Numero Tarea	Tarea	Duración en Semanas	Tareas Precedentes
2.1	Elaboración de diagramas de casos UML gráficos y textuales	2	
2.2	Esquema de navegación web por la aplicación tanto el modo compra como en modo administración.	3	2.1
2.3	Diseño preliminar de la interfaz grafica de usuario.	3	2.2
2.4	Instalación librerías ,aplicaciones y entorno de desarrollo	6	
2.5	Recopilación de información	16	
3.1	Diseño de la base de datos.	1	2.1 2.2 2.3
3.2	Modelo Entidad/Relación.	2	2.1 2.2 2.3 3.1
3.3	Generación de Base de datos,Tablas e índices.	1	2.1 2.3 2.3 3.1 3.2
3.4	Juego de pruebas de la bases de datos.	1	2.1 2.3 2.3 3.1 3.2 3.3
3.5	Especificaciones de actores de la aplicación.	1	2.1 2.2 2.3
3.6	Diagramas de clases.	3	2.1 2.2 3.2 3.5
4.1	Implementación clases de entidad.	2	3.6
4.2	Implementación de otras clases.	2	3.6
4.3	Implementación de gestor de conexión y acceso a base de datos.	3	3.1 3.2 3.3 3.4 3.6 4.1 4.2
4.4	Implementación interfaz grafica HTML y JSP.	6	4.1 4.2 4.3
4.5	Implementación servlets de gestión.	5	4.1 4.2 4.3 4.4
4.6	Control de errores y excepciones.	6	4.1. 4.2 4.3 4.4 4.5
4.7	Optimización de código.	6	4.6
4.8	Empaquetado de la aplicación.	1	4.7
4.9	Pruebas finales.	2	4.8
4.10	Elaboración memoria final y entrega.	1	4.9

SEM	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2.1																
2.2																
2.3																
2.4																
2.5							7 -11 PEC2									
3.1																
3.2																
3.3																
3.4																
3.5																
3.6											9 -12 PEC3					
4.1																
4.2																
4.3																
4.4																
4.5																
4.6																
4.7																
4.8																
4.9																
4.10																9 -1 PEC4

4.1.5 Productos obtenidos

El producto que hemos obtenido “Librería Virtual”

no es mas que una aplicación web desarrollada en JAVA, usando para ello los Servlets y las paginas web dinámicas JSP, accediendo a base de datos con tecnología JDBC.

Nuestro producto se instalara bajo Apache Tomcat, siendo necesario el SDK de JAVA disponible así como el conector JDBC.

Este producto se puede instalar fácilmente en cualquier servidor de la empresa, simplemente debe de contar con una conexión a Internet con un ancho de banda lo suficientemente rápido para permitir que se conecten a el con múltiples peticiones para efectuar las ventas a distancia, también deberá contar con un navegador web para poder administrar los catálogos de la librería.

La aplicación se divide en :

- Paginas JSP
- Servlets programados en JAVA
- librerias JSTL
- graficos e imágenes
- Base de Datos en MYSQL
- conector JDBC-MYSQL

Todo ello distribuido bajo un fichero de aplicación web .war y unos scripts SQL para la creación de las tablas y los registros con datos iniciales y de pruebas.

4.1.6 Descripción del resto de capitulos

4.2 Capitulo 2, comentaremos el Analisis de requisitos de la aplicación, con los apartados que hemos mencionado en el apartado 4.1.3.

4.2 Capitulo 3, en este apartado hablaremos del diseño de la aplicación

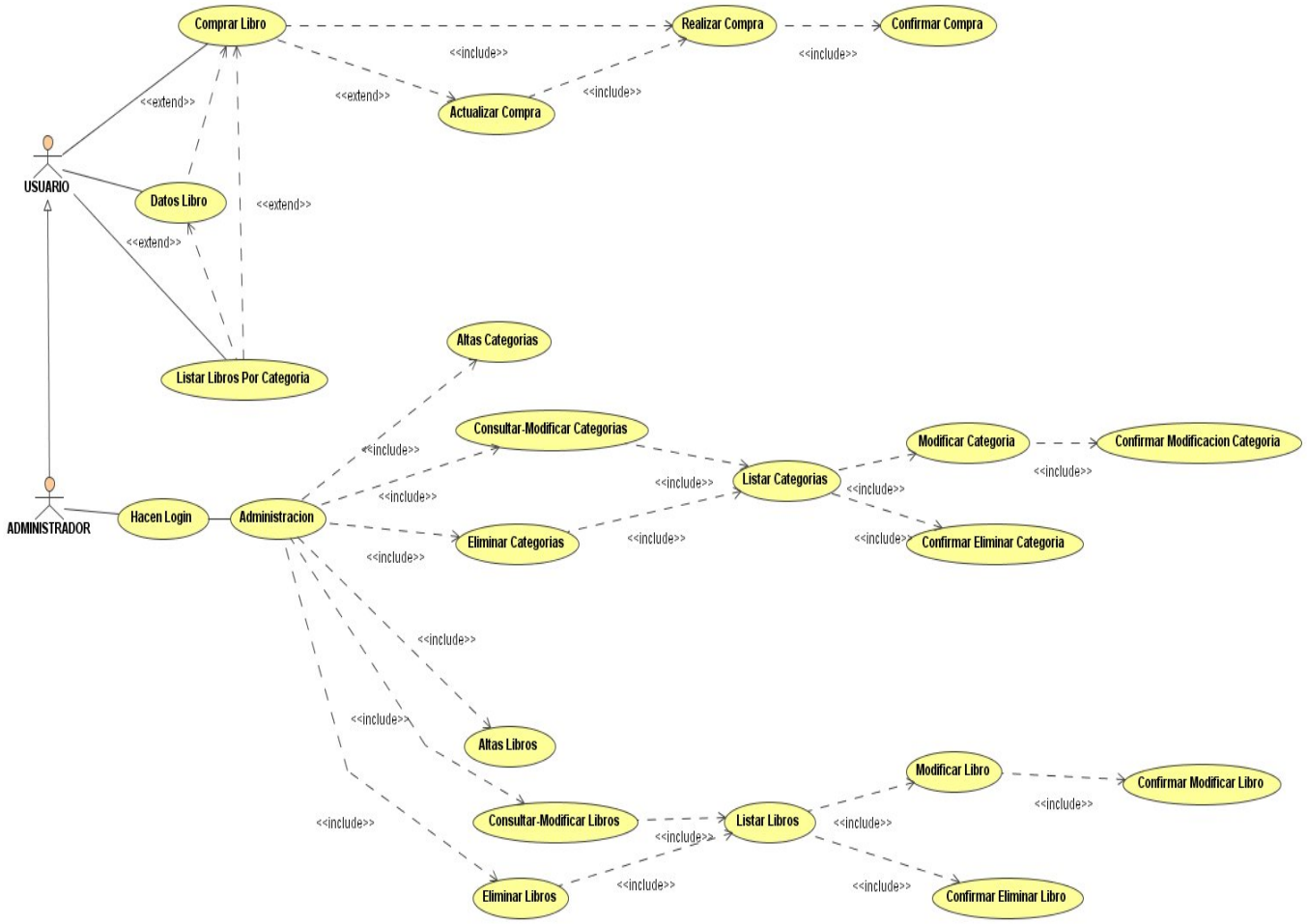
4.3 Capitulo 4, en este apartado comentaremos la fase de implementacion de la aplicación asi de cómo se debe de instalar en un servidor.

4.2 Capitulo 2,Analisis de requisitos

En esta fase de Análisis de nuestro Trabajo Final de Carrera , trataremos los siguientes puntos:

- Elaboración de diagramas de casos y secuencias UML gráficos y textuales
- Esquema de navegación Web por la aplicación tanto el modo compra como en modo administración.
- Diseño preliminar de la interfaz grafica de usuario.
- Instalación librerías ,aplicaciones y entorno de desarrollo
- Recopilación de información.

4.2.1 CASOS DE USOS GRAFICOS



Partimos principalmente de 2 Actores principales en la aplicación:

El Usuario de la Web que quiere comprar un libro y el Administrador de nuestro sitio Web.

En la fase de diseño de la aplicación tal como marcamos en el Plan de Trabajo se analizara mas en profundidad el papel que juegan estos dos actores en nuestro sitio Web.

4.2.2 CASOS DE USOS TEXTUALES

1.Datos Libro

Resumen de la funcionalidad: Se consulta los datos completos de un libro
Papel dentro del trabajo del Usuario:básico
Actores: Todos
Casos de usos relacionados: 3.Comprar Libro
Precondición:ninguna (se entiende que esta en la Web)
Postcondición: Se compra el libro
Flujo de eventos:
El usuario entra en la pagina índice de la WEB y a través de la zona central de la misma donde aparecen los últimos libros añadidos a la Web y siempre que este ahí el que le interese pulsa sobre el título del libro en cuestión saltando a otra pagina donde se muestran todos los datos del libro: titulo,autor,comentarios,precio,…
Desde esta página el usuario puede volver al índice o comprar el libro en cuestión.

2.Listar Libros por Categorías

Resumen de la funcionalidad: Se consulta un breve resumen (en tabla) con todos los libros de una categoría en cuestión.
Papel dentro del trabajo del Usuario:básico
Actores: Todos
Casos de usos relacionados: 1.Datos libros ; 3.Comprar Libro
Precondición:ninguna (se entiende que esta en la Web)
Postcondición:Se compra el libro
Flujo de eventos:
El usuario entra en la pagina índice de la WEB y a través de la zona izquierda de la pantalla pulsa en la categoría (se muestran todas en tabla) de la cual quiere saber todos los libros que existen en la librería, y se accede a otra pagina donde se muestran en forma de tabla todos los libros de esta categoría.
Desde esta página el usuario puede volver al índice , comprar el libro en cuestión o pulsando sobre el título del libro ver el libro en cuestión.

3.Comprar Libro

Resumen de la funcionalidad: A partir de esta pagina se compra un/unos libro/s (carrito de la compra , shopping cart) , se elige la cantidad de libros a comprar o en momento dado se elimina de los libros a comprar alguno en cuestión.
A esta página se puede acceder directamente a través de un link desde cualquier página de la aplicación cuando somos usuarios del sitio Web.
Papel dentro del trabajo del Usuario:básico
Actores: Todos
Casos de usos relacionados: 4.Actualizar compra (se vuelve a este caso de uso , el 3 con el libro/s eliminado/s en su caso, con los importes actualizados a las cantidades y se actualiza el total de compra) ; 5.Realizar compra
Precondición:se ha comprado algún libro.
Postcondición:se elimina algun libro comprado,se actualizan los datos, se confirma la compra,
Flujo de eventos:
A través de este “carrito de la compra” el usuario gestiona sus compras en la Web, pulsando eliminar se elimina el libro de la compra, y indicando la cantidad de libros y pulsando actualizar se actualizan los importes y total de la compra, pulsando realizar compra se confirma la compra realizada.

5. Realizar Compra

Resumen de la funcionalidad: Tras terminar la compra se introduce los datos del usuario para realizar el envió del libro/s
Papel dentro del trabajo del Usuario:básico
Actores: Todos
Casos de usos relacionados: 3.Comprar Libro.
Precondición:se ha realizado la compra.
Postcondición:se confirma la compra con un mensaje.
Flujo de eventos:
Se introducen todos los datos del cliente (algunos son requeridos) y se pulsa en confirmar, se falta algún dato el sistema me lo indica y si todo es correcto se nos informa mediante un mensaje.

6. Hacer Login

Resumen de la funcionalidad: Se introduce el nombre de usuario y la clave del administrador para administrar el sitio Web
Papel dentro del trabajo del Usuario:ocasional
Actores: Administrador
Casos de usos relacionados: 7.Administración.
Precondición:se ha realizado login correctamente
Postcondición:se accede a la página de administración.
Flujo de eventos:
Se introduce el login y la clave y se pulsa en entrar si el login y la clave es correcto se accede a la administración si no es correcto se nos indica con un mensaje.

7.Administración

Resumen de la funcionalidad: A partir de esta pagina se gestiona el catalogo de libros y las categorías de estos.
Papel dentro del trabajo del Usuario:ocasional
Actores: Administrador
Casos de usos relacionados: 8.Altas Libros o Categorias ; 9 Listar Libros o Categorías.
Precondición:se ha realizado login correctamente
Postcondición:se gestiona los libros y categorías
Flujo de eventos:
Desde esta pagina y pulsado sobre los correspondientes enlaces se accede a la administración básica del sitio Web o en su defecto se vuelve a la página índice.

8.Altas Libros o Altas Categorías

Resumen de la funcionalidad: A partir de esta página se dan de altas libros o categorías en la base de datos del sistema.
Papel dentro del trabajo del Usuario:ocasional
Actores: Administrador
Casos de usos relacionados: 7.Administración
Precondición:se ha pulsado sobre Altas en la pagina de Administración.
Postcondición:se da de alta un libro o una categoría.
Flujo de eventos:
Desde esta pagina se introduce todos los datos de un libro o una categoría en cuestión, si algunos de los datos son obligatorios no los indica al intentar efectuar la grabación de los datos,

pulsando grabar se graban los datos en la base de datos del sistema confirmándonos mediante un mensaje que todo es correcto.

8.Listar Libros o Listar Categorías

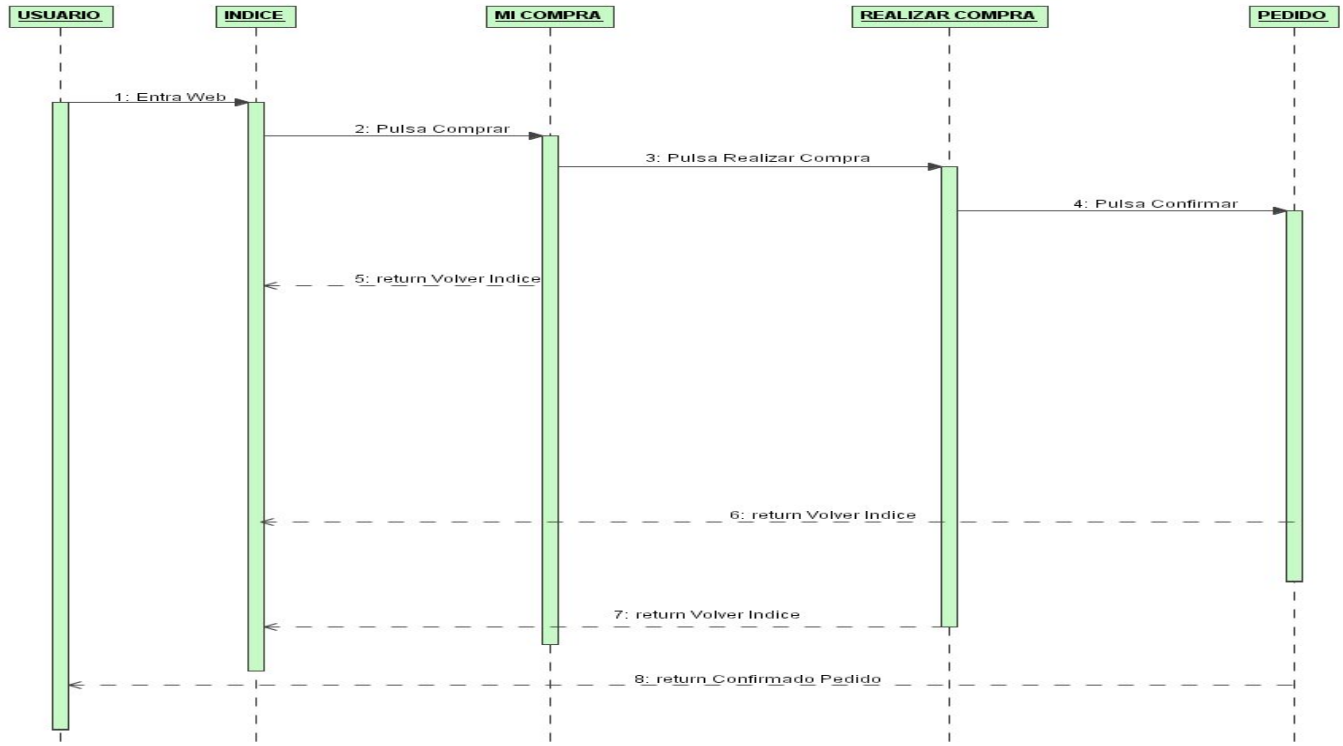
Resumen de la funcionalidad: A partir de esta pagina nos muestran todos los libros del catalogo o todas las categorías existentes.
Papel dentro del trabajo del Usuario:ocasional
Actores: Administrador
Casos de usos relacionados: 7.Administración ; 9.Modificar Libro o Categoría.
Precondición:se ha pulsado sobre consultar-modificar libro o categoría o sobre eliminar libro o categoría en la pagina de Administración.
Postcondición:se elimina un libro o categoría o se accede a los datos de un libro o categoría.
Flujo de eventos:
Desde esta pagina se nos muestra en forma de tabla todos los libros o categorías existentes en el sistema (los datos principales de ambas tablas no todos los datos), por cada registro podemos pulsar en eliminar , en este caso no pide confirmación y si es así nos indica mediante un mensaje que el dato se ha eliminado, en el caso de que pulsemos consultar-modificar accedemos a otra pagina con los datos completos del registro.

9.Modificar-Consultar Libros o Modificar-Consultar Categorías

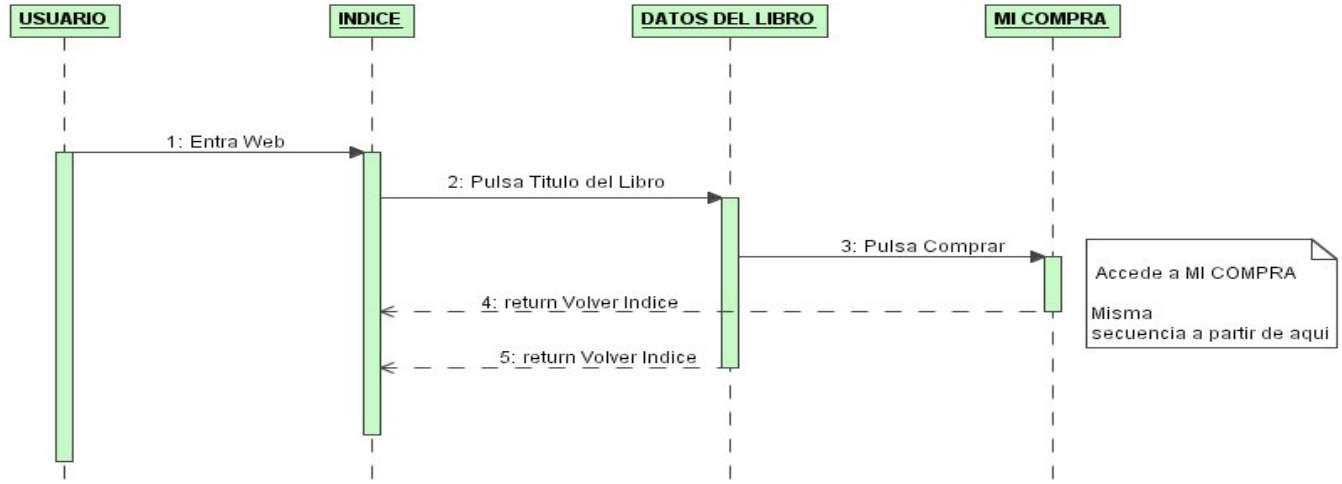
Resumen de la funcionalidad: A partir de esta página consultamos los datos completos de un libro o categoría pudiendo modificarlo en su caso
Papel dentro del trabajo del Usuario:ocasional
Actores: Administrador
Casos de usos relacionados: 8 Listar Libros o Categorías.
Precondición: Se ha pulsado sobre la tabla de Libros o Categorías el botón de consultar-modificar.
Postcondición:se graba las modificaciones de la ficha de un libro o categoría o se vuelve al índice sin grabar.
Flujo de eventos:
Se nos muestra los datos actuales completos de un libro o categoría pudiendo modificar algunos de ellos y confirmando la grabación pulsando grabar o simplemente se consulta los datos y no se pulsa grabar y se pasa de nuevo al índice.

4.2.3 DIAGRAMAS DE SECUENCIAS

Se realiza una compra directamente desde el índice



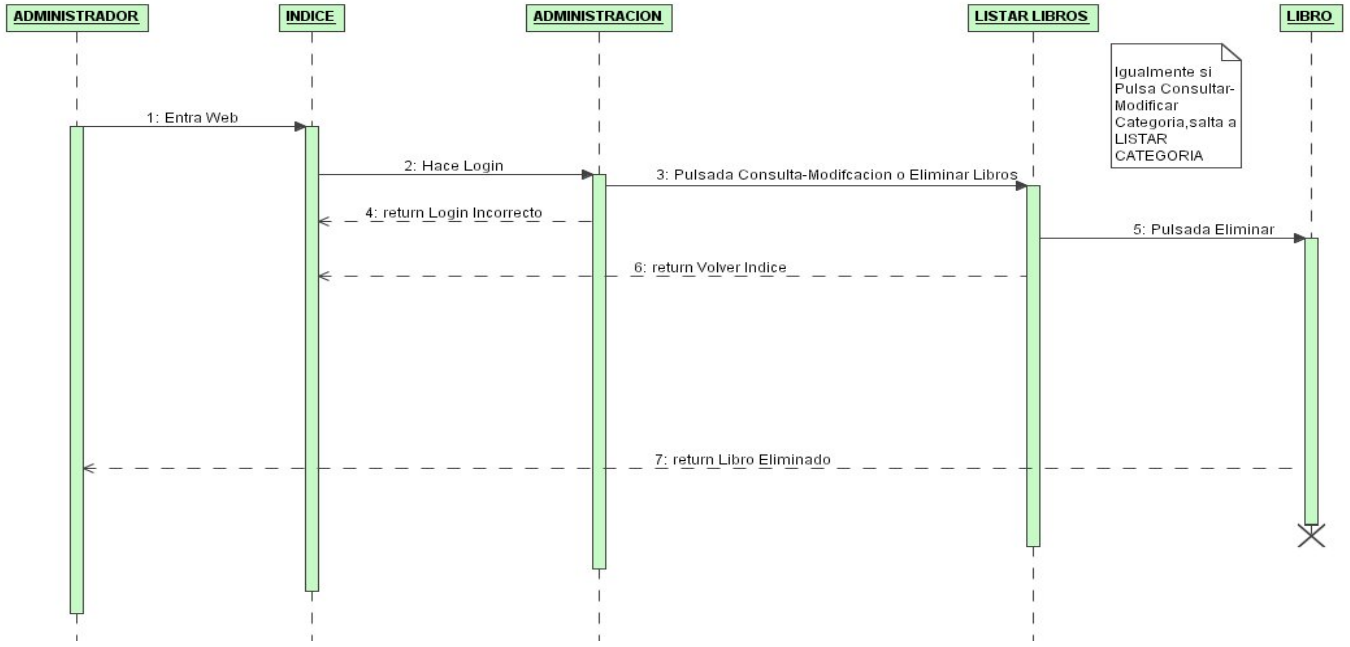
Se realiza una compra viendo los datos del libro, desde su ficha.



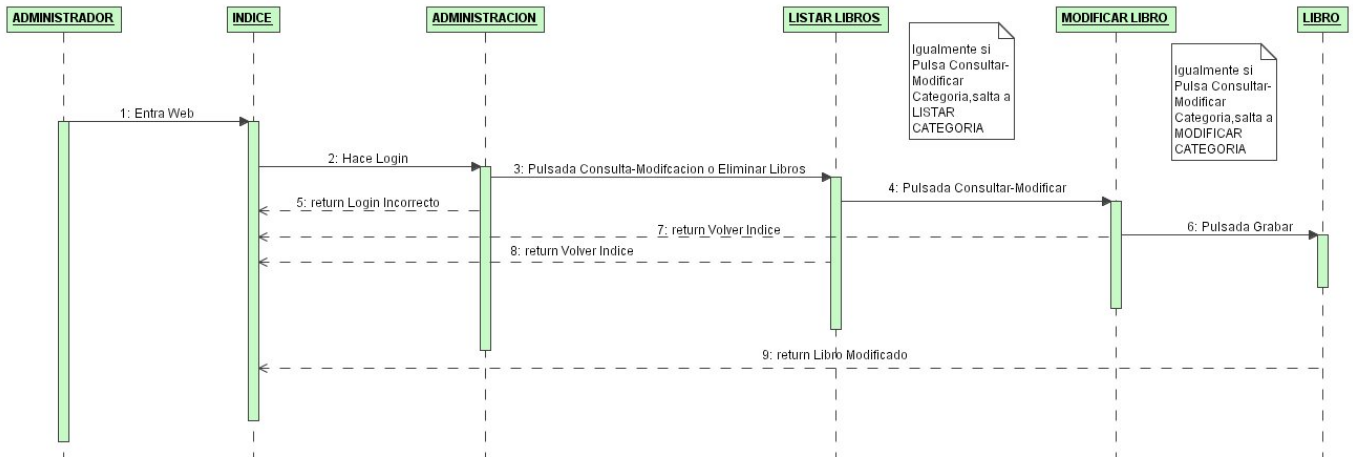
Se realiza una compra desde la lista de libros de una categoría en concreto

El Administrador da de alta un libro

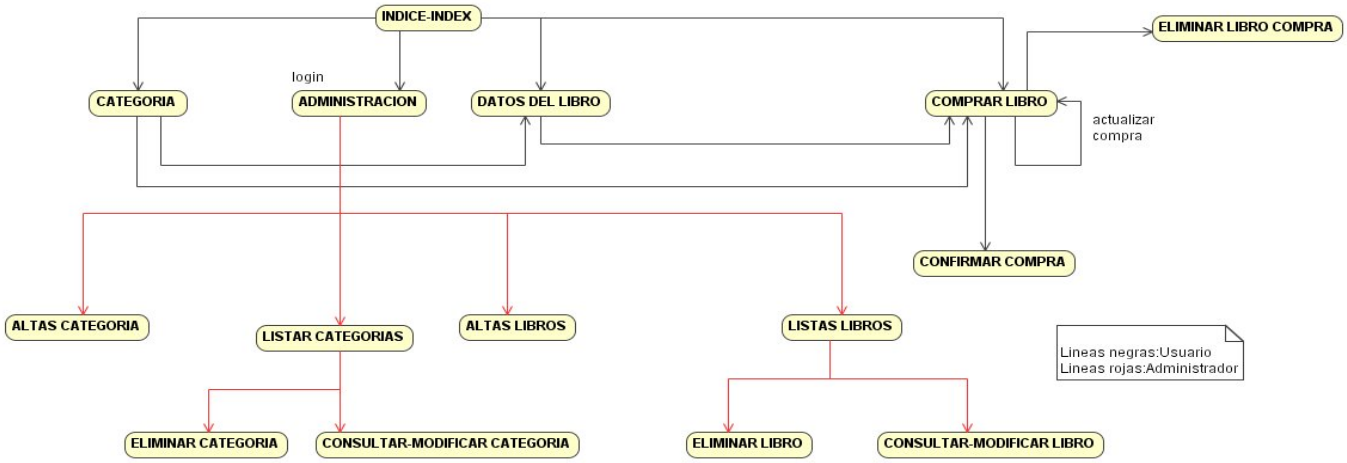
El Administrador elimina un libro



El Administrador modifica un libro

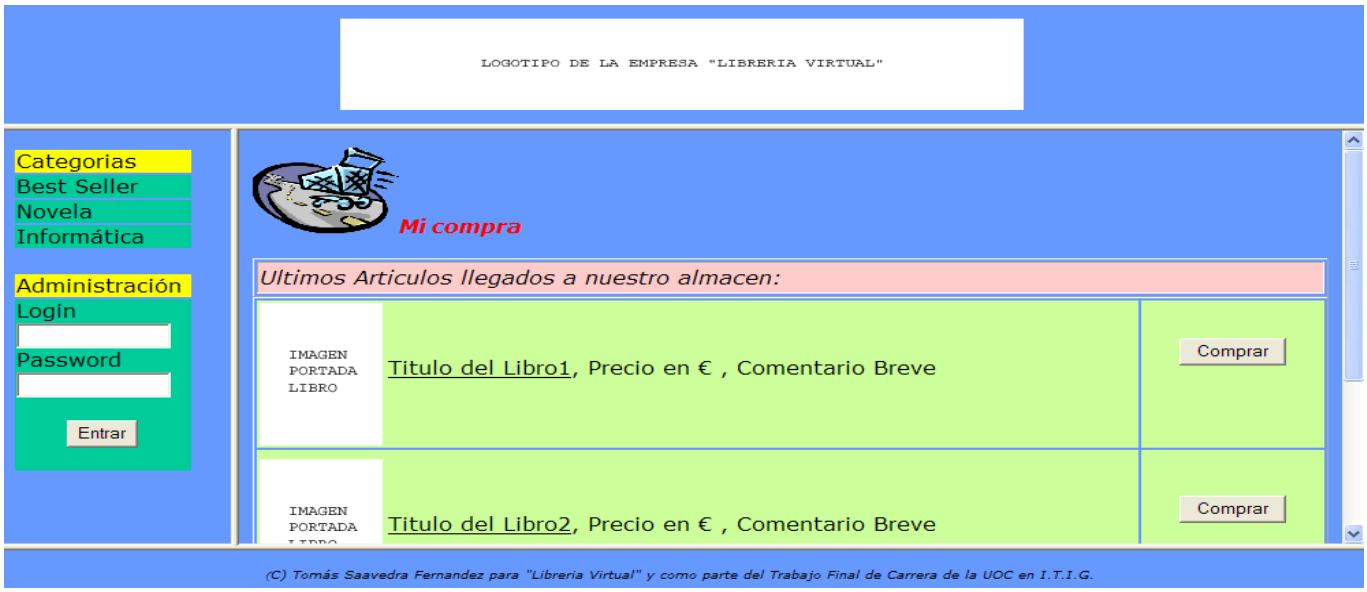


4.2.4 FLUJO DE NAVEGACION



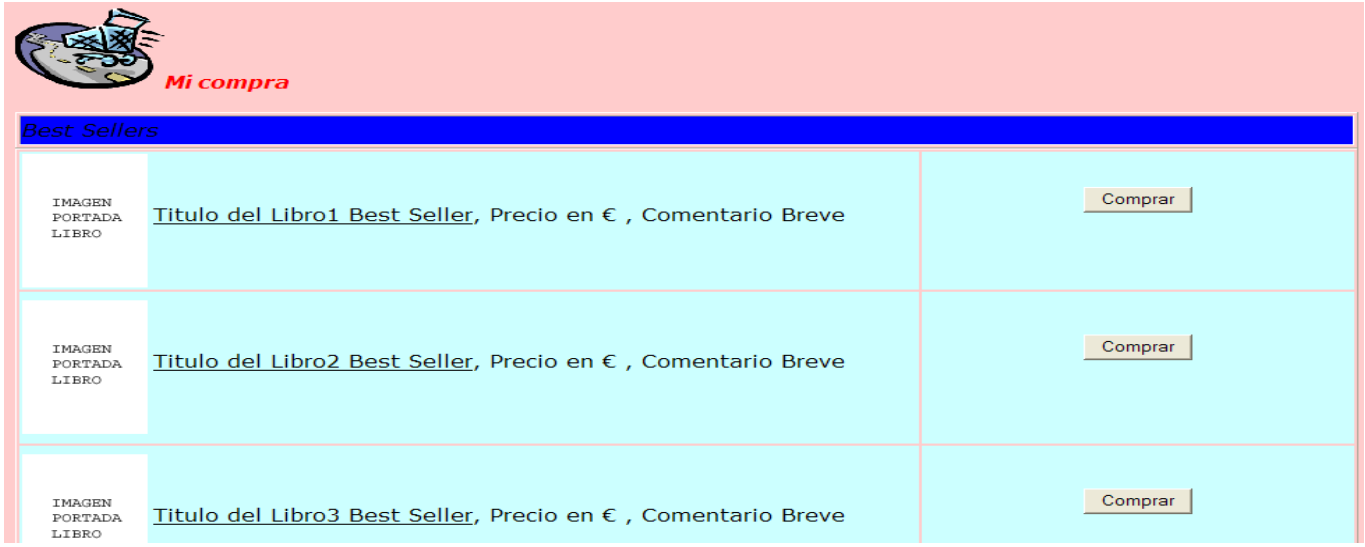
4.2.5 DISEÑO PRELIMINAR INTERFAZ GRAFICA

Pagina índice (con marcos)



Datos del Libro (pulsado el titulo del libro en el índice)

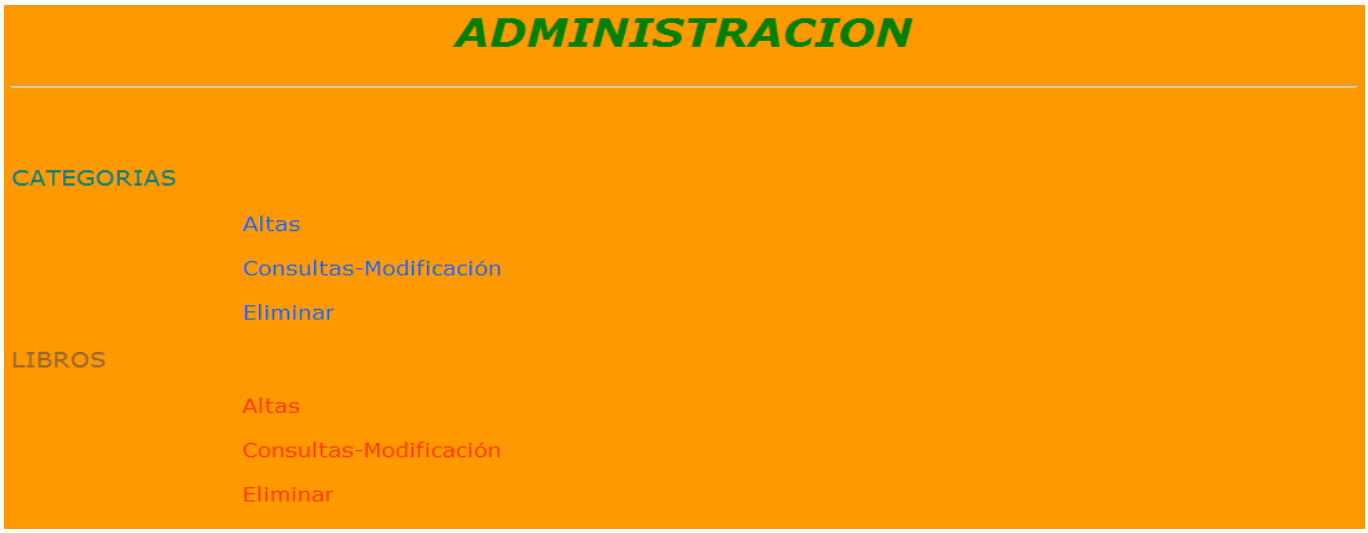
Categorías (pulsada el nombre de una categoría en el índice)



Mi compra, carrito de la compra (Pulsada Comprar en índice en Categorías ,datos del libros o sobre el icono Mi compra)

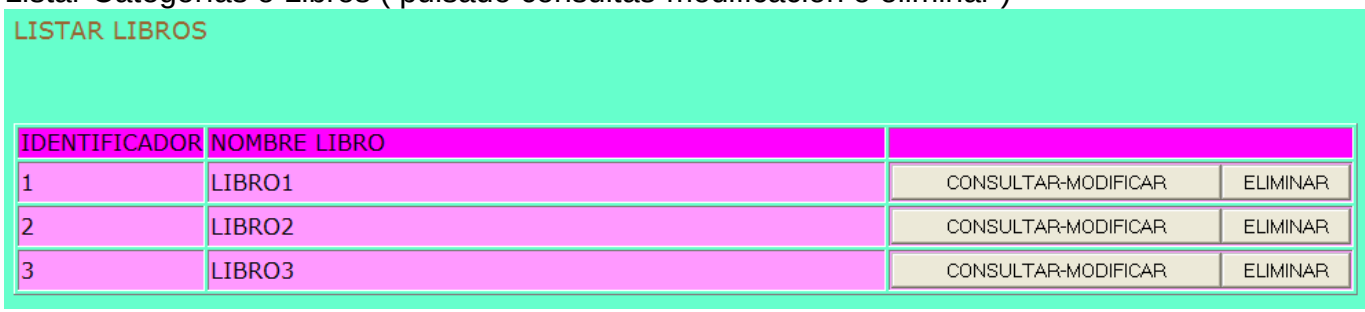
Realizar Compra (pulsado el botón Realizar Compra en Mi compra)

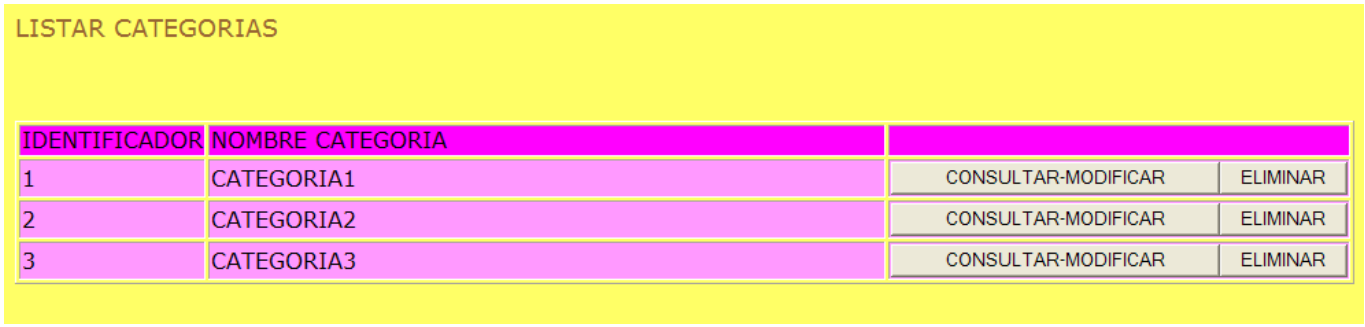
Administración (pulsada entrar tras hacer login en el índice de manera correcta)



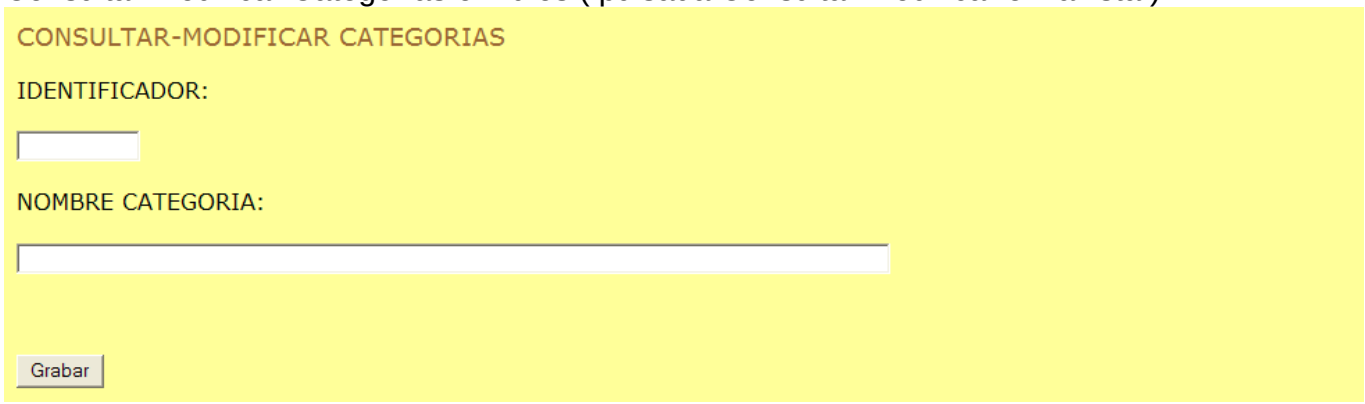
Altas Categorías o Altas Libros (pulsada Altas)

Listar Categorías o Libros (pulsado consultas-modificación o eliminar)





Consultar-Modificar Categorías o Libros (pulsada Consultar-Modificar en la lista)



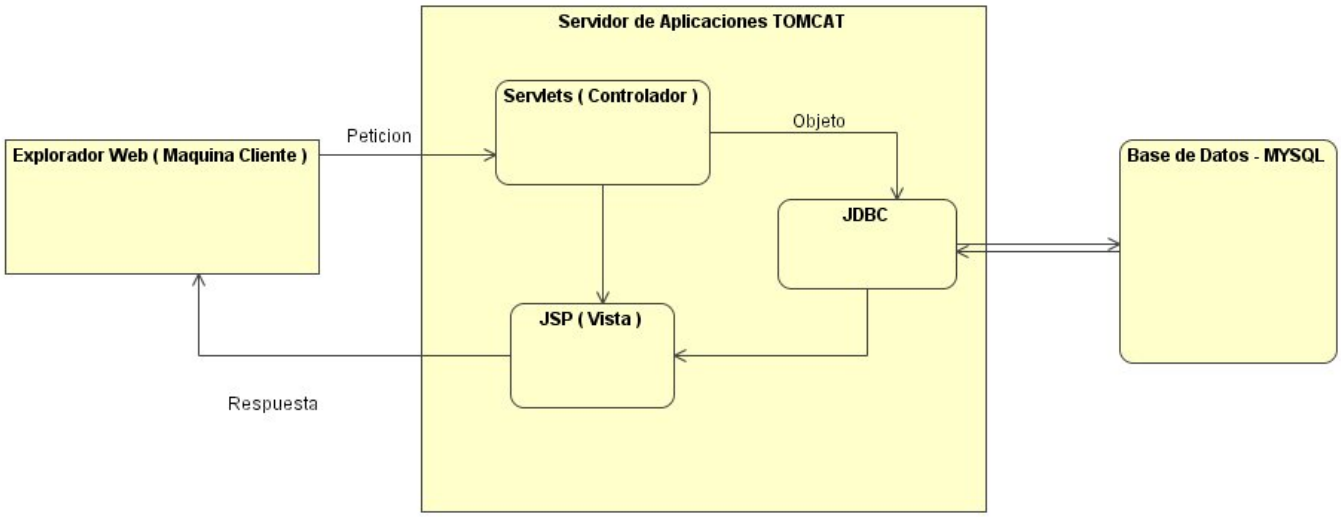
Quizás en versiones posteriores se gestiones desde la administración la gestión completa de pedidos, clientes, perfiles de usuarios y enlace con una pasarela de pagos seguros.

4.2.6 INSTALACIÓN LIBRERÍAS ,APLICACIONES Y ENTORNO DE DESARROLLO

Para todo el desarrollo de nuestra librería virtual hemos utilizado el entorno de desarrollo NetBeans IDE 4.1 baja la plataforma JAVA JDK 1.5 y como servidor de aplicaciones para servlets y JSP , Jakarta Tomcat 5.5.7 ambos integrado dentro del mismo entorno de desarrollo, como base de datos hemos usado MYSQL Server 4.1 y MYSQL Control Center 0.9.4 como entorno grafico para MYSQL , para su utilización en java con JDBC usamos el driver mysql-connector 3.1.10.

La principal técnica de trabajo que utilizaremos en J2EE serán Servlets y JSP.

Los Servlets son clases Java con capacidad para procesar peticiones Web a través de formularios y procesar de manera dinámica las posibles respuestas, en cambio JSP se incrustan dentro de documentos HTML para proporcionar contenidos dinámicos a las paginas Web y como sistema de visualización de los datos procesados por los Servlets.



Nuestro modelo usa las paginas JSP para la presentación y los Servlets para el procesamiento y la de creación de objetos software (como el control de acceso a los datos por JDBC) necesario para las paginas JSP, también los propios Servlets deciden a que pagina JSP envían la información.

La ventaja de esta arquitectura es que no hay procesamiento dentro de la capa de presentación de datos, se separa claramente el proceso de diseño y el de programación.

Se utiliza el patrón de diseño modelo vista-controlador (MVC) , donde el modelo lo forman los componentes que controlan los datos que utilizan la aplicación , la vista los que representan los datos al cliente y el controlador lo forman los componentes responsables de la gestión de eventos y de coordinar las actividades del modelo y de la vista.

4.2.7 RECOPILACIÓN DE INFORMACIÓN

Para recompilar información útil en todo nuestro proyecto nos hemos basados en librerías Web o sistemas de comercio electrónico ya desarrollados como son las tiendas virtuales de AreaPc, Amazon y el sistema OSCommerce aunque todo esto principalmente usan PHP y/o ASP nos sirve de forma de ejemplo para ver como desarrollar un sistema que se adapte de manera adecuada a las exigencias de nuestros futuros clientes.

4.3 Capitulo 3 ,Diseño de la aplicación

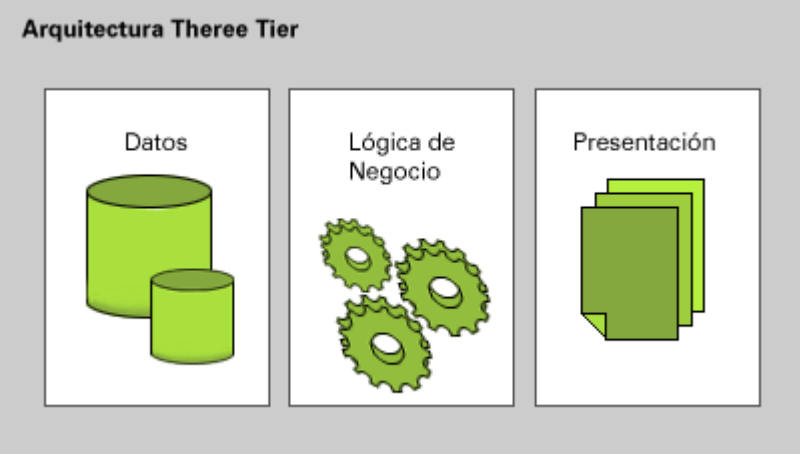
En esta fase de Diseño de la Aplicación , trataremos los siguientes puntos:

- Arquitectura
- Patrones de Diseño
- Diagramas de clases.
- Persistencia.
- Modelo Entidad/Relación.
- Diseño de la base de datos.

4.3.1 Arquitectura

Como bien comentamos en la fase de analisis de la aplicación, vamos a utilizar como arquitectura de diseño en nuestra aplicación un modelo de 3 capas- three tier:

Vista- Modelo – Datos



La **capa de datos** representa el mecanismo por el cual se manipula y persiste la información. Consiste en un administrador de bases de datos relacional (RDBMS), y el esquema de datos propio de nuestra aplicación. Cuando hay varias aplicaciones presentes, los modelos de datos se complementan, evitando la duplicación de información y aumentando las facilidades de brinda el sistema como un todo.

En la **capa de lógica de negocio-modelo** se modela el comportamiento del sistema, basándose en los datos provistos por la capa de datos, y actualizándolos según sea necesario. Esta capa describe los distintos procesos de negocio que tienen lugar en las organizaciones, desde el ciclo de aprobación de un documento hasta la política de consultar un pedido.

Finalmente, la **capa de presentación-vista** contiene todos los elementos que constituyen la interfaz con el usuario. Esta capa incluye todo aquello con lo que el usuario puede interactuar, como por ejemplo las pantallas de las aplicaciones, el modelo de navegación del sistema y los adaptadores para cada modo de acceso .

Los diseños "three tier" son ampliamente utilizados en el mercado, y a lo largo del tiempo han probado sus ventajas. Las aplicaciones en tres capas típicamente tienen mayor capacidad de crecimiento y son más sencillas de mantener, dada su naturaleza altamente modular.

De esta manera separamos la interfaz de usuario, la logica de negocio y los datos.

la ventaja del uso de esta arquitectura es que no hay procesamiento dentro de la capa de presentación; esta unicamente se encarga de recoger objetos de la capa de negocio-modelo e insertarlo para que el cliente lo vea, esta separación favorece el trabajo futuro entre el desarrollador y diseñador, otra de las ventajas del modelo es que la capa de modelo es que el esta presenta un solo punto de entrada a al aplicación , ademas de reposabilizarse del estado de la aplicación y de la seguridad, lo que facilita muchismo el mantenimiento futuro, tambien es verdad que el modelo puede resultar complejo de implementar en la parte de comunicación entre la capa modelo y capa vista , para la actualización de cambio que se vayan produciendo.

- Lo más reutilizable y que es menos susceptible de cambio, es el modelo. Las reglas del negocio no cambian de un día para otro. Si tenemos un conjunto de clases que mantengan hagan algo en concreto, es posible que esas clases (o funciones y estructuras de datos) nos sirvan durante mucho tiempo sin necesidad de tocarlas. En un punto intermedio está el controlador. Es posible que mejoremos con cierta frecuencia nuestro algoritmo de busqueda de un pedido, posiblemente cada vez que saquemos una nueva versión de nuestra aplicacion.
- Finalmente, lo que más cambia, es la vista. De hecho, un misma de una librería virtual suele darnos posibilidad de varias presentaciones. El modelo y el controlador serían los mismos, pero habría varias vistas distintas.

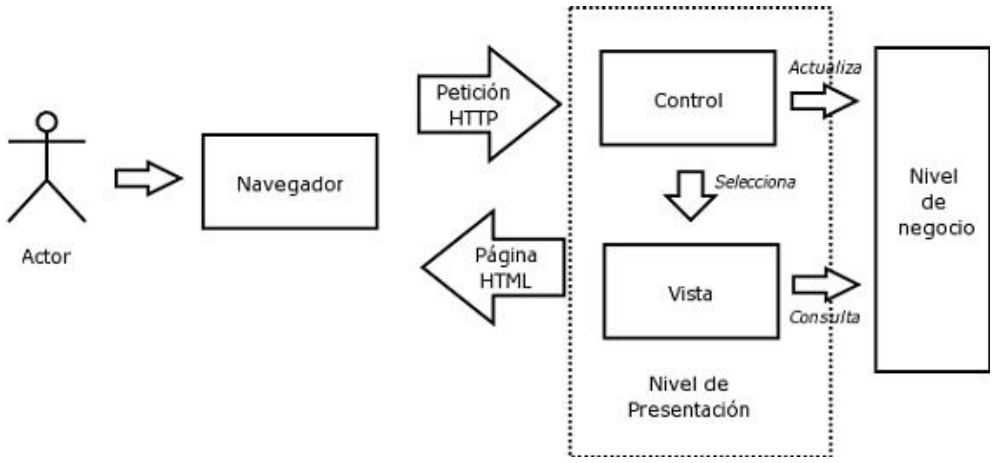
4.3.1.1 CAPA VISTA

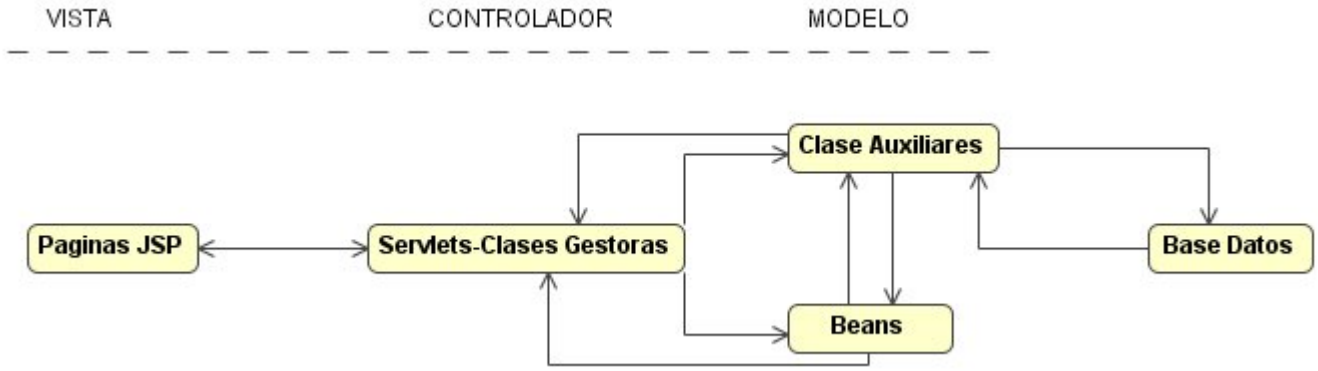
Para esta capa utilizaremos el “Modelo Vista Controlador” o “Modelo 2-Model 2”, donde integramos la utilización de paginas web dinamicas JSP y Servlets; las paginas JSP,para la capa de presentacion ; Y los Servlets para la capa de procesamiento y como responsable de procesar las peticiones asi como la creación de cualquier objeto beans.

En nuestro caso las clases gestoras, que son Servlets, son las que actuan como controladoras de la web y son las encargadas de comunicarse con las clases auxiliares que son las que se comunican con los datos almacenos.

En base al lenguaje de programación Java han surgido estas tecnologías para construcción de páginas dinámicas. Los Servlets son clases Java que pueden ser llamadas como cualquier página, y devuelven como respuesta una nueva página HTML o WML. Por otra parte, la tecnología JavaServer Pages permite hacer páginas HTML o WML que contienen código Java embebido. A través de estos segmentos de código las páginas en coordinación con los Servlets interactúan con los objetos de lógica de negocio para obtener los datos a presentar o cambiar. En el marco de este modelo, los "controladores" se materializan como Servlets y las "vistas" como páginas. La combinación de lógica y control en Java, con Servlets y JSP, todo dentro del marco de MVC, es conocida coloquialmente como "Model 2", y ha probado ser un ambiente efectivo para el desarrollo de aplicaciones empresariales de media y gran envergadura. A partir de la especificación 2.2 de Servlets y 1.1 de JSP, se ha unificado en gran medida los mecanismos de configuración e implementación de aplicaciones entre las distintas empresas que producen contenedores de Servlets/JSP. Esto permite que el lado servidor de la capa de presentación pueda ser instalada sobre varios servidores de Servlets/JSP, incluyendo el que vamos a usar Apache Tomcat.
Normalmente nuestra aplicación seguira este flujo de trabajo:

1. El usuario interactúa con la interfaz de alguna manera (ej. presionando un botón, enlace)
2. El controlador recibe (por parte de los objetos de la interfaz-vista) la notificación de la acción solicitada por el usuario
3. El controlador accede al modelo, posiblemente actualizando los datos enviados por el usuario
4. El controlador delega a los objetos de la vista la tarea de desplegar la interfaz de usuario
5. La vista usa el modelo para generar la interfaz apropiada para el usuario donde se refleja los cambios en el modelo. Nota: En algunas implementaciones la vista no tiene acceso directo al modelo, dejando que el controlador envíe los datos del modelo a la vista.
6. La interfaz espera por nuevas interacciones de usuario para iniciar nuevamente el ciclo.





Para el diseño de las clases gestoras en nuestro caso Servlets (aunque se podria haber utilizado otras tecnicas como “Apache Struts” o “Java Server Faces”) usaremos un patron de diseño,tal como se especifica en documento original de Sun:

http://java.sun.com/blueprints/corej2eepatterns/Patterns/index.html

Un patrón de diseño es una abstracción de una solución en un nivel alto. Los patrones solucionan problemas que existen en muchos niveles de abstracción. Hay patrones que abarcan las distintas etapas del desarrollo; desde el análisis hasta el diseño y desde la arquitectura hasta la implementación.

Mediante el uso de estos patrones de diseño conseguimos soluciones comunes a problemas diversos,unificando la gestión del codigo y del sistema, siendo esto sobre todo mucho mas claro de enteder, mantener y sobre todo mas facil detectar posibles errores o escritura de código superfluo.

En este contexto, ¿ qué es lo que nos han proporcionado los patrones de diseño? Estos elementos nos han dado la posibilidad de organizar nuestras clases en estructuras comunes y bien probadas; modificando el sistema para mejorar su flexibilidad y extensibilidad. En otras palabras, incrementando la facilidad para adaptar el software a los cambios de especificación e incrementando su posible reutilización. De alguna forma, los mismos beneficios producidos por la programación orientada a objetos pero mejorando aún más la calidad del diseño, y por lo tanto el software en sí.

El 8 de Marzo del 2001, el Centro Java de Sun publicó (como beta) un catálogo de patrones .
Los patrones J2EE describen típicos problemas encontrado por desarrolladores de aplicaciones empresariales y proveen soluciones para estos problemas. En esencia, estos patrones contienen las mejores soluciones para ayudar a los desarrolladores a diseñar y construir aplicaciones para la plataforma J2EE. Aunque estos patrones son representados desde un nivel lógico de abstracción, todos contienen en sus descripciones originales en la web de Sun, varias estrategisa que ofrecen una gran cantidad de detalles para su implementación.

Dentro de todos los posibles patrones de diseño especificados en el documento original de Sun, eligiremos el especificado en:

http://java.sun.com/blueprints/corej2eepatterns/Patterns/FrontController.html

Front Controller

Un objeto que acepta todos los requerimientos de un cliente y los direcciona a manejadores apropiados. El patrón Front Controller podría dividir la funcionalidad en 2 diferentes objetos: el Controller y el Dispatcher. En ese caso, El Controller acepta todos los requerimientos de un cliente y realiza la autenticación, y el Dispatcher direcciona los requerimientos a manejadores apropiada.

Centralizar el control en el controlador y reduciendo la lógica de negocios en la vista permite reutilizar el código entre peticiones. Es una aproximación preferible a la alternativa de embeber código en varias vistas porque esta aproximación trata con entornos más propensos a errores, y de reutilización del tipo copiar-y-pegar.

Aunque el patrón **Front Controller** sugiere la centralización del manejo de peticiones, no limita el número de manejadores en el sistema. Nuestra aplicación podra utilizar varios controladores , cada uno mapeado a un conjunto de servicios distintos.

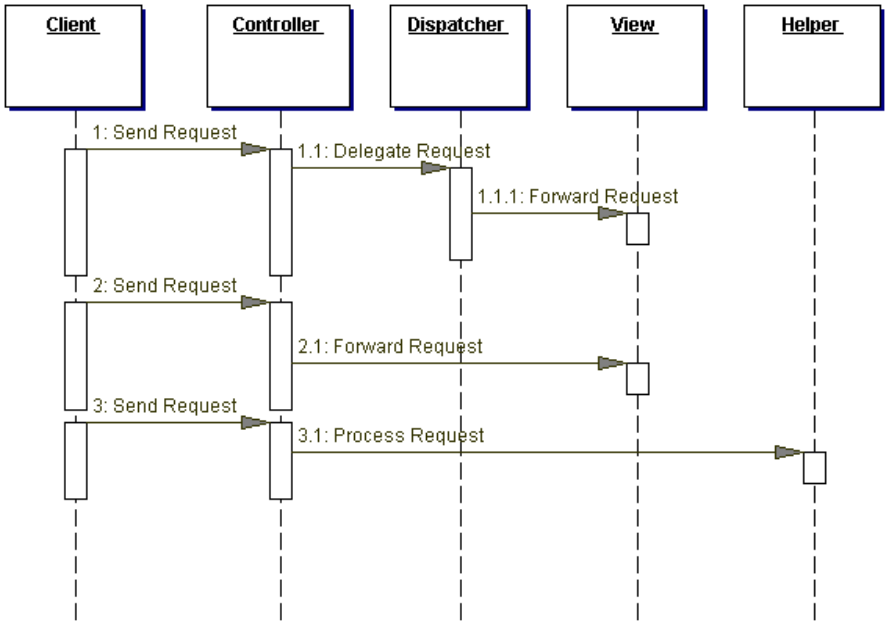
Tiene la principal ventaja de el poder centralizar en un solo punto, servicios como la gestión de conexiones a base de datos, comprobaciones de seguridad o gestión de errores favorecen que la aplicación sea mucho más robusta y aísla de todos estos aspectos al resto de componentes.

ademas:

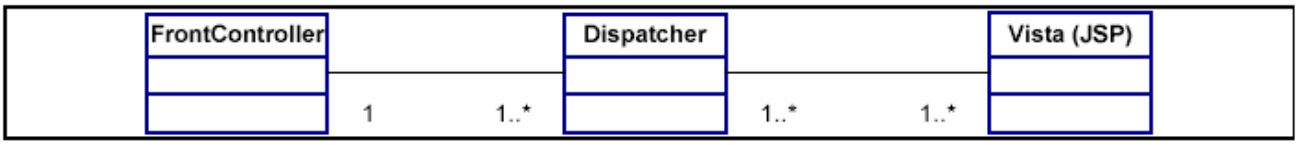
La gestión de las peticiones http (como por ejemplo www.MiAplicacion.com/controller?op=VerRegistro&id=58) a nuestra aplicación web puede ser centralizada o distribuida. Como ya hemos visto anteriormente, tener distribuida la gestión de peticiones en nuestras páginas jsp lleva a aplicaciones de baja calidad, en las que generamos mucho código similar y distribuido por todas nuestras páginas (vistas).

Solución: Usar un controlador como punto inicial para la gestión de las peticiones. El controlador gestiona estas peticiones, y realiza algunas funciones como: comprobación de restricciones de seguridad, manejo de errores, mapear y delegación de las peticiones a otros componentes de la aplicación que se encargarán de generar la vista adecuada para el usuario.
Centralizando los puntos de decisión y control, el controlador ayuda a reducir la cantidad de código java que se encuentra embebido en nuestras páginas jsp (a este código se le suele llamar Scriptlet). Lo que hacemos es centralizar control en el controlador y reducir lógica de negocio en las vistas (un modelo ejemplar de aplicación del metapatrón 'encapsulator/centralizer').

Esquema de trabajo del patron:



esta patron de diseño consta de :



Controller

Un controlador (las propias clases gestoras-Servlets), dependiendo de la pagina jsp en que nos encontremos se invocara a uno u otro , en nuestro caso concreto a una clase gestora u otra con la técnica que describiré posteriormente.

Este controlador como el punto inicial de contacto para manejar las peticiones. El controlador maneja el control de peticiones, incluyendo la invocación de los servicios de seguridad como la autentificación y autorización, delegar el procesamiento de negocio, controlar la elección de una vista apropiada, el manejo de errores, y el control de la selección de estrategias de creación de contenido dinámico JSP.

Dependiendo de la gestión que realizara el controlador usaremos un Servlet u otro, en nuestro sistema tenemos el siguiente catalogo de Servlets (las clases gestoras)

- GestorLogin
- GestorCompra
- GestorBeans
- GestorAdministración

Para que nuestro contenedor web Tomcat pueda trabajar con estos Servlets deberan estar definidos dentro del fichero *web.xml* que reside en la carpeta WEB-INF de nuestra aplicación.

A este fichero se le denomina descriptor de despliegue, y nos indica los Servlets disponibles en nuestra aplicación aparte de mas información util para Tomcat como cual es la pagina indice,…

Ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-
app_2_4.xsd">
  <servlet>
    <servlet-name>GestorLogin</servlet-name>
    <servlet-class>GestorLogin</servlet-class>
  </servlet>
 <servlet-mapping>
    <servlet-name>GestorLogin</servlet-name>
    <url-pattern>/GestorLogin</url-pattern>
  </servlet-mapping>
   <session-config>
    <session-timeout>
          30
     </session-timeout>
  </session-config>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

Típicamente, un controlador se coordina con un componente *dispatcher*.

Dispatcher

Un dispatcher es el responsable del manejo de la vista y de la navegación, controlando la elección de la siguiente vista que se le presentará al usuario, y proporcionando el mecanismo para dirigir el control a ese recurso.
Un dispatcher se puede encapsular dentro de un controlador o se puede separar en otro componente que trabaja de forma coordinada. El dispatcher puede proporcionar un re-envío estático de la vista o un mecanismo de re-envío más sofisticado.
El dispatcher utiliza un objeto RequestDispatcher (soportado en la especificación Servlet) y encapsula algún procesamiento adicional.

Así, un *dispatcher* elige la siguiente vista por el usuario y dirige el control al recurso. Los dispatchers en nuestro son invocados mendiante el metodo ***forward*** y la inserción en el ***resquest*** de los atributos necesarios.

Ejemplo de esto:

```
//dentro de la clase-servlet GestorAdministracion para hacer el salto si la identificación es //correcta a
administracion.jsp

RequestDispatcher dispatcher = request.getRequestDispatcher("/administracion.jsp");
     if (dispatcher != null)
          dispatcher.forward(request, response);
```

View

Las vistas, en nuestro caso concreto son la paginas JSP, que se crean dinamicamente, por ejemplo para mostrarnos contenidos de una tabla de una base de datos o datos de un registro concreto.
Ejemplo de esto

```
    <%--
         ejemplo de cómo mostramos todas las categorias que tenemos en la base de datos
         esto se implementa dentro de index.jsp ( menu izquierdo de acceso a libros por categorias )
      --%>
</table>
    <table width=100% cellspacing="0" cellpadding="0" border="0">

     <%-- Cabeceras --%>
     <tr bgcolor="#6A63F3">
       <td colspan=100% height="20"><font face="Verdana,Helvetica" size="3"
          color="#000000"><b>Categorias</b>
       </font></td>
     </tr>

     <%-- Filas --%>
      <%--
         Recogemos los valores de cada categoria
      --%>

     <c:forEach var="fila" items="${requestScope.categorias}">
     <tr>
       <td width="5%"><font face="Verdana,Helvetica" size="2">
        ${fila.nombrecategoria}</font></td>
     </tr>
     </c:forEach>

   </table>
```

La paginas JSP se comunica directamente con alguna de las clases gestoras bien por :

- El ***action*** de la etiqueta html form,por ejemplo pulsado un boton y captando el Servlet dicha pulsación y gestionando que es lo que debe de realizar
- O bien Por un link html a traves de la etiqueta html ***href*** para el paso de parametros de la pagina JSP al Servlet.

Ejemplo de esto:

```
//ejemplo de codigo por action de form desde index.jsp en Servlet GestorLogin.java
//cuando se pulsa el boton “entrar” en index.jsp para identificarnos como administrador web

String boton = request.getParameter("boton"); //datos del boton del form html
if( boton.equals("entrar") //controlo si se ha pulsado el boton entrar de index.jsp
      {
          ….
    }
```

En nuesta Web usaremos las siguientes paginas JSP:

PAGINA JSP	***ESQUEMA DE ITERACCIÓN***
index.jsp pagina indice de la web	→ GestorLogin → administracion.jsp → GestorCompra → mensaje.jsp → GestorBean → categoria.jps datoslibro.jsp
datoslibro.jsp muestra los datos completo de un libro	→ GestorCompra → mensaje.jsp
categorias.jsp muestra todos los libros de una categoria	→ GestorCompra → mensaje.jsp → GestorBean → datoslibro.jsp
micompra.jsp muestra el carrito de la compra	→ GestorCompra →realizarcompra.jsp
realizarcompra.jsp para introducir datos del cliente una vez realizada la compra	→ GestorCompra → mensaje.jsp
administracion.jsp pagina principal de adminsitracion desde la cual se accede a todas las de gestion de libros y categorias	→ GestorAdministrador → altascategoria.jsp altaslibros.jsp listarcategorias.jsp listarlibros.jsp consultarcategoria.jsp consultarlibros.jsp
altascategoria.jsp para dar de alta una categoria	→ GestorAdministrador →mensaje.jsp
altaslibros.jsp para dar de alta un libro	→ GestorAdministrador →mensaje.jsp
listarcategorias.jsp nos muestra todas la categorias del sistema	→ GestorAdministrador →mensaje.jsp
listarlibros.jsp nos muestra todos los libros del sistema	→ GestorAdministrador →mensaje.jsp
consultarcategoria.jsp vemos datos una categoria en cuestion	→ GestorAdministrador →mensaje.jsp
consultarlibros.jsp vemos datos de un libro en cuestion	→ GestorAdministrador →mensaje.jsp
mensaje.jsp mensajes de control varios	

Helper

Estas clases son ayudantes para la gestión adecuada de los controladores en nuestro caso se encargan principalmente de la comunicación con la base de datos,son las que nosotros hemos llamado clases auxiliares.

4.3.1.2 CAPA MODELO

Nuestra capa modelo representa como clases todas la clases de entidad que nos son útil , viene representada por las entidades principales del sistema:

- Cliente
- Libro
- Categoría
- Pedido
- LineaPedido

Mediante el uso de estos Beans, se puede simplificar el acceso por JSP y ayudan a conseguir un puente entre la lógica de negocio y las vistas, en principio todos nuestros Beans, tendrán un constructor si parámetros y sus correspondientes métodos get y set.

De esta manera se puede acceder a los objetos desde paginas JSP con la sintaxis:

${objeto.propiedad}

Ver ejemplo anterior en apartado de *view*

En nuestra aplicación todos estos objetos son creados por las correspondieste clases gestoras, limitandose las paginas JSP a acceder a ellos.

Todo ello por supuesto con las facilidades de intercomunicación entre paginas JSP y Servlets que proporciona el lenguaje de expresion en JSP 2.0 y los objetos implicitos como *requestScope*

Por ejemplo:

```
//en el Servlet GestorBean puede haber una linea como esta:
//que recupera en todaslascategorias a traves de las clases auxiliares todas las categorias de la base de datos

request.setAttribute(“categorias”,todaslascategoria);

//y en la pagina JSP se reciben asi:

<c:forEach var="fila" items="${requestScope.categorias}">
      <tr>
        <td width="5%"><font face="Verdana,Helvetica" size="2">
          ${fila.nombrecategoria}</font></td>
      </tr>
</c:forEach>
```

Igualmente incluimos dentro de esta capa el carrito de la compra , que manejaremos como una tabla en memoria temporal y el control de sesiones a traves de cookies que llevara a cabo las clases gestoras.

4.3.1.3 CAPA DATOS

Todo el acceso a los datos se encuentra encapsulado y solamente se puede acceder a ellos a través de las clases auxiliares que hemos definidos, es decir todo el código JDBC de comunicación directa con la base de datos y las tablas se encuentra implementado en estas clases, tan solo las clases gestoras a través de los métodos definidos en estas clases auxiliares podrán consultar o actualizar la base de datos.

Para una mejor gestión de las conexiones con la base de datos utilizamos el servicio de conexiones (Connection Pool) que nos ofrece JDBC y Tomcat, de esta manera las conexiones con una base de datos solo se abren una vez y se mantienen en memoria para poder trabajar tantas veces queramos con ellas sin necesidad de realizar conexiones nuevas.

Para ello usamos las características *JNDI* (interfaz de nombres y directorios para java) del paquete *javax.naming.**

Habitualmente se creara la conexión en el método init de algunos de los Servlets, preferiblemente el primero que se invoque y se recuperara en los demás.

El servicios de conexiones no los ofrece el propio contenedor web Tomcat y podemos trabajar con el a través de la interfaz *Datasource* y a través del método *getConnection* de *Datasource.*

Para lograr todo esto debemos configurar nuestra base de datos en Tomcat en el fichero *server.xml* del directorio conf de la instalación de Tomcat, por ejemplo este podria ser parte de este fichero para nuestra aplicación:

```
<Resource
     name="jdbc/libreria"
     type="javax.sql.DataSource"
     password="admin"
     driverClassName="org.gjt.mm.mysql.Driver "
     maxIdle="2"
     maxWait="5000"
     username="root"
     url="jdbc:mysql://localhost:3306/libreria"
     maxActive="4"/>
```

y el fichero *context.xml* de la carpeta META-INF de la aplicación este podría ser el fichero:

```
<?xml version="1.0" encoding="UTF-8"?>
<Context path="/libreriavirtual">
     <ResourceLink name="jdbc/libreria" type="javax.sql.DataSource" global="jdbc/libreria"/>
     <Logger className="org.apache.catalina.logger.FileLogger" prefix="libreriavirtual" suffix=".log"
timestamp="true"/>
 </Context>
```

Este podria ser el codigo del metodo *init* de alguno de nuestras clases gestoras para poder trabajar con esta conexión:

```
private DataSource pool;

public void init(ServletConfig config) throws ServletException
  {
    super.init(config);
    try
    {
      // Recuperar el contexto inicial
      Context ctx = new InitialContext();
      // Referencia al servicio de conexiones
      pool =  (DataSource)ctx.lookup("java:comp/env/jdbc/libreria");
    }
    catch (Exception e)
    {
      throw new ServletException(
         "Imposible recuperar java:comp/env/jdbc/libreria", e);
    }
  }
```

Igualmente este seria el codigo para trabajar con sentencias SQL dentro por ejemplo de la clase auxiliar BaseDatos.java

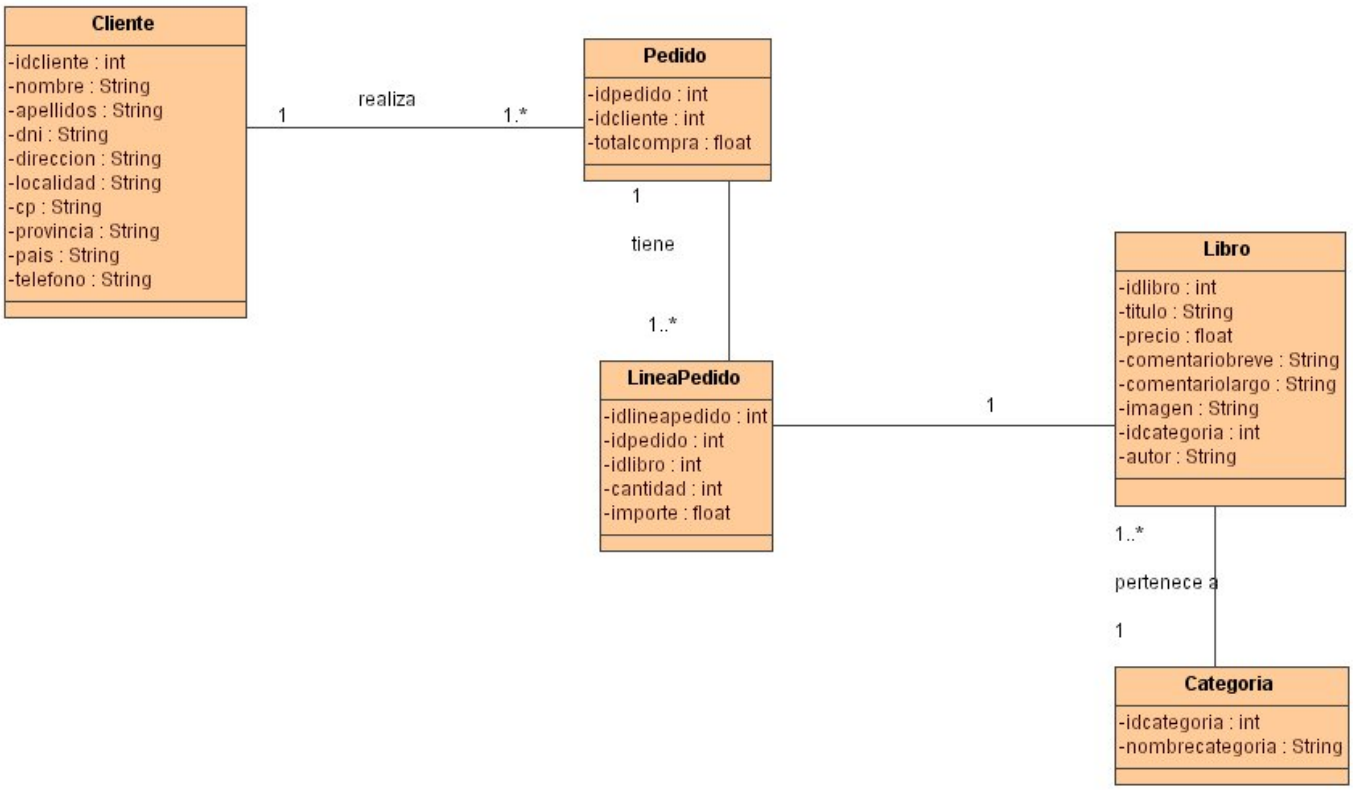
```
/* Crea una nueva instancia de BaseDatos */
    public BaseDatos(DataSource servCon)
    throws java.sql.SQLException, javax.servlet.ServletException
    {
        synchronized (servCon)
        {
            conexión = servCon.getConnection(); // recuperar la conexión
        }
        if(conexión == null)
            throw new ServletException("Problemas con la conexion");

        // Crear una sentencia SQL
        sentenciaSQL = conexión.createStatement(
                    ResultSet.TYPE_SCROLL_INSENSITIVE,
                    ResultSet.CONCUR_UPDATABLE);

        System.out.println("\nConexión realizada con éxito.\n");
    }
```

Por supuesto dentro de las paginas JSP no se indicara ninguna sentencia JDBC sino que se accedera a los datos con las tecnicas ya comentadas.

4.3.2 Diagrama de Clases

4.3.2.1 CLASE DE ENTIDAD-BEANS

Hemos identificado las siguietes clases de entidad-beans:



Como se puede observar hemos identificado 5 clases de entidad:

Cliente: con los datos del cliente que realiza la compra.
Libro: con los datos del libro a comprar.
Categoria: categoría a la que pertenece el libro.
Pedido: datos de la cabecera del pedido que se realiza.
LineaPedido: datos de detalle del pedido que se realiza.

Vemos como un Cliente se asocia a un Pedido, donde un Cliente puede hacer un Pedido con muchos lineas de pedido ,cada linea de pedido se asocia a un libro.

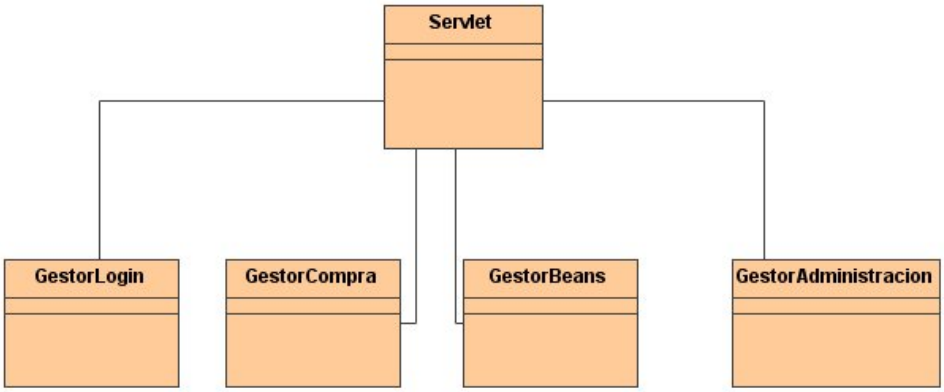
También vemos como un Libro tiene asociada una sola Categoría, aunque una Categoría puede tener muchos Libros.

Con respecto al pedido , vemos que es una asociación maestro-detalle tipica donde un Pedido puede tener asociadas varias LineasPedido.

Todas estas clases deben de ser persistentes, por lo tanto existiran como tablas de una base de datos.

4.3.2.2 Clases Gestoras

Estas clases se implementaran como Servlets, y nos servira para gestionar toda la aplicación.



GestorLogin: nos servira para gestionar la identificación de manera adecuada del administrador en el sistema.

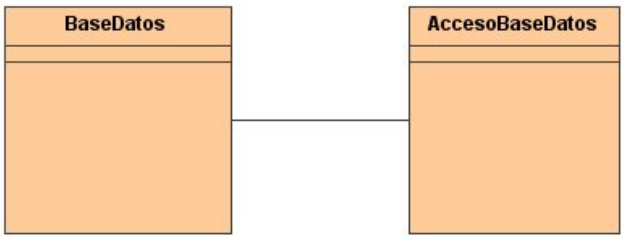
GestorCompra: nos servira para gestionar todo lo relacionado con las compras que van a realizar los clientes, incluido el carrito de la compra.

GestorBeans: en este caso esta clase nos ayudara a gestionar la relación de las clases con los Beans de entidad.

GestorAdministración: sirve para gestionar la administración de nuestra librería, altas,bajas,consultas todas la tareas de mantenimiento sobre nuestro sitio web.

4.3.2.3 Clases Auxiliares

Mediante estas clases podremos gestionar desde nuestras clases gestoras la información contenida en nuestra base de datos.

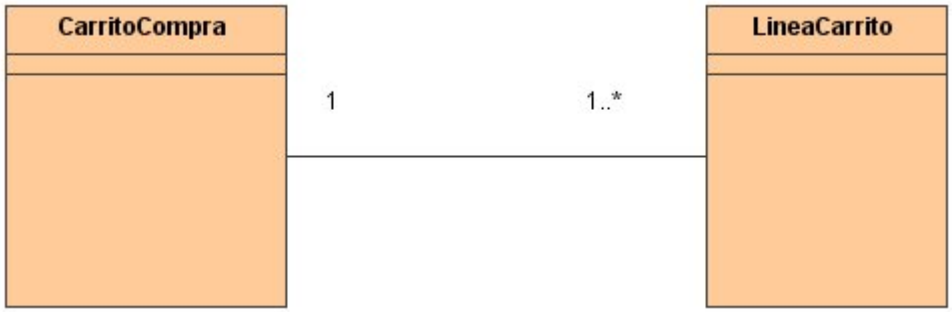


BaseDatos: mediante esta clase podemos comunicarnos con la base de datos, permite conectarnos a ella introducir o modificar datos en las tablas, son las instrucciones JDBC-JAVA para comunicarnos con MYSQL

AccesoBaseDatos: nos sirve como intermediario directo entre los Servlets y la base de datos, para poder optimizar de una mejor manera los recursos,sobre todo en creación de objetos para acceder a los datos, encapsula el acceso a los datos mendiante objetos, para que las nistas no usen instrucciones SQL.

4.3.2.4 Clases Carrito Compra

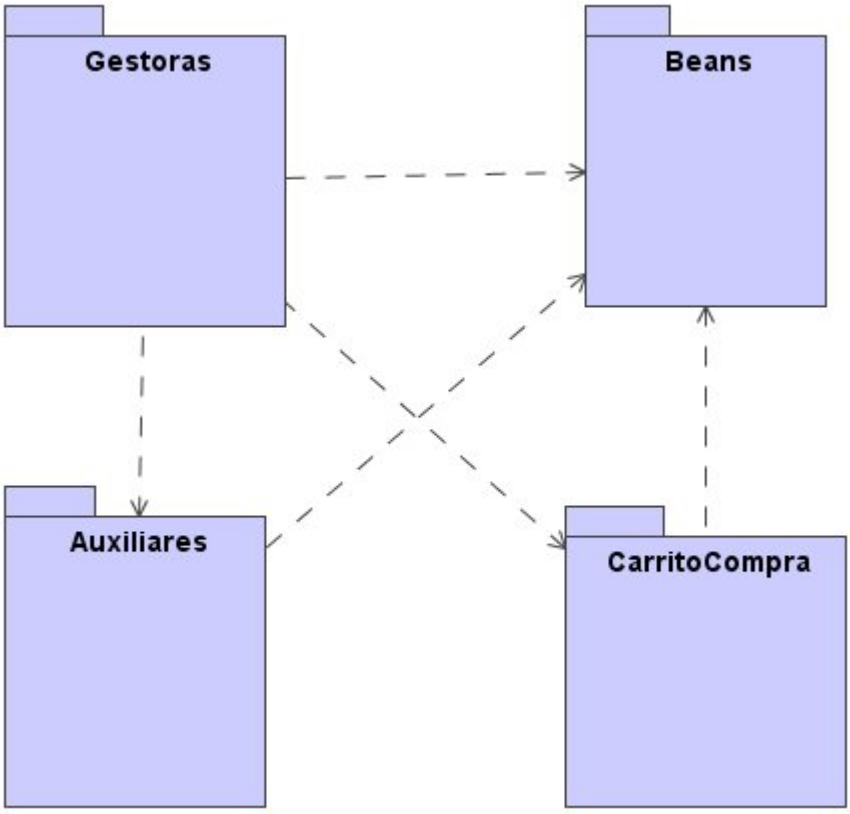
Mediante estas clases manejamos el carrito de la compra necesario para nuestra aplicación, se creara un CarritoCompra por cada sesión abierta por un cliente web.



CarritoCompra: clase principal del carrito de la compra, no es mas que un ArrayList que contiene todos los objetos LineaCarrito que forman parte del carrito de la compra.

LineaCarrito: representa un objeto que hemos comprado en nuestra Librería virtual, por cada compra pueden existir muchos objetos comprados.

4.3.3 Diagrama de paquetes



4.3.4 Persistencia

Nos interesa tan solo hacer persistentes las Clases de entidad y para conseguir que nuestras clases sean persistentes ,nuestra ideas es mapear nuestros objetos al mundo relacional, tendremos que trabajar con una base de datos para conseguirlo, lo realizaremos a traves de JDBC y usaremos como gestor de base de datos MYSQL, para ello instalaremos el driver JDBC correspodiente es este caso:

mysql-connector-java-3.1.10-bin.jar

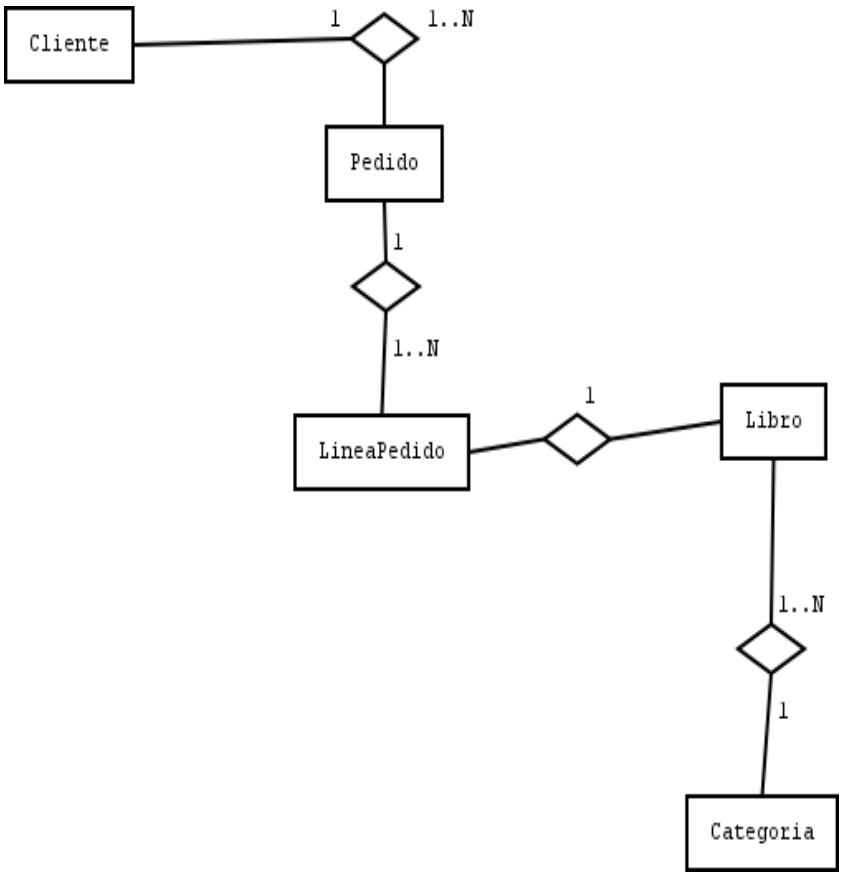
usaremos para ello la clase de este driver: *org.gjt.mm.mysql.Driver*

Todo ello bajo un Pool de Conexiones que hemos comentado en el apartado de Patrones de Diseño.

Podriamos haber elegidos otras soluciones al estilo de Hibernate, pero como tenemos un modelo de clases no excesivamente complejo hemos optado por un mapeo directo entre ambos mundos.

Fundamentalmente nos interesara que sean persistentes las clase de entidad que hemos definido para ello trasformamos el diagrama de clases en el siguiente diagrama Entidad/Relación.

4.3.5 Modelo Entidad/Relación



4.3.6 Diseño de la base de datos

El script de creación de la Base de Datos y sus tablas es el siguiente:

```
CREATE DATABASE libreria;

USE libreria;

CREATE TABLE cliente (
  idcliente int(11) NOT NULL auto_increment,
  nombre varchar(100),
  apellidos varchar(100),
  dni varchar(10),
  direccion varchar(100),
  localidad varchar(100),
  cp varchar(5),
  provincia varchar(100),
  pais varchar(100),
  telefono varchar(9),
  PRIMARY KEY  (idcliente)
);


CREATE TABLE categoria (
  idcategoria int(11) NOT NULL auto_increment,
  nombrecategoria varchar(100),
  PRIMARY KEY  (idcategoria)
);


CREATE TABLE libro (
  idlibro int(11) NOT NULL auto_increment,
  titulo varchar(100),
  autor varchar(100),
  precio float,
  comentariobreve varchar(255),
  comentariolargo varchar(255),
  imagen varchar(255),
  idcategoria int(11),
  PRIMARY KEY  (idlibro),
  FOREIGN KEY  (idcategoria) REFERENCES categoria(idcategoria)
);

CREATE TABLE pedido (
  idpedido int(11) NOT NULL auto_increment,
  idcliente int(11),
  totalcompra float,
  PRIMARY KEY  (idpedido),
  FOREIGN KEY  (idcliente) REFERENCES cliente(idcliente)
);


CREATE TABLE lineapedido (
  idlineapedido int(11) NOT NULL auto_increment,
  idpedido int(11),
  idlibro int(11),
  cantidad int(11),
  importe float,
  PRIMARY KEY  (idlineapedido),
  FOREIGN KEY  (idpedido) REFERENCES pedido(idpedido),
  FOREIGN KEY  (idlibro) REFERENCES libro(idlibro)
);
```

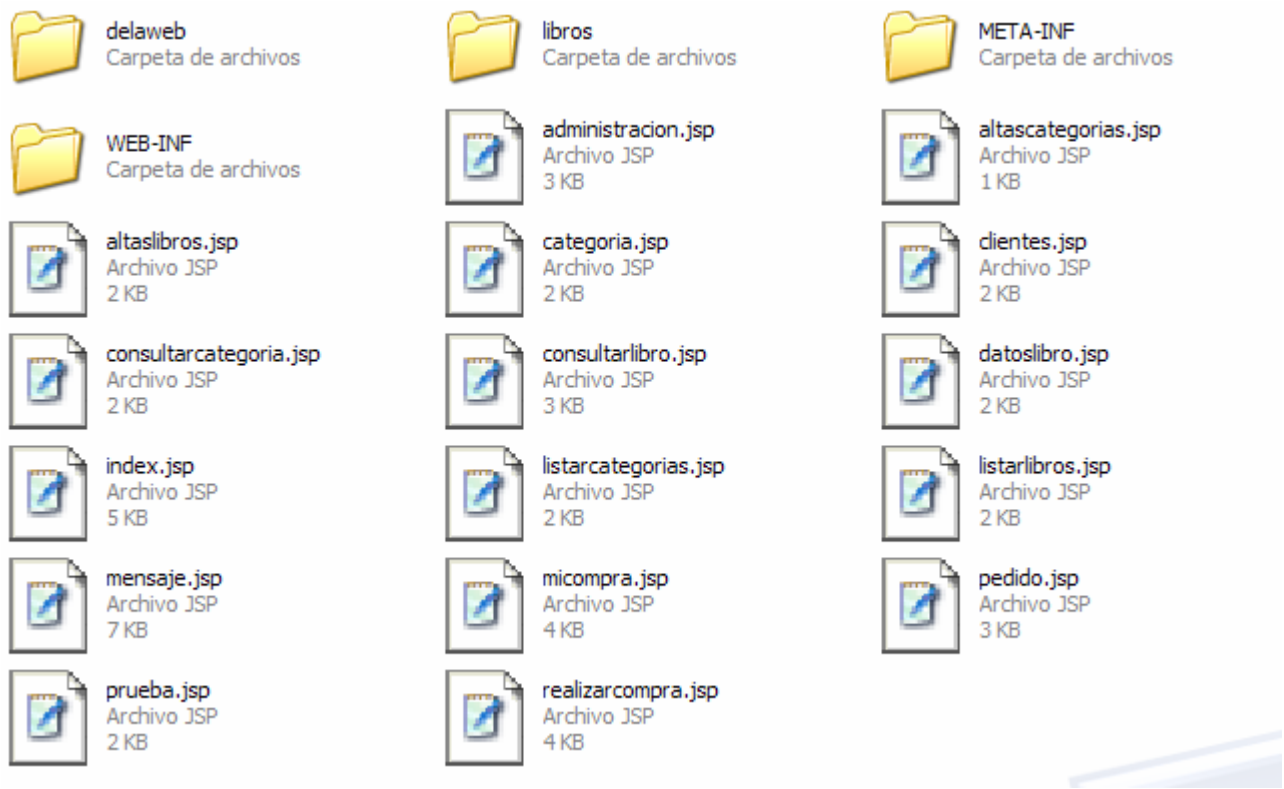
4.4 Capitulo 4 ,Implementación

4.4.1 Estructura

Nuestra aplicación esta formada por :

- Clases JAVA
- Clases JAVA – Servlets
- Paginas JavaServer Pages – JSP
- Gráficos e imágenes

la estructura de carpetas de la aplicación una vez instalada bajo el servidor TOMCAT es la siguiente (mantiene la estructura estandar de TOMCAT):



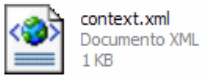
como se ve dentro la carpeta principal de la aplicación nos encontramos con nuestro catalogo de paginas JSP

dentro de la carpeta delaweb se encuentran aquellos gráficos e imágenes necesarios para la visualización de nuestra aplicación , incluido la imagen por defecto de los libros que demos de alta en el sistema.

dentro de la carpeta libros se encuentran las imágenes de la portada de los libros que hemos dado de alta en el sistema y hemos indicado esta ruta al dar el alta.

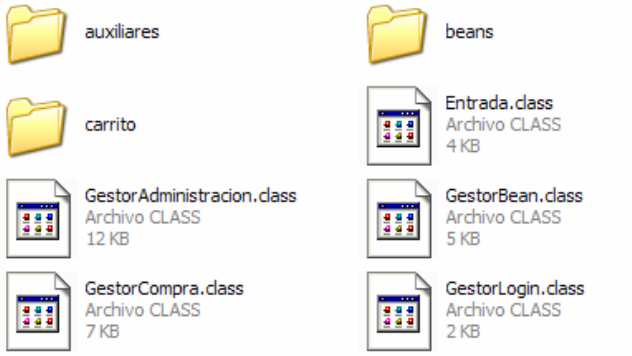
las carpetas META-INF y WEB-INF son las carpetas principales de la aplicación siguiendo la estructuras estandar de TOMCAT.

dentro de META-INF: no encontramos con el fichero de contexto de TOMCAT *context.xml*, ya comentado en un apartado anterior de esta memoria.

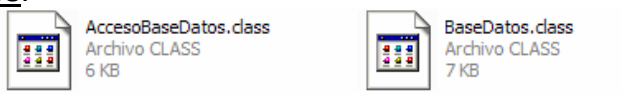


dentro de WEB-INF: nos encontramos con el fichero de mapeo de Servlets *web.xml*, la carpeta classes, donde estan las clases JAVA de la aplicación y la carpeta lib donde estan las librerías no JDBC necesarias para la aplicación , en nuestro caso la librería de etiquetas standard JSTL.

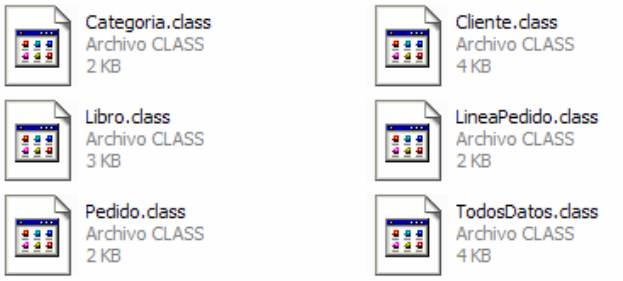
dentro de classes:



dentro de auxiliares:



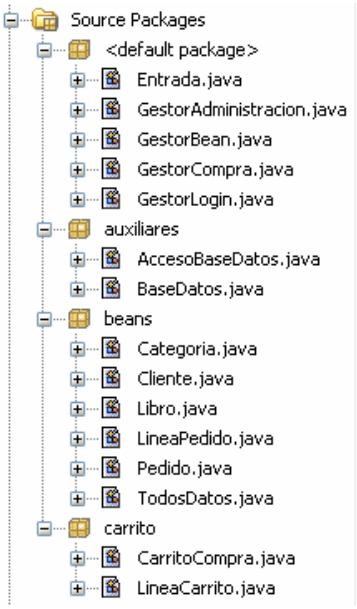
dentro de carrito:



dentro de beans:

lib:

4.4.2 Catalogo de clases JAVA



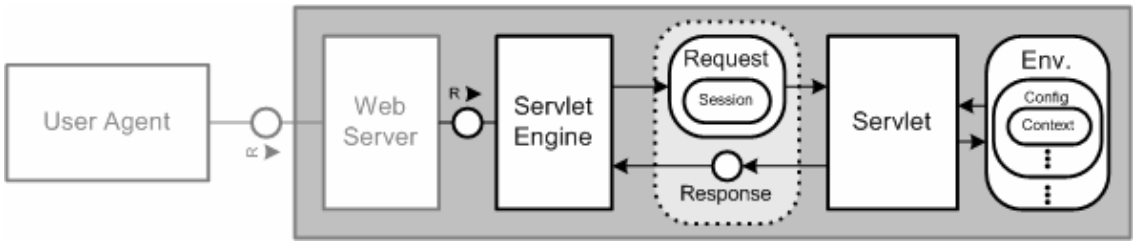
dentro de las clases JAVA de los paquetes auxiliares,beans y carrito tenemos estas clases con la siguiente misión:

CLASE	PAQUETE	MISION
AccesoBaseDatos.java	auxiliares	Encapsula el acceso a la base de datos, mediante Beans y diversos métodos para su utilización en Servlet y JSP.
BaseDatos.java	auxiliares	Clase que accede directamente mediante instrucciones SQL a las tablas de las Base de Datos, se usa el acceso mediante JDBC con el driver MYSQL oportuno.
Categoria.java	beans	Representa un objeto categoria dentro del sistema , después lo haremos persistente en la base de datos
Cliente.java	beans	Representa un objeto cliente dentro del sistema , después lo haremos persistente en la base de datos
Libro.java	beans	Representa un objeto libro dentro del sistema , después lo haremos persistente en la base de datos
LineaPedido.java	beans	Representa un objeto lineapedido dentro del sistema , después lo haremos persistente en la base de datos
Pedido.java	beans	Representa un objeto pedido dentro del sistema , después lo haremos persistente en la base de datos
TodosDatos.java	beans	Representa en forma de objeto los datos que obtenemos de la consulta SQL con todos los datos de un pedido en concreto del que conocemos su idpedido
CarritoCompra.java	carrito	Representa el objeto Carrito de la compra que no es mas que un ArrayList con las líneas que representa los datos del producto que hemos comprado
LineaCarrito.java	carrito	representa los datos de una línea del carrito de la compra por cada carrito de la compra puede existe de 1 a muchas líneas como esta como objetos del ArrayList que representa el CarritoCompra esta línea se hace después persistente con LineaPedido

4.4.3 Catalogo de Servlets

Nuestras clases gestoras estan representas por Servlets de JAVA, dentro las especificaciones 2.4 de J2EE para los servlets soportadas por APACHE TOMCAT 5.5.7

Un servlet en un programa que se ejecuta en el contenedor web de un servidor de aplicaciones como es en nuestro caso TOMCAT , los clientes web los invocan a través del protocolo http, en concreto los invoca nuestras paginas JSP.



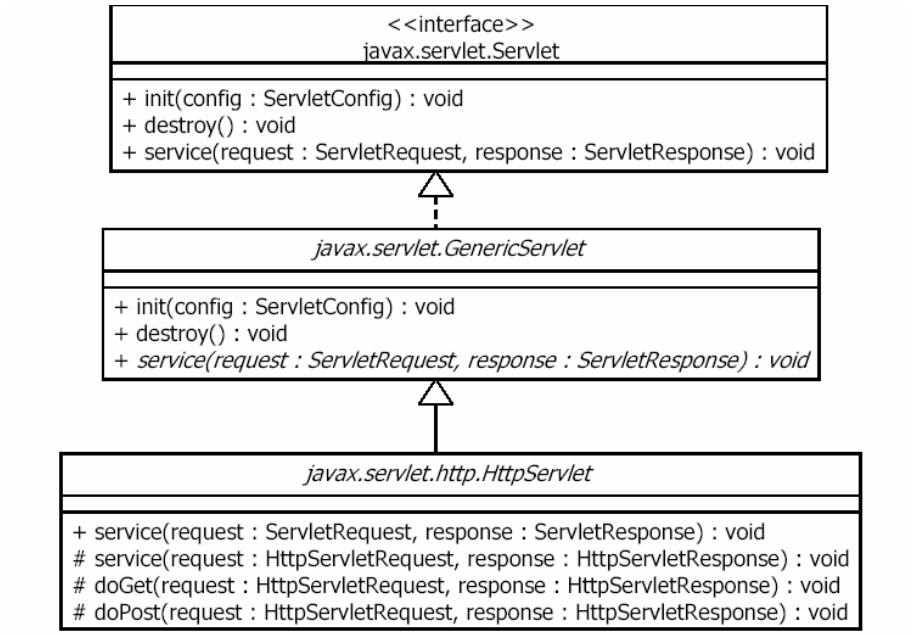
un servlet acepta peticiones de un cliente, procesa la información y la devuelve al mismo cliente.

los servlet no es más que la alternativa de SUN a la tradicional programación para Internet con CGI, teniendo con estos enormes ventajas como:

- son independientes de plataforma al estar escritos en JAVA
- consumen menos recursos
- son mas rápidos
- son mas seguros y portables
- no requiere de soporte JAVA en el navegador web.

En una maquina virtual JAVA solo existe una instancia de cada servlet que se programe, y en consecuencia puede recibir peticiones concurrentes.

La estructura básica de un servlet, nos es mas que un objeto de una clase del API de JAVA, en concreto implementa la interfaz Servlet.



El servlet se instancia al iniciarse el contenedor o al recibir la primera petición (configurable en *web.xml*)

El método init es el método al que se llama justo después de instanciar el Servlet, sólo una vez, nosotros lo utilizamos para recuperar el contexto del pool de conexiones.

La clase HttpServlet proporciona métodos para dar soporte a los métodos de HTTP 1.1:

public void **doGet**(HttpServletRequest req, HttpServletResponse resp) throws IOException, ServletException;
public void **doPost**(HttpServletRequest req, HttpServletResponse resp) throws IOException, ServletException;

para recoger datos enviados desde paginas web o en nuestro caso JSP, desde envíos GET y POST.

a traves de:

public String **getParameter**(String name)

podemos recoger cualquier parámetro de un formulario web de una pagina JSP

igualmente de:

public void **setAttribute**(String name, Object o)

podemos añadir un atributo a la petición

también podemos redirigir hacia una pagina JSP a través de :

getRequestDispatcher("/xxxx.jsp").forward(req,res);

Incluso podemos manejar sesiones a través del objeto HttpSession para almacenar datos a lo largo de la vida de una sesión

- Interfaz javax.servlet.http.HttpSession = abstracción del concepto de sesión
- Se usa como contenedor de atributos que representan el "estado" del cliente en el servidor
- Las sesiones se crean/recuperan de la interfaz HttpServletRequest, haciendo transparente al programador toda la creación/recuperación de cookies

Métodos para recuperar una sesión:

- public HttpSession getSession(boolean create)
  - Devuelve la sesión correspondiente al cliente y si no existe:
    - Si create es true, la crea.
    - Si create es false, devuelve null.
- public HttpSession getSession()
  - Igual a getSession(true)

CLASE	MISION
Entrada.java	clase gestora auxiliar que se carga por primera vez cuando se entra en la aplicación, también se carga cuando desde cualquier pagina se invoca al índice.
GestorBean.java	gestiona las pulsaciones sobre cualquier categoria en la pagina index.jsp para desviarlo a la pagina donde nos muestra todos los libros de esa categoria en cuestión también gestiona cuando se pulsa sobre el titulo de un determinado libro en index.jsp ,para saber mas datos sobre él.
GestorLogin.java	gestiona a través de este Gestor si los datos recogidos del formulario de identificación de index.jsp se corresponde con el login y el password del administrador de la web, si esto es así, desvio hacia administracion.jsp; se recogen los datos a través del post del formulario.
GestorAdministracion.java	se encarga principalmente de 2 tareas recoger los enlaces a través del método doget para saber donde hemos pulsado en adminsitracion.jsp y también gestiona todas las altas,consultas-modificaciones,eliminar de las tablas presente en la aplicación.
GestorCompra.java	Gestiona todo lo relacionado con las compras, cuando pulsamos comprar sobre algún libro en las distintas paginas jsp, también gestiona el carrito de la compra para mostrarlo, eliminar productos actualizarlo con cantidades nuevas, calcula el total del pedido y graba el pedido de forma persistente transformando el carrito en un pedido maestro-detalle todo el carrito de la compra se gestiona por sesiones en una sesión guardamos el carrito de la compra como un objeto de tipo Arraylist.

Todas estas clases aparecen perfectamente comentadas en el código fuente de la aplicación.

4.4.4 Catalogo de paginas JSP

Todas la paginas JSP que utilizo en la aplicación están perfectamente documentadas a lo largo de este documento tanto el la parte de análisis como de diseño, también están todas comentadas directamente en el código fuente de las mismas.

Las JavaServer Pages (JSP) nos permiten separar la parte dinámica de nuestras páginas Web del código estático. Para ello:

- Simplemente escribimos el HTML (o XML) regular de la forma normal, usando cualquier herramienta de construcción de paginas Web.
- Encerramos el código de las partes dinámicas en unas etiquetas especiales, la mayoría de las cuales empiezan con “<%” y terminan con “%>.

la principales ventajas de la utilización de paginas JSP son:

- Separación contenido dinámico-estático
- Complemento ideal de los Servlets
- Fundamental para el modelo de 3 capas
- Ahorrar escribir código HTML en los Servlets
- fácilmente integrable con XML

Nosotros para una mejor gestión de estas paginas usaremos la librería estándar de etiquetas JSTL (incluida en la aplicación) y la versión de la sintaxis 2.0 soportada por la versión de TOMCAT que vamos a usar.

para ello las primeras líneas de nuestras paginas JSP serán:

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>


<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
```

4.5 Capitulo 5, Valoración económica.

	Horas	Precio	Impote
Ingenieria del Software			
• Analisis	30	30€	900€
• Diseño	40	30€	1200€
• Implementación	50	30€	1500€
• Pruebas e Instalación	30	20€	600€
Software			
• Windows 2003			1200€
Hardware			
• Servidor PC-ATHLON64			2000€
• Conexión Internet ADSL2 IP Fija(anual)			1200€
• Infraestructura de red			600€
TOTAL PRESUPUESTO			9200€

4.6 Capitulo 6, Conclusiones.

En principio hemos alcanzado los objetivos propuestos, hemos analizado,diseñado e implementado una buena aproximación a lo que sería una aplicación real para una determinada empresa, consiguiendo para ello desarrollar un interfaz lo sufiencentemente intuitivo como para lograr que la curva de aprendizaje del manejo del producto sea realmente corta, hemos obtenidos unos buenos conocimientos en el desarrollo del lenguaje JAVA para aplicaciones de Internet y desarrollo visual con JSP.

Hemos logrado desarrollar un software tanto en su desarrollo como su implantación en el mundo real totalmente Open Source, sin tener por tanto el “pesado peso” de una política de licencias a nuestras espaldas, con el ahorro tan enorme en costes como podemos tener , como es fácilmente comprobable en el presupuesto de la aplicación, que aunque esta desarrollada bajo entorno Windows ,esta se podría haber desarrollado e implantado en un sistema LINUX, sin ningún tipo de problema, al ser todas las herramientas software utilizadas totalmente multiplataforma (Incluido J2EE).

Hemos conseguido diseñar la aplicación totalmente enfocada a nivel multicapa,logrando así cumplir con la mayoría de los objetivos que deseamos:
Fácil de mantener, fácil de corregir, fácil de compartir el desarrollo, libre de cargas,….

Hemos comparado otras herramientas y productos finales con el que hemos desarrollando , pensando que no tiene nada que envidiar a muchas soluciones comerciales que he visto ya implantadas.

Hemos cumpliendo el ciclo de vida del software , desarrollando este siempre basándonos en cada una de las fases que hemos realizado, sin desviarnos en excesos de los análisis y diseños en un principio planteado, facilitando todo esto una implementación con un corto tiempo de codificación.

La herramienta de desarrollo utilizada y aprendida a manejar durante el desarrollo de este Trabajo: NetBeans, ha sido una herramienta potente,sencilla y agradable de usar que ha favorecido también el desarrollo de una manera ideal del producto final.

Quizás la única pega y por falta material de tiempo no ha sido posible, el desarrollo del sistema utilizando algún Framework adecuado al estilo de Apache Struts, que después de documentarme al respecto (con algo de retraso,quizas) creo que hubiera sido una solución muy elegante sobre todo para no alargar demasiado la codificación de las clases gestoras.

También quizás se podría haber desarrollado un sistema de identificación de clientes no solo del usuario administrador para lograr que los clientes a través de un login y un password obtenidos en un proceso de alta, poder estos consultar sus pedidos y estado de los mismos.

5.ANEXOS

Quizás por el tiempo que ha sobrado al final del desarrollo de la fase de implementación, he logrado completar algo mas la parte de administración del producto, con opciones no comentadas en la fase de análisis y diseño, en concreto estas nuevas funcionalidades son:

- Consulta de todos los clientes con pedidos en el sistema
- Consulta de un pedido en concreto ya tramitado , sabiendo el identificador del mismo.

6.BIBLIOGRAFIA

- JAVA 2, Interfaces gráficas y aplicaciones para Internet; Javier Ceballos. Ed. RA-MA.
- JAVA 2, Manual de referencia; Herbert Schildt. Ed. Mcgraw-Hill
- Core Servlets and JavaServer Pages: Volume1 …;Marty Hall.Ed. Prentice Hall.
- Teach yourself J2EE; Martin Bond,… Ed. SAMS
- JAVA2 Enterprise Edition 1.4 ;James McGovern. Ed. Wiley Publishing,Inc.
- Manual JDBC. Grupo Eidos.
- Diseño orientado a objetos.Grupo Eidos

7. GUIA DE INSTALACION Y PRUEBAS

El producto se entrega como un fichero *.war* para su desplege en un servidor TOMCAT mediante su herramienta manager, también junto con este fichero .war , añadimos 2 scripts de bases de datos MYSQL para la creación de la base de datos y de sus tablas y de la inserción de unos registros de prueba.

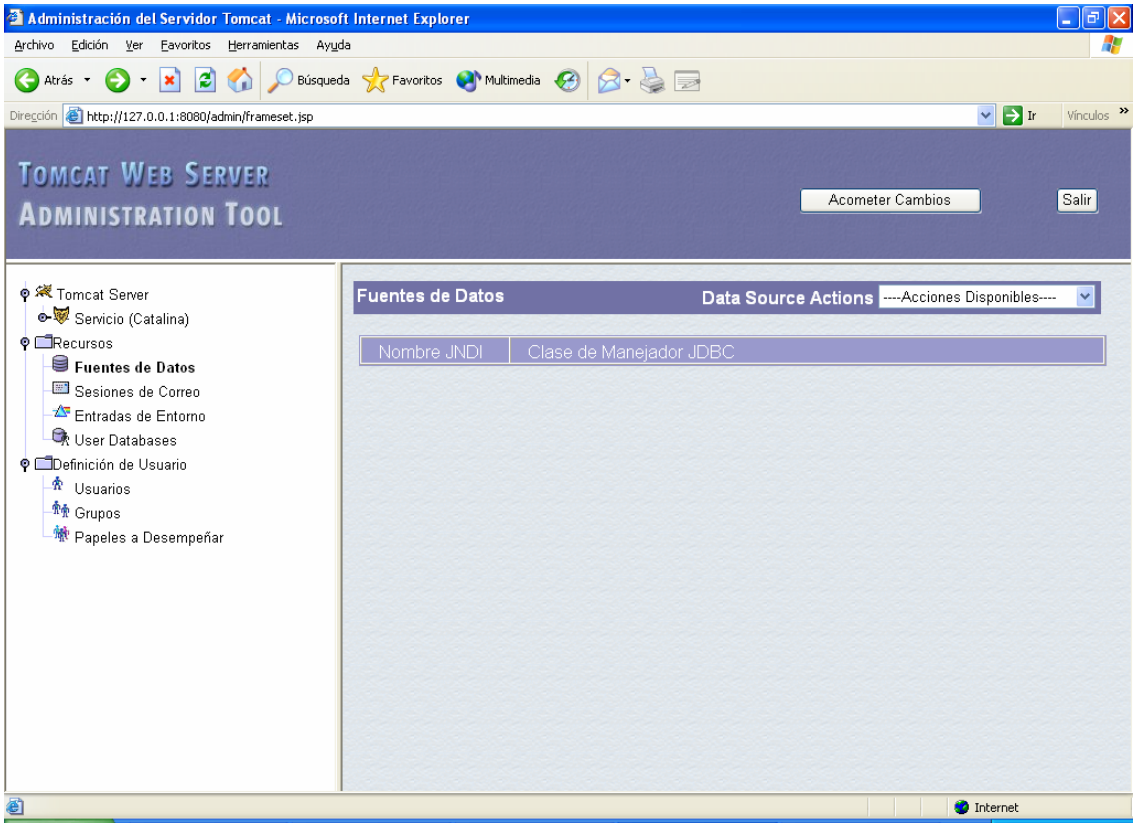
En el propio fichero *.war* **,**van incluido todos los ficheros fuentes *.java* de la aplicación en mismo directorio donde se encuentran los ficheros compilados .class, también en este fichero se incluyen algunas portadas de libros de los registros de prueba.

Software necesario para la instalación del producto y configuración de los mismos, todo para WIndows XP.

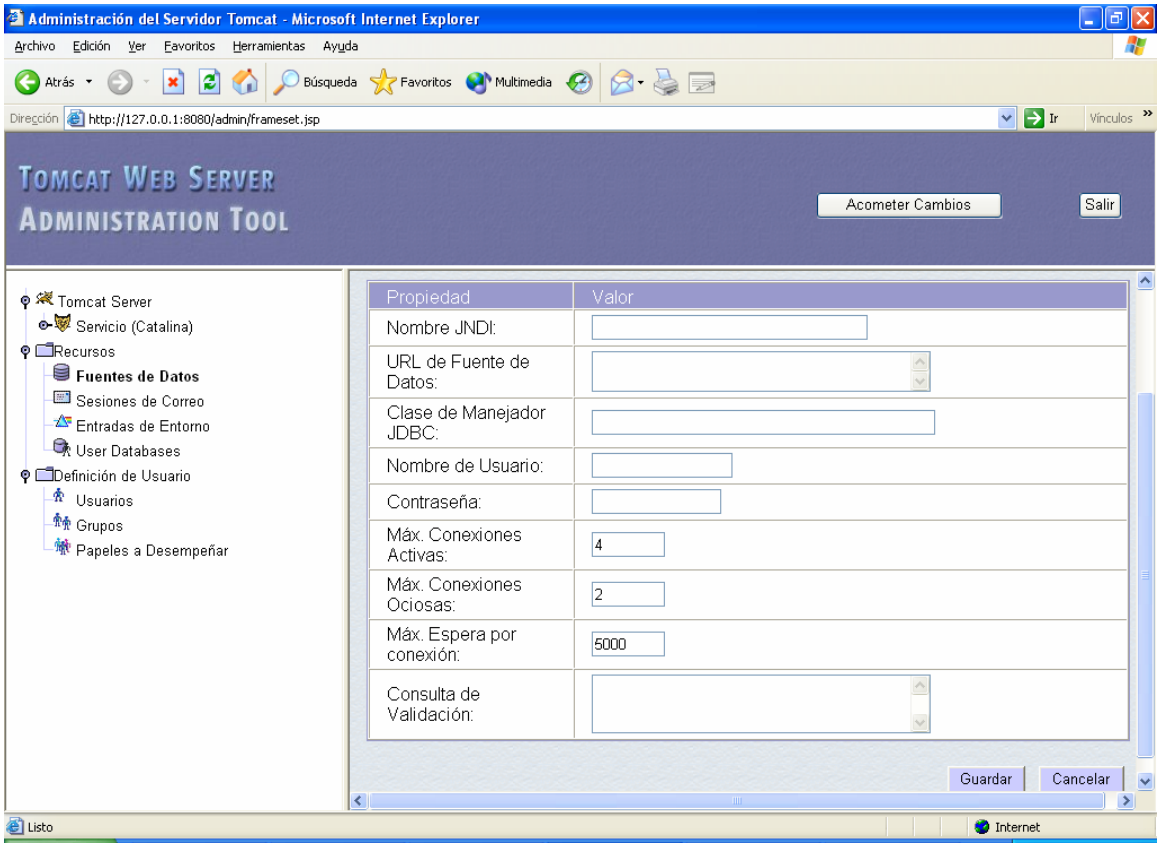
- J2EE 1.4 SDK and Sun Java System Application Server Platform Edition 8.1 2005Q2 UR2
  j2eesdk-1_4_02_2005Q2-windows-ml.exe
  descargable de: http://java.sun.com/j2ee/1.4/download.html#sdk
- Apache Jakarta Tomcat 5.5.7
  jakarta-tomcat-5.5.7.exe
  descargable de : http://archive.apache.org/dist/tomcat/tomcat-5/archive/v5.5.7/bin/
- Herramienta de configuración Admin para Tomcat 5.5.7
  jakarta-tomcat-5.5.7-admin.zip
  descargable de : http://archive.apache.org/dist/tomcat/tomcat-5/archive/v5.5.7/bin/
- Servidor de base de datos MYSQL 4.1
  mysql-4.1.15-win32.zip
  descargable de : http://dev.mysql.com/downloads/mysql/4.1.html
- Conector JAVA-JDBC con MYSQL 4.1 . Versión 3.1
  mysql-connector-java-3.1.11.zip
  descargable de: http://dev.mysql.com/downloads/connector/j/3.1.html/

Pasos de instalación:

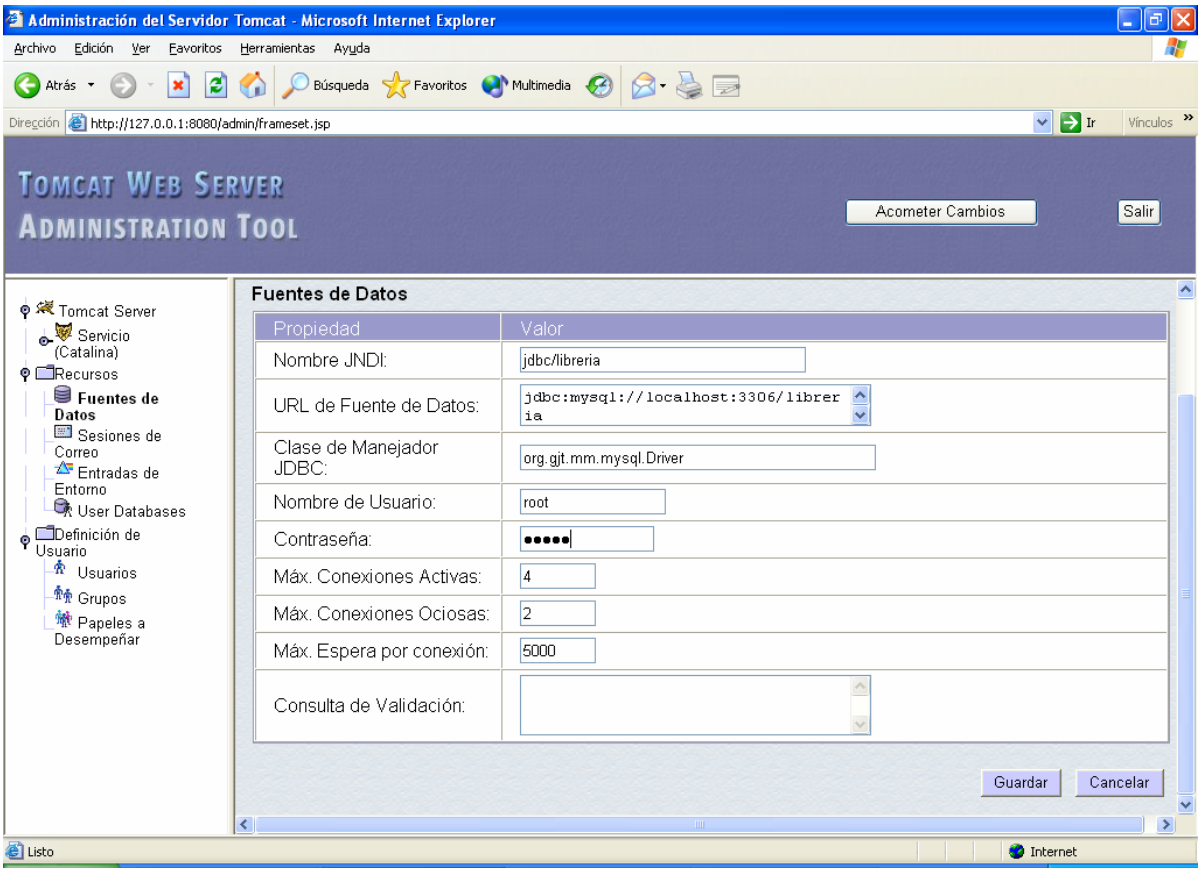
1. Instalamos J2EE 1.4 con las opciones por defecto y el directorio sugerido en la instalación
2. Instalamos Tomcat 5.5.7 , con todas las opciones (instalación completa) , crear el usuario administrador de Tomcat con su clave correspondiente.
3. Instalamos la herramienta Admin para Tomcat , descomprimiendo esta en la carpeta de instalación de Tomcat.
4. Instalamos MYSQL Server son las opciones por defecto , con el usuario *root* con clave *admin*, para no tener problemas despues con las conexiones.
5. Descomprimimos el conector JDBC y copiamos el fichero:
   mysql-connector-java-3.1.11-bin.jar en el directorio *common/lib* de la carpeta de instalación de Tomcat
6. Mediante la consola de MYSQL Server y mediante los scripts SQL suministrados creamos la base de datos,las tablas y los registros de prueba . (la base de datos se llamara *librería*)
7. Creamos el pool de conexiones para Tomcat con la herramienta Admin del propio Tomcat



Creamos una nueva fuente de datos:



los datos son:



como nombre JNDI: jdbc/librería
URL de Fuente de Datos:mysql://localhost:3306/librería
Clase de Manejador JDBC: org.gjt.mm.mysql.Driver
Nombre de Usuario:root (es la de MYSQL)
Contraseña: admin (es la que se crea en la instalación de MYSQL)

y se graba

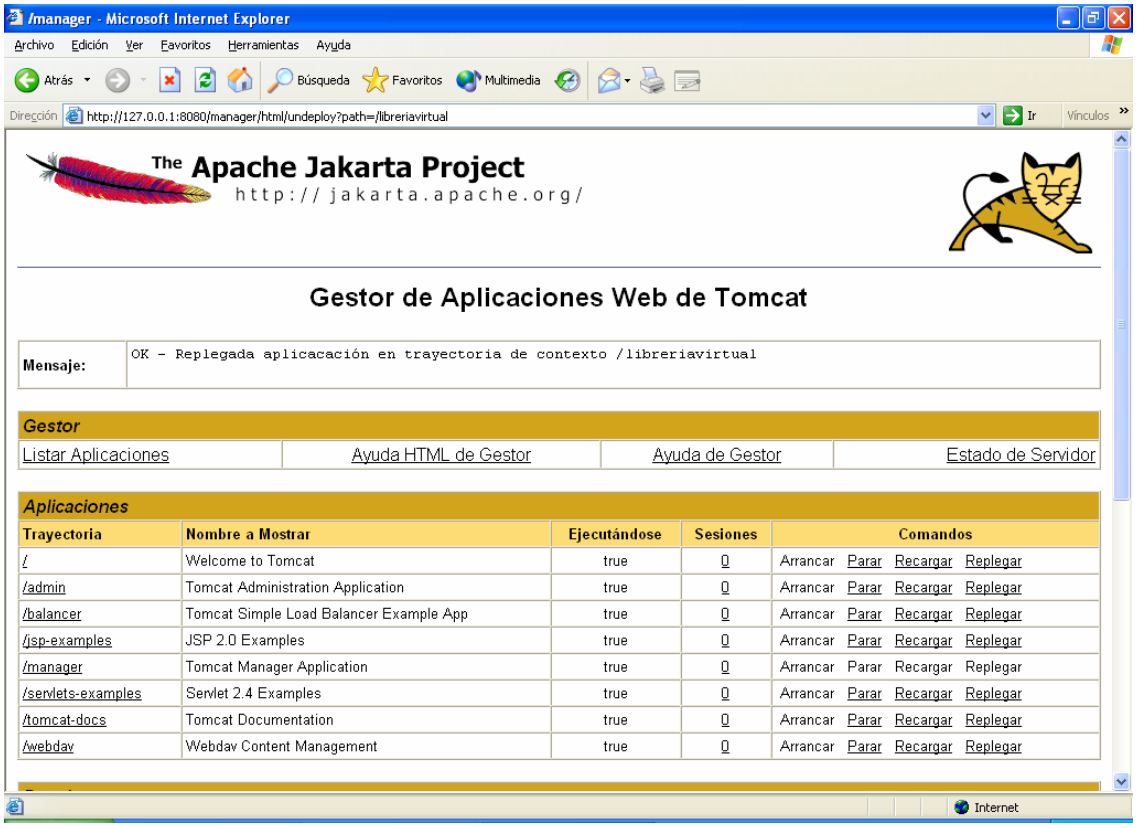
y se reinicia a traves del monitor de Tomcat el propio Tomcat

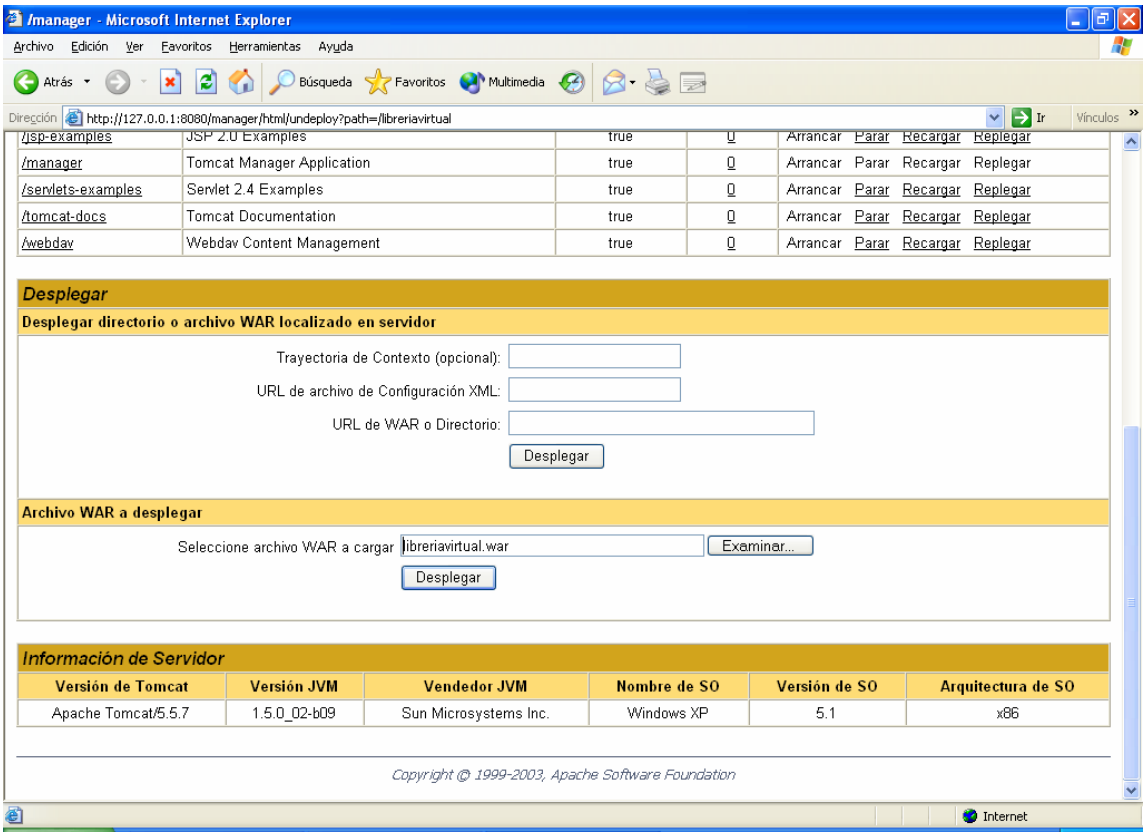
asegurarnos que el fichero *server.xml* de la carpeta *conf* de la instalación de Tomcat:

```
<Resource
    name="jdbc/libreria"
    type="javax.sql.DataSource"
    password="admin"
    driverClassName="org.gjt.mm.mysql.Driver "
    maxIdle="2"
    maxWait="5000"
    username="root"
    url="jdbc:mysql://localhost:3306/libreria"
    maxActive="4"/>
```

para que el pool de conexiones funcione adecuadamente.

8. Desplegamos la aplicación del fichero .war a traves del manager de Tomcat





9. Probamos la aplicación a traves del navegador web (suponiendo que en la instalación de Tomcat no hayamos modificado el puerto de conexión por defecto) con esta URL:

   http://localhost:8080/libreriavirtual

10. Si se desea dar de alta algun *libro* nuevo con una foto de la portada, mandar este fichero de la portada al directorio libros de donde se ha instalado la aplicación (sería idea hacer este directorio publico por FTP para enviar ahí directamente las fotos de manera remota).
11. El Login y el Password para administración es: *admin* y *clave* respectivamente
12. Las sesiones duran 15 minutos, configurables en el fichero *web.xml.*
13. La aplicación se ha instalado (con el Manager) bajo la carpeta *webapps/libreriavirtual* del directorio de instalación de Tomcat