1. Introducción

En la actualidad, es impensable una sociedad sin Internet, las posibilidades de esta herramienta a la hora de abrirse camino en los negocios son infinitas ya que se cuentan por docenas las ventajas. Internet aumenta el marco de actuación de las empresas que se dedican al comercio electrónico de forma exponencial, la única condición es tener los conocimientos tecnológicos necesarios como para poder aplicar las nuevas tecnologías a la misión de la organización.

En mi caso, creo que durante la carrera se han dedicado pocas horas a realizar aplicaciones relacionadas con la red de redes. Por esta razón me incliné por llevar a cabo un proyecto relacionado con Internet. Hay que tener en cuenta que actualmente, dominar las tecnologías relacionadas con este mundo es un valor añadido de cara al la inminente entrada en el mundo laboral.

Mi idea era llevar a cabo un proyecto, con un poco de esfuerzo añadido por mi parte, que se centrara en la implementación de una tecnología relativamente joven de la que no se hubieran realizado aportaciones durante la carrera. Diremos pues que como principal objetivo, estaría el de aplicar una nueva tecnología a un caso práctico.

El caso práctico al que se hace referencia es el de una tienda virtual, típico ejemplo y más que estudiado actualmente, pero el hecho es que este proyecto lo que busca no es llevar a cabo una aplicación pionera, sino la posibilidad de empaparse de conocimientos extras los cuales espero que me sirvan en un futuro no muy lejano.

JavaServer Faces es la tecnología a la que unas líneas más arriba hacía referencia. En la única asignatura relacionada con aplicaciones Web que hemos llevado a cabo en la carrera, se hacia referencia a Struts, tecnología que lleva ya bastante tiempo en marcha.

Lo verdaderamente interesante y atractivo de este proyecto es que las tecnologías Web están a la orden del día y son exigidas en la mayoría de empresas, si a esto le sumamos un

especial interés por mi parte para llegar a aprender el funcionamiento de alguna, llegamos a la conclusión de que mi elección ha sido la acertada.

La motivación personal, las ganas de aprender así como de ver hasta donde soy capaz de llegar son comparables a las ganas que tengo de ponerme a desarrollar el proyecto final de carrera.

2. Objetivos

El principal objetivo es aplicar la tecnología JavaServer Faces (JSF) para el desarrollo de las principales y más básicas funcionalidades de una tienda virtual. Teniendo en cuenta que se trata de un caso práctico más que estudiado y puesto en marcha, es condición indispensable para el desarrollo de la aplicación dicha tecnología. Esta condición obliga a aprender desde cero, la metodología de JavaServer Faces además de todos los aspectos tecnológicos que la rodean, ya que durante la carrera no se ha realizado ningún aprendizaje de la misma. La utilización de JavaServer Faces, obliga a tener unos conocimientos previos sobre el desarrollo de aplicaciones en entornos Web, así como de otros lenguajes de programación html, jsp o java, sin olvidar los conocimientos para la gestión de bases de datos relacionales.

El segundo objetivo está centrado en la aplicación a desarrollar. Una tienda virtual no lo seria si no se cumpliese su objetivo final, realizar ventas a sus clientes. Por esta razón, el proceso de compra y todo lo que éste conlleva es sin duda el otro gran propósito del proyecto.

Por otro lado, y teniendo en cuenta mi poca experiencia en aplicaciones Web, me he planteado como objetivo personal este proyecto y así aumentar mis conocimientos.

2.1 Objetivos generales

Tras la aceptación del proyecto y una primera reunión con el ponente, se fijaron las funcionalidades a desarrollar.

1. Implementación de las funcionales necesarias par poder llevar a cabo una compra a través de Internet. A continuación los más significativos.
   a. Administración del catálogo de productos.
   b. Gestión de clientes.
   c. Utilización de una cesta de la compra.

2. Siempre que sea posible, la utilización de software libre.
    a. Un entorno de desarrollo integrado (IDE) gratuito como Netbeans.
    b. Sistema gestor de bases de datos MySql.
    c. Tecnología a utilizar Java.
    d. Marco de trabajo JavaServer Faces.
    e. Visualización y puesta en funcionamiento de la Web a través de Internet Explorer.

2.2 Objetivos funcionales

Es de suma importancia llevar a cabo el diseño de la aplicación teniendo en cuenta aspectos tan significativos en el desarrollo software como el posterior mantenimiento o la adaptación a nuevas funcionalidades.

Toda la información referente a los clientes, las ventas y los productos ha de quedar almacenadas en la base de datos. Los clientes podrán en cualquier momento ejercer su derecho a darse de baja de la tienda.

Teniendo en cuenta los tres posibles roles de usuario que pueden existir invitado en una tienda virtual invitado, cliente y administrador, clasificaremos las diferentes funcionalidades:

1- Funcionalidades para invitado:
    a. Usar catalogo.
    b. Usar cesta de la compra.
    c. Registrarse, dándose de alta.
2- Funcionalidades para cliente:
    a. Entrar al sistema
    b. Cerrar sesión saliendo del sistema.
    c. Baja y modificación de cliente.
    d. Recordar contraseña.
    e. Usar cesta de la compra.
    f. Usar catalogo.
    g. Hacer pedido.

3- Funcionalidades para administrador:
    a. Gestión del catalogo realizando altas, bajas, modificaciones de categorías.
    b. Gestión de catalogo realizando altas, bajas, modificaciones de productos.
    c. Gestión de catalogo introduciendo o sacando productos del catalogo.
    d. Gestionar las ventas.
    e. Gestionar sus clientes dándolos de baja.

2.3 Objetivos específicos

En este apartado se comentarán los objetivos relacionados con los conocimientos técnicos ha adquirir así como de los diferentes marcos de trabajo a utilizar durante el transcurso del proyecto.

- Implementación de la aplicación a través del framework JSF, basado en la tecnología Java y del módulo proporcionado por Netbeans Visual Web JSF.

- Implementación del patrón Modelo Vista Controlador (MVC) aplicado a JSF y de otros patrones arquitectónicos entre los que destacan el patrón DAO (Objeto de acceso a datos) para llevar a cabo las diferentes funcionalidades.

- Utilización del framework iBatis, de Apache Software Fundation para llevar a cabo la implementación de la persistencia.

- Puesta en marcha de la Base de datos a través de herramientas como MySQL WorkBench 5.0 OSS para el diseño y MySQL Administrador 1.2.12 para la administración y gestión de la base de datos. Ambas herramientas son software libre proporcionado por la plataforma MySQL.

3. Estudio previo

A continuación se llevará a cabo un estudio de los diferentes sitios Web dedicados al comercio por Internet. El objetivo de esta sección es realizar una comparativa entre tres claros ejemplos de tiendas virtuales dedicadas desde hace tiempo a esta actividad, para poder extraer todos los aspectos que tienen en común, así como aquellas funcionalidades ofrecidas a sus clientes. Interesan cosas como información pedida a los clientes, pasos a realizar hasta llegar a efectuar una compra, diseño del web, usabilidad o accesibilidad.

Las tres tiendas llevadas a estudio han sido Figura 1.www.fnac.es , Figura 2. www.elcorteingles.es y Figura 3. www.vinosonline.es. Todas ellas tienen aspectos en común, de los que destacaríamos el objetivo principal o la variedad de productos ofertados. Hay que tener en cuenta que las dos primeras son dos de las tiendas virtuales más conocidas por los compradores online. Destacaremos funciones comunes y básicas para poder llevar a cabo ventas, como registro de clientes, carrito de la compra o buscador.

3.1 Casos a estudiar

Figura 1. www.fnac.es

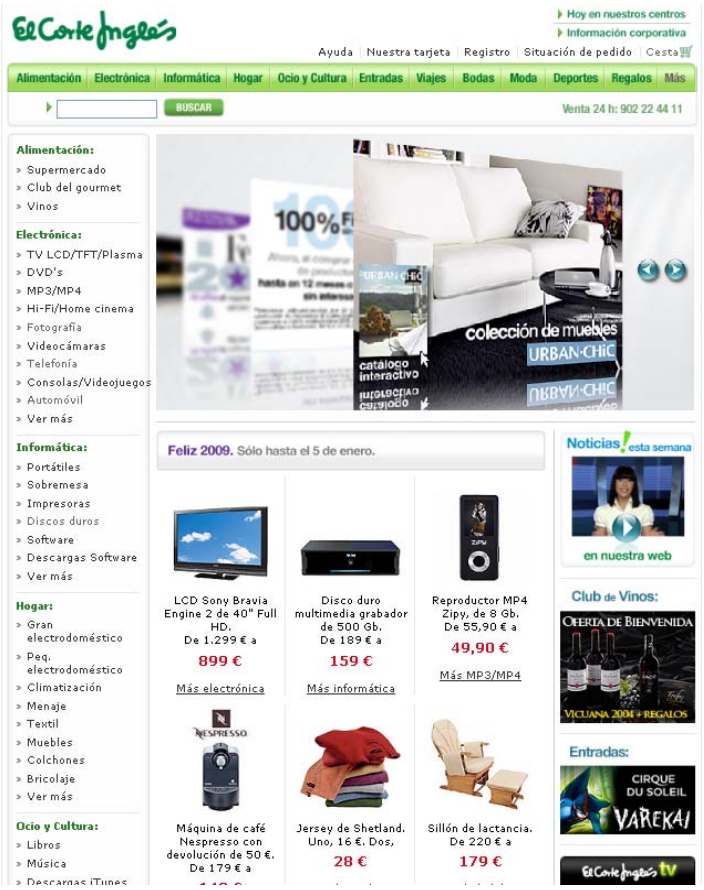

Figura 2. www.elcorteingles.es

Figura 3. www.vinosonline.es

3.2 Resultados del estudio

Tras estudiar a fondo cada uno de los ejemplos seleccionados, hemos obtenido los siguientes resultados expuestos en una tabla comparativa de las principales características y funcionalidades. Resaltando las que en un futuro pueden llegar a implantarse en nuestra tienda virtual. También se observan cuales son las tendencias de las tiendas actualmente.

	www.fnac.es	www.elcorteingles.es	www.vinosonline.es
Administración Web	SI	SI	SI
Buscador	SI	SI	SI
Gestión completa de gastos de transporte.	SI	SI	NO
Secciones de novedades	SI	SI	NO
Funcionalidad de recomendaciones	SI	SI	NO
Gestión de marcas fabricantes	SI	SI	SI
Ficha de producto	SI	SI	SI
Soporte gratuito	SI	SI	SI
Formas de pago diferentes	SI	SI	SI
Pagos PayPal	NO	NO	SI
Subscripciones	SI	SI	NO
Gestión de descuentos en pedidos	SI	SI	NO
Cesta	SI	SI	SI
Histórico de cesta	NO	SI	NO
Proceso compra	4 tras registro	3 tras registro	5 tras registro(PayPal)
Menú horizontal /vertical	SI/SI	SI/SI	SI/NO
Registro de clientes	SI	SI	SI

Tabla 1. Resultados del estudio

4. Tecnología y metodología

4.1. Estudio de tecnologías

Aunque la elección de la tecnología a utilizar en el proyecto ya está decidida, JavaServer Faces, es de suma importancia echar un vistazo a los lenguajes de programación más populares que existen actualmente. Diferentes lenguajes de programación capaces de llevar a cabo la consecución de los objetivos propuestos para la elaboración de este proyecto. Cual va a ser el framework que se utilizará es otra importante decisión a tomar tras la elección de la tecnología. Se llevará a cabo una comparativa

A continuación se realizará un análisis comparativo con el fin de asimilar las principales diferencias entre las tecnologías expuestas a la comparativa, éstas serán Java y .net.

4.1.1. Java vs. .Net

Ambas son actualmente las dos tecnologías más populares en el mundo del desarrollo de aplicaciones.

Windows .NET Framework [1] es el componente de Microsoft para crear y ejecutar de aplicaciones de software y servicios Web XML. Es compatible con más de 20 lenguajes de programación diferentes. Se encarga de la mayor parte de la estructura necesaria para generar software, lo que permite a los programadores centrarse en el código lógico esencial para el negocio. Facilita más que nunca la creación, implementación y administración de aplicaciones seguras, sólidas y de gran rendimiento. Windows .NET Framework se compone de Common Language Runtime (Lenguaje común en tiempo de ejecución) máquina virtual de la plataforma .Net de Microsoft, un conjunto unificado de bibliotecas de clases tales como ASP.NET para aplicaciones Web y servicios Web XML, Windows Forms para aplicaciones cliente inteligentes y ADO.NET para el acceso a datos sin rigidez.

Figura 4. Microsoft .net

Windows .NET Framework ofrece entre otras cosas un entorno enormemente productivo, basado en estándares y la flexibilidad para afrontar los retos que suponen la implementación y la operación de aplicaciones en toda la empresa.

Por otro lado, la verdadera competencia de .Net no es JAVA sino J2EE (Java 2 Enterprise Edition), J2EE [2] es un grupo de especificaciones que forman lo que es denominado, Java Application Server, y vendría a ser la plataforma software de .net (Common Language Runtime). J2EE define un estándar para el desarrollo de aplicaciones multi-capa diseñado por Sun Microsystems. J2EE simplifica las aplicaciones basándolas en componentes modulares y estandarizados, proveyendo de un completo conjunto de servicios a estos componentes, y manejando muchas de las funciones de la aplicación de forma automática, sin necesidad de una programación compleja.

Figura 5. Java

A favor de .NET es que ha tenido la oportunidad de aprender de los errores de J2EE, debido a que es una tecnología más actual en el tiempo, su diseño esta bien definido y estructurado, cumpliendo todas las exigencias de una buena ingeniera de software.

Para acabar, [3] .net encuentra en Visual Studio una gran ventaja respecto a J2EE ya que se trata de un potente entorno de desarrollo. En cambio J2EE cuenta con un mayor número de entornos pero por el momento ninguno se encuentra a las alturas de Visual Studio.

4.1.2. Elección de la tecnología

Craig R. McClanahan creador de la tecnología Struts también está presente en la especificación de JavaServer Faces.

La tecnología JavaServer Faces (JSF) [4] tiene como principal objetivo la elaboración de aplicaciones Web emulando la forma de trabajar en aplicaciones locales java Swing, estilo AWT o cualquier otra API similar.

Esta tecnología pretende modificar la estructura de peticiones tradicional, petición a través de formulario-respuesta construida en HTML a través de páginas JSP, Java Server Faces pretende gestionar las acciones producidas por el usuario en su página HTML, traducirlas a eventos y enviarlos al servidor con el objetivo de actualizar la página HTML de origen reflejando los cambios provocados por dichas las acciones. De forma que en lugar de realizar aplicaciones que utilicen ventanas basadas en la clase JFrame, lo hagan en una página HTML.

JavaServer Faces está basada en la tecnología Java y en el patrón Modelo Vista Controlador (MVC). Son como cualquier otra aplicación Web Java, se ejecutan en un contenedor de servlets y típicamente contienen:

- Objetos de modelo (JavaBeans) que contienen datos y funcionalidades típicas de la aplicación.
- Páginas JSP.
- Librería para representar y escuchar eventos, validaciones, navegación de páginas y otras acciones en el lado del servidor.
- Clases en el lado del servidor capaces de acceder a la base de datos.
- Librerías de etiquetas personalizadas para JavaServer Pages que permiten expresar una interfaz JavaServer Faces dentro de una página JSP a través de componentes UI.

La interfaz de usuario que se crea con la tecnología JavaServer Faces se muestra en la figura 6 como miUI en el lado del servidor.

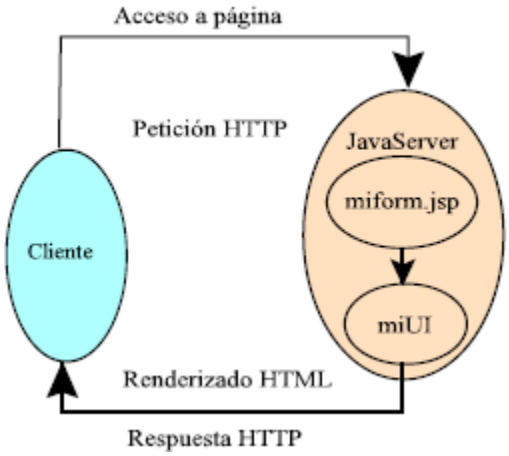

Figura 6. JavaServer Faces

4.1.3. JavaServer Faces vs. Struts

Ambos pretenden normalizar y estandarizar el desarrollo de aplicaciones Web. En la línea temporal, Struts es una tecnología más antigua y por lo tanto con una base de experiencia importante, por lo tanto JavaServer Faces se ha aprovechado en gran medida de esta característica. La similitud entre JSF y Struts es significativa en cuanto a representación de los datos al usuario ya que en ambas la interfaz de usuario se implementa a través de páginas JSP. Ambas están basadas en Java y el patrón Modelo Vista Controlar (MVC).

JSF se presenta como la gran opción a Struts para desarrolladores que no tengan conocimientos de Struts. Ha tener en cuenta que Struts es claro vencedor en el aspecto de madurez debido a que JSF es una tecnología relativamente joven.

Ventajas JavaServer Faces

- JSF trata la vista de usuario a través de componentes y basada en eventos como por ejemplo pulsar un botón. Más próximo a una interfaz basada en Java Swing que en la típica aplicación Web.
- JSF es muy flexible.
- La total separación entre comportamiento y presentación siempre buscada por otras tecnologías, es en JSF cuando toma su mayor protagonismo. El

manejo de eventos específicos de los componentes o el manejo de elementos UI como objetos con estado en el servidor es otra gran ventaja.

- La separación entre presentación y la lógica del negocio permite dividir la aplicación para que en un mismo grupo de desarrollo cada componente se centre en una parte.
- JSF es una especificación estándar la cual no te obliga a atarte a un proveedor de componentes.
- Permite la utilización de varias tecnologías de presentación junto a JSP.

Conclusión

La tecnología JavaServer Faces contiene fundamentos de peso así como una rica arquitectura para manejar el estado de los componentes, procesar los datos, validar la entrada del usuario, y manejar eventos.

4.1.4. JavaServer Faces y el Modelo Vista Controlador

En general todos los patrones de diseño permiten la estandarización en el desarrollo software. Particularmente el patrón Modelo Vista Controlador (MVC) [4] [5], permite separar la lógica del negocio, la presentación de la información y la lógica de control consiguiendo de esta manera aplicaciones más mantenibles, con menos acoplamientos, más inteligibles e incluso más adaptables potenciando en gran medida la calidad del software.

Una de las ventajas más destacadas es que te permite desarrollar la aplicación por separado. Esto toma sentido cuando se trata de una aplicación desarrollada por un grupo de trabajo.

El MVC consta de:

- Una o más vistas.
- El modelo.
- Y un controlador.

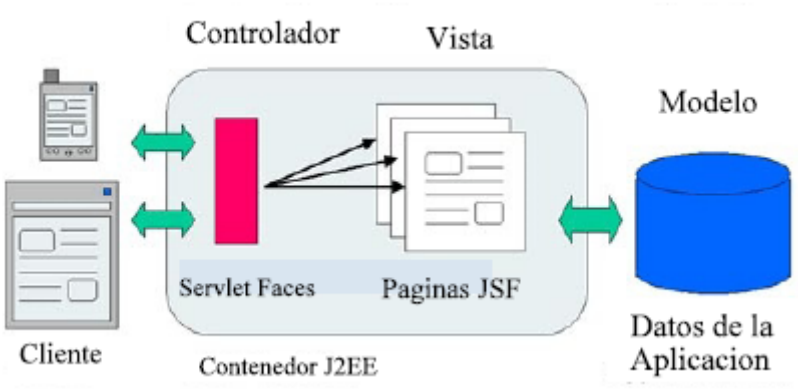

Figura 7. JavaServer Faces-MVC

Modelo

Objeto que representa y trabaja directamente gestionando los datos y controlando todas sus transformaciones. El modelo no tiene ninguna referencia de las vistas. En nuestro proyecto vendrá determinado por la capa Dominio. JSF se comunica con el modelo de la aplicación a través del fichero **faces-config.xml** donde se detalla un objeto perteneciente a la clase del dominio Usuario, la clase a la que hace referencia y el ámbito de sesión.

```
<managed-bean>
<managed-bean-name>usuario</managed-bean-name>
<managed-bean-class>CapaDominio.Usuario</managed-bean-class>
<managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

Figura 8. Faces-config.xml

Vista

Objeto que maneja la presentación visual de los datos gestionados por el Modelo. Genera una representación visual del modelo y muestra los datos al usuario. En nuestro proyecto vendrá determinado por cada una de las páginas JSF. La manera de conectar el modelo con la vista a través de componentes JSF es haciendo referencia a objetos del modelo existentes en el controlador.

```
<h:inputText value="#{usuario.nombre}"/>
```

Figura 9. Componente de la vista

Controlador

Objeto que actúa sobre los datos del modelo y que entra en acción cuando se realiza una petición por parte de la vista o una actualización por parte del modelo hacia la vista. En JSF opera como un gestor que reacciona ante los eventos provocados por el usuario, procesa sus acciones y los valores de estos eventos, ejecutando código para actualizar el modelo o la vista. En nuestro proyecto vendrá determinado por “domotechonline”.

Un ejemplo de petición seria:

```
<h:commandButton value="Aceptar" action="login"/>
```

Figura 10. Ejemplo de acción

Donde la regla de navegación estará definida en el archivo **faces-config.xml** de la siguiente manera:

```
<navigation-rule>
    <from-view-id>/index.jsp</from-view-id>
    <navigation-case>
        <from-outcome>login</from-outcome>
        <to-view-id>/hola.jsp</to-view-id>
    </navigation-case>
</navigation-rule>
```

Figura 11. Regla de navegación

4.1.5. Objetos del modelo (JavaBeans)

Un bean es una clase Java que contiene atributos. Un atributo es un valor identificado por un nombre, pertenece a un tipo determinado y puede ser leído y/o escrito sólo a través de métodos a tal efecto llamados métodos get y set. En nuestro caso son todas aquellas clases que componen la capa Aplicación (domotechonline). En el fichero de configuración faces-config.xml estarán alojadas las definiciones de los beans. De esta manera podemos hacer referencia mediante los métodos get y set a los atributos del bean desde los diferentes componentes de las páginas jsp. La declaración de los objetos

del modelo compartirá espacio en el fichero de configuración con las reglas de validación.

```
<managed-bean>
<managed-bean-name>usuario</managed-bean-name>
<managed-bean-class>CapaDominio.Usuario</managed-bean-class>
<managed-bean-scope>session</managed-bean-scope>
</managed-bean>
</faces-config>
```

Figura 12. Declaración de un bean

Para comodidad del programador aplicaciones Web, un contenedor de servlets suministra diferentes ámbitos, de petición, de sesión y de aplicación. Estos ámbitos normalmente mantienen beans y otros objetos que necesitan estar disponibles en diferentes componentes de una aplicación Web.

4.1.6. Páginas JSF

Se necesita una página JSF por cada pantalla de presentación. Dependiendo de lo que se quiera hacer, las páginas típicas son .jsp o .jsf.

Todo componente que pueda generar eventos tienen un atributo denominado “action” cuyo valor es un String y que es utilizado en gran medida para activar las reglas de navegación que contiene el fichero faces-config.xml, de manera que al producirse una determinada acción, la regla de navegación dirigirá la página hasta la nueva página solicitada.

Un ejemplo de petición de atributo de un objeto del modelo a través de una página jsp sería:

```
<h:inputText value="#{usuario.nombre}"/>
```

Figura 13. Petición de atributo de objeto del modelo

El método getNombre() existente en la clase Usuario de la capa Dominio, permite el acceso al atributo nombre del objeto usuario, existirá una referencia del objeto del dominio en el bean o controlador de la capa Aplicación, siendo éste el encargado de facilitar a la página el nombre del objeto usuario.

4.1.7. Validación y conversión de formularios

En el momento en que el usuario realiza una petición tras haber introducido datos erróneos o inesperados en un formulario, JSF primero convierte y valida [4] los datos de entrada del usuario, si se encuentran errores la página se recarga mostrando el error al usuario para que vuelva a intentarlo, la actualización de los objetos del Modelo (Dominio) empieza cuando se superaron cada una de las validaciones. Ésta es la forma que tiene JSF de proteger la integridad del dominio, es decir da preferencia a la validación y conversión antes que a la petición por parte del usuario.

A continuación a modo de ejemplo la validación y conversión de un tipo de dato fecha.

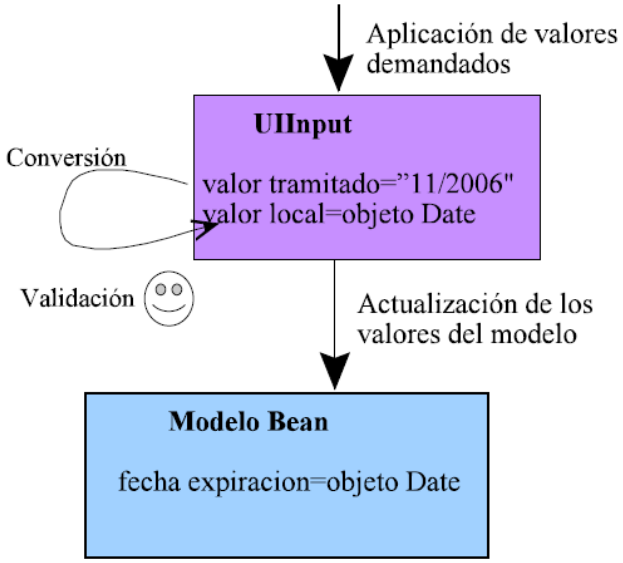

Figura 14. Validación y conversión

4.1.8. Netbeans, IDE (Entorno de desarrollo integrado) escogido

Se ha optado por la utilización de Netbeans 6.1 [6] para la puesta en marcha del proyecto, en primer lugar porque se trata de una herramienta de software libre y en segundo lugar porque es la que utilizamos en la escuela. Netbeans cuenta con una gran base de usuarios, una comunidad en constante crecimiento, y con cerca de 100 socios en todo el mundo.
Sun Microsystems fundó el proyecto de código abierto NetBeans en junio 2000 y continúa siendo el patrocinador principal de los proyectos.

Existe abundante documentación en forma de tutoriales sobre JSF en www.netbeans.org ya que entre otros, Netbeans cuenta con un plugin llamado Visual Web Java Server Faces, basado en la tecnología JSF. Para el desarrollo del proyecto se ha optado por utilizar este framework ya que se adapta perfectamente a sus características.

4.1.9. Visual Web JavaServer Faces, el marco de trabajo

La peculiaridad de este framework perteneciente a Netbeans.org es que te facilita en gran media tu adaptación a JavaServer Faces y a los nuevos componentes, de los que es necesario saber cuales son sus características y comportamientos.

A continuación, mostraremos cuales son las características más destacables del framework Visual Web JavaServer Faces [6]:

- Se trata de una herramienta capaz de aplicar el MVC de JavaServer Faces de una manera visual, sin apenas programar ni el archivo de configuración **faces-config.xml**, ni el comportamiento de los componentes que te facilita en su paleta. Visual Web JSF lo hace por ti.
- Las reglas de navegación se gestionan visualmente, tampoco tienes que programarlas. Lo único que has de hacer es relacionar mediante un simple clic cual es el origen y el destino de las diferentes peticiones del usuario.

- En el momento en que añades una página jsf, Visual Web JSF añade la clase asociada .java de la vista a lo que se llamará "domotechonline" la cual ejercerá de capa Aplicación, es decir de controlador.
- Visual Web JSF, contiene tres pestañas en las que te permite interactuar entre vista (.jsp), controlador(.java) y una tercera pestaña dedicada al diseño de la jsp.
- En toda clase .java generada existen tres atributos dedicados a cada uno de los tres posibles ámbitos Sesión, Aplicación o Petición, de manera que siempre puedes acceder a los atributos de la Sesión a través de sus métodos get y set.
- Existe la posibilidad de añadir cualquier componente de la paleta a tu página jsf únicamente arrastrándolo de la paleta, una vez allí puedes modificar sus características en la ventana de propiedades y lo que es de mayor importancia puedes añadir el componente como atributo de la clase .java asociada a tu jsp a través de la función "Add Binding Attribute". Esto te permitirá cambiar los comportamientos, estados o las características de los componentes a la hora de mostrar la información o de dar respuestas al usuario.
- Otra de las características de Visual Web JSF es la existencia de una opción llamada "Property Bindings", la cual te permite de manera visual editar las propiedades de los componentes relacionándolos con los atributos de los beans de la capa Aplicación.
- Las reglas de validación así como las reglas de conversión de igual forma se gestionan visualmente.
- Es interesante ya que te permite crear tus propias colecciones en los beans e introducir sus contenidos de una forma muy sencilla, simplemente hemos de tener los get y set de cada colección como condición indispensable.

Visual Web JSF también implementa a su manera el patrón DAO para su acceso a los datos de persistencia. En nuestro caso esta funcionalidad del módulo de Netbeans la hemos dejado de lado ya que para darle un punto más atractivo al proyecto hemos preferido utilizar iBATIS para el mapeo de objetos persistentes. Me consta que Visual Web JSF te permite gestionar las transacciones de forma visual, fácil y rápida. En mi

opinión, de una forma poco mantenible y con grandes acoplamientos. Sería interesante dicha utilización para proyectos simples y de nulas ampliaciones siempre que se empiecen desde cero, nunca para proyectos ya empezados.

Para finalizar, comentar que para una primera toma de contacto con la tecnología este marco de trabajo puede llegar a ser interesante puesto que te permite llevar a cabo aplicaciones Web basadas en JavaServer Faces sin tener muchos conocimientos de la tecnología en un tiempo razonable.

4.1.10. iBATIS, el marco de trabajo en persistencia

iBATIS es un marco de trabajo de código abierto basado en capas desarrollado por Apache Software Fundation, que se ocupa de la capa de Persistencia (se sitúa entre la lógica de Negocio y la base de datos). Puede ser implementado en Java y .NET.

Figura 15. Logotipo iBATIS

iBATIS asocia objetos del Dominio con sentencias SQL o procedimientos almacenados mediante ficheros descriptores XML, simplificando la utilización de bases de datos.

Toda implementación de iBATIS incluye los siguientes componentes:

- **Data Mapper:** proporciona una forma sencilla de interacción de datos entre los objetos Java y .NET y bases de datos relacionales.
- **Data Access Object:** abstracción que oculta la persistencia de objetos en la aplicación y proporciona un API de acceso a datos al resto de la aplicación

El acceso a los métodos proporcionados por iBatis se realizará a través de la librería **ibatis-2.3.4.726.jar** [7].

4.2. Metodología

Teniendo en cuenta a los niveles de complejidad que actualmente está llegan el desarrollo del software, la industria se ha visto obligada a realizar modelos en los que se detallan los diferentes procesos software para elaborar productos de calidad.

El proceso software que se utilizará en el desarrollo del proyecto estará basado en la programación clásica o secuencial.

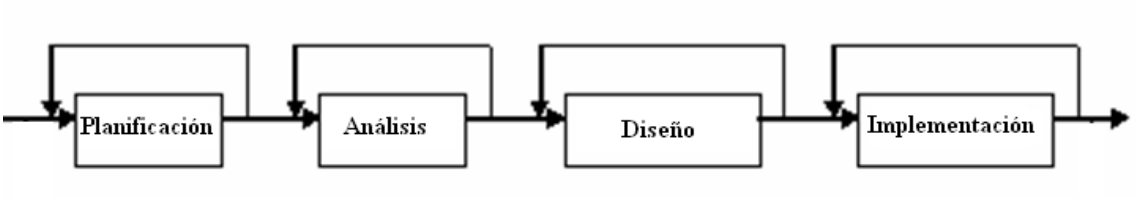

Figura 16. Proceso software

Fase de planificación

Durante esta fase, principalmente se llevará a cabo una planificación temporal detallada del proyecto, dejando claro cada una de las actividades así como de su duración. También se procederá al análisis de recursos tanto tecnológicos, hardware/software, como humanos necesarios para la implementación del proyecto. Finalmente los costes donde se realizará el cálculo económico detallado del proyecto, teniendo en cuenta la planificación temporal y los recursos.

Así pues las actividades son planificación temporal, análisis de recursos y costes.

Fase de análisis

Es la fase de mayor importancia ya que determinará que y quien llevará a cabo las funcionalidades del sistema. Durante esta fase se llevarán a cabo especificaciones detalladas del sistema a través del análisis de requisitos y de unos modelos que sean capaces de soportar las funcionalidades a implementar posteriormente.

Las actividades de análisis del sistema, análisis de requisitos y el análisis de casos de uso serán las principales actividades a llevar a cabo durante esta fase.

Fase de diseño

El diseño es el proceso mediante el cual se llevará a cabo la traducción de las especificaciones en una representación software. Las actividades de esta fase serán el diseño del diagrama de clases y el diseño físico de la base de datos.

En el diseño de diagramas de clases se realizarán todos los diagramas de clases pertenecientes a las diferentes capas a desarrollar: presentación, aplicación, dominio y persistencia. A comentar la importancia de un buen diseño de diagrama de clases especialmente el de la capa dominio, ya que esta capa es la representación de la lógica del negocio. Se tendrán en cuenta decisiones tomadas sobre el entorno tecnológico.

En el diseño físico de la base de datos definiremos la estructura física de los datos que utilizará el sistema a partir del diagrama de clases. El objetivo es llegar a obtener la máxima eficiencia en el tratamiento de los datos teniendo en cuenta aspectos como el sistema gestor de BBDD a utilizar.

Fase de implementación

En la fase de implementación se llevará a cabo la programación de la aplicación teniendo en cuenta la tecnología Java, la tecnología, JavaServer Faces y el framework a utilizar, Visual Web JavaServer Faces. Se tendrán muy presente aspectos como diseño y metodologías de los diferentes patrones de arquitectura software.

5. Planificación

En toda elaboración de un nuevo proyecto, una de las premisas a seguir es dejar claro en un principio, cuales van a ser los pasos que se llevaran a cabo para conseguir los objetivos esperados. Por ello la planificación tendrá en cuenta factores como costes o planificación temporal, donde se detalla entre otras cosas, cuando empieza y cuando acaba el proyecto. En cuanto a los costes se llevará a cabo un estudio detallado de todos los recursos necesarios, tanto humanos como tecnológicos.

Una planificación detallada permite sobrellevar posibles sorpresas que surjan durante la ejecución, evitando que se demore en el tiempo y provocando un aumento en los costes. La etapa de planificación llevará su mayor actividad al inicio del proyecto, ya que es aquí donde se estudian las tareas a llevar a cabo. Para la elaboración de la fase de planificación, existen una serie de herramientas, denominadas “Herramientas Case”, las cuales tienen en cuenta todos y cada uno de los factores que intervendrán en el proyecto, facilitando una estimación temporal y un análisis detallado de cada una de las tareas a llevar a cabo.

5.1. Planificación temporal

Debido a la tecnología escogida y teniendo en cuenta que la experiencia en ella es nula, hay que tener en cuenta la curva de aprendizaje en la planificación del proyecto, esto obliga a destinar parte del tiempo a estudiar y comprender el funcionamiento de dicha tecnología. Tiempo que se ha de imputar al resultado final de los costes. La curva de aprendizaje se experimentará al principio del desarrollo del proyecto obligando a realizar una búsqueda exhaustiva de información, pequeños pruebas y aplicaciones a modo de ejemplo [8][9][10][11][12]. En un principio se comentarán muchos errores que en fases posteriores irán disminuyendo, aplanado de esta manera la curva de aprendizaje. A menudo se volverá a fases anteriores a realizar posibles cambios debido al aprendizaje, sobretodo en las fases iniciales.

Teniendo en cuenta esto y partiendo de la idea que se empeñaran 320 horas para el desarrollo del proyecto, tomaremos la unidad de tiempo en semanas (de lunes a viernes), se destinarán 20 horas a la semana, esto nos da un total de 15 semanas y 4 horas por jornada.

A continuación un análisis detallado de la planificación temporal en función de las tareas a realizar:

Semana 1 – semana 6

Se llevará a cabo instalación del software necesario para las futuras implementaciones.
Empezaremos con la curva de aprendizaje del framework Visual Web Java Server Faces elaborando pequeñas aplicaciones de prueba, familiarizándose con el marco de trabajo escogido así como con IDE (Entorno de desarrollo integrado) Netbeans 6.1. Se llevará a cabo el estudio de la tecnología JavaServer Faces insistiendo en su funcionamiento y características principales [8][9][10][11].
Seguiremos con la curva de aprendizaje de iBATIS, marco de trabajo escogido para el mapeo de objetos de la base de datos. Elaborando pruebas a nivel de usuario y entendiendo su funcionamiento [12].

Semana 7

Durante esta semana realizaremos las tareas de análisis de requisitos, especificación de los casos de uso, diseño del modelo de la base de datos y diseño del diagrama de clases del domino.

Semana 8

Puesta en marcha de la bases de datos aplicando el modelo anteriormente diseñado. Implementación del diagrama de clases del dominio. Y diseño de la interfaz de usuario.

Semana 9 - semana10

Implementación de los primeros casos de uso relativos a la Gestión del catálogo, alta categoría, baja categoría, modificación categoría y alta producto, baja producto, y modificación producto.

Semana 11

Implementación de los casos de uso pertenecientes a Entrar al sistema, Log-In, Log-Out, registrarse y Alta usuario.

Semana 12

Implementación de los casos de uso referentes a la Gestión de usuario, baja usuario y modificación usuario.

Semana 13

Implementación del caso de uso Usar catálogo.

Semana 14

Implementación del casos de uso Usar cesta.

Semana 15

Implementación del caso de uso Hacer pedido.

Para terminar, hacer hincapié en la reserva de una semana más (Semana 16) para la realización de pruebas y diseño gráfico.

5.2. Análisis de recursos

En este apartado se realizará un estudio de los recursos necesarios para el buen desarrollo del proyecto. Distinguiremos entre diferentes tipos de recursos, tecnológicos y humanos.

En cuanto a recursos tecnológicos, hemos tomado la decisión de utilizar el máximo software libre posible para abaratar los costes. Teniendo en cuenta esto utilizaremos el IDE (Entorno de desarrollo integrado) Netbeans 6.1 [6] herramienta de código abierto fácil de utilizar y de grandes prestaciones. También utilizaremos el marco de trabajo

iBATIS [12], de código abierto, facilitado por Apache Software Fundation . Finalmente, MySQL [13] para la base de datos relacional, multi-hilo y multiusuario de Apache Software Fundation.

Los recursos hardware serán, un PC que sea capaz de albergar el software anterior y una conexión a Internet mediante una línea ADSL.

Los recursos humanos constan de un programador o desarrollador software capaz de cumplir con perfiles como analista, diseñador o programador. Es importante que cuente con gran facilidad de aprendizaje. Será el encargado de llevar a cabo cada una de las tareas planificadas en el apartado anterior.

5.3. Costes

A continuación se entrará en detalle de los costes. Los costes en recursos software son 0 €, en cambio los recursos hardware deben tenerse en cuenta.

- Procesador Intel Dual Core E5200 2.5Ghz
- Placa Base Asus P5GC-MX/1333 Socket 775
- Disco Duro 500GB SATA Western Digital/Seagate
- Memoria Kingston 2GB DDR2 800
- Tarjeta gráfica Asus 8400GS 512MB PCI-e
- Monitor LCD 20"

Figura 17. Costes-Hardware

El precio del PC es de 468 € este precio debemos imputarlo al proyecto, si tenemos en cuenta que el precio del PC será pagado en un año, en las 16 semanas que dura el proyecto cuesta 9 € a la semana (468 €/52semanas). La conexión a Internet tiene un coste de 44 € mes un total de 11 € a la semana. Por último la hora del programador corre a 20 €/hora.

	Precio	**Cantidad**	**Subtotal**
Recursos Software	0 €	1 unidad	0 €
Recursos Hardware	9 €	16 semanas	468 €
Programador	20 €/hora	320 horas	6400 €
		Total:	**6868 €**

Tabla 2. Cálculo de costes

Tras el cálculo, el precio total asciende ha 6868 € de los que si se tratara de un proyecto destinado a su venta, se tendrían que descontar todas las horas empleadas en la curva de aprendizaje ya que dichas horas no se le pueden imputar al futuro comprador. Siendo el primer proyecto de este tipo, y restándole las horas de la curva de aprendizaje, se obtendrían perdidas que en un futuro y con la elaboración de un par de proyectos similares serian recuperadas en el conjunto puesto que ya no se le aplicarían curvas de aprendizaje.

6. Análisis

6.1. Análisis de requisitos

La puesta en marcha del proyecto pasa por un previo y exhaustivo estudio de los requisitos necesarios para que el usuario lleve a cabo las funcionalidades descritas en la especificación de casos de uso más adelante. En nuestro caso, al tratarse de una tienda virtual, el objetivo principal del proyecto es que cualquier usuario registrado por el sistema pueda comprar los productos expuestos en la Web a través de Internet. Existen una serie de elementos ha tener en cuenta para que el cliente, que únicamente necesita de una dirección de correo electrónico y una tarjeta de crédito, pueda llevar a cabo la compra final de dichos productos, como por ejemplo la existencia de un catalogo donde se muestren los productos o una cesta en la que el cliente pueda ir depositando los productos destinados a la futura compra. Por otro lado y teniendo en cuenta que el administrador será el encargado de montar el catálogo se le ha de poder facilitar la forma de llevar a cabo dicha funcionalidad, así como llevar un seguimiento las ventas realizadas.

En primer lugar se llevara a cabo el análisis de los posibles usuarios destinados a la utilización de la Web, podemos diferenciar entre tres tipos de roles diferentes, invitado, cliente y administrador.

El invitado es aquel usuario que visita la Web con el fin de recopilar información o simplemente por curiosidad, el cual tiene derecho a la navegación por el catalogo, visitando cada uno de los productos expuestos o realizar búsquedas por nombre de producto y realizar su almacenamiento en la cesta, siendo esta ultima funcionalidad común a los roles de cliente e invitado. En ningún caso éste usuario podrá acceder a realizar un pedido sin antes llevar a cabo el proceso de registro en el que el sistema almacenará su información personal. Mediante una confirmación de usuario y contraseña, el usuario será identificado por el sistema, es decir logado, cambiando así su rol de invitado por el de cliente.

El cliente, que en su momento fue un invitado, llevó a cabo el proceso de registro para llegar a obtener el acceso a la compra final. En dicho proceso, el sistema pedirá datos tales como el nombre, apellidos, dirección, población, provincia, país, código postal, una dirección de correo electrónico, imprescindible y un número de teléfono. Tras el proceso de registro el usuario recibirá en su dirección de correo electrónico una contraseña, previamente adjudicada por el sistema, en la cuenta de correo facilitada. Además de poder realizar el proceso de compra en su totalidad, el cliente podrá realizar gestiones básicas en lo referente a su condición de cliente, como por ejemplo modificar sus datos personales o darse de baja de la tienda en cualquier momento. En ningún caso el cliente tendrá derecho a funcionalidades exclusivas para el administrador como la gestión del catalogo. Al igual que el invitado el cliente podrá utilizar la cesta en cualquier momento, introduciendo productos previamente seleccionados del catalogo.

En cuanto al administrador, comentar que entre otras funcionalidades tiene como principal objetivo llevar a cabo la gestión del catalogo, para ello, ha de poder introducir en el sistema conceptos como categorías, subcategorías o productos. Tendrá total libertad a la hora de sacar o introducir un determinado producto en el catalogo así como de asociarlo a una determinada categoría. Por otro lado podrá llevar a cabo funcionalidades como dar de baja a cualquier usuario o gestionar las ventas tras previa consulta de las mismas.

Los tres tipos de roles diferentes que puede tener un usuario, en ocasiones pueden compartir funcionalidades, es el caso de el proceso para entrar en el sistema, compartida por el cliente y al administrador, la utilización del catalogo, compartida por los tres roles diferentes o la utilización de la cesta funcionalidad compartida por el cliente y el invitado.

Una vez centrados los diferentes tipos de usuarios, procederemos a analizar algunos de los aspectos necesarios que permitirán cumplir con el objetivo planteado anteriormente. Es el caso del catalogo, de la cesta o de la realización de pedido.

El catalogo será la ventana por la cual el usuario visualizará los diferentes productos de la tienda, podemos decir que su comportamiento es de un contenedor dinámico de productos, productos de diferentes tipos, precios, categorías, subcategorías y ofertas.

Una de las principales características además de facilitar la información clave de los productos, nombre, descripción, precio, una imagen representativa y si se trata de una oferta, es su forma de presentar la información. Los productos serán presentados a los usuarios mediante paginaciones, consiguiendo así un orden necesario y una estructuración de la información acorde a los tiempos que corren con el fin de facilitar el uso y el acceso a la información. El usuario podrá ir y venir entre el catalogo y la cesta cuando le convenga. Al tratarse de un catalogo dinámico, éste siempre mostrará las categorías, subcategorías y productos que actualmente existan en la base de datos y que de los cuales existan existencias, a menos que el administrador crea lo contrario. El catalogo contará con una opción de búsqueda la cual facilitará el acceso rápido fácilmente a los productos con el nombre igual al que el usuario introduzca en el campo de búsqueda. Dado que la búsqueda se realiza de productos y los productos se muestran en el catalogo, los resultados de la búsqueda han de presentarse al usuario con el mismo formato en que se le muestra el catalogo este lleva a la utilización de la paginación en los resultados de las búsquedas igual que en el catalogo.

Finalmente, en cuanto a lo que el catalogo se refiere, comentar que toda la información que se muestre de los productos en el catalogo es de suma importancia, pero de todas ellas destacaríamos dos en las que por norma general se presta más atención en la visita a páginas de este estilo, una es la imagen y otra es el precio, de ahí que ambas queden claramente visibles en el catalogo cuando el usuario navegue por él.

La cesta, funcionalidad clave e indispensable en la futura venta, será accesible desde cualquier punto de la aplicación, facilitando así su visita. Una vez que el usuario introduce un determinado producto en la cesta, éste se almacena en la misma y cuando el usuario la visite, ésta mostrará la información referente al futuro pedido en cuanto a productos se refiere, el número de productos introducidos hasta el momento, nombre del producto, cantidad que desea comprar, el precio del artículo, el subtotal de cada artículo, el total de la cesta, opción de borrar un producto, opción de actualizar la cesta, opción de seguir comprando y opción de tramitar el pedido para seguir con el proceso de compra de la cesta actual.

El acceso a la cesta actual de la sesión, estará localizado en la imagen de la “mini cesta” que constantemente se muestra al usuario. En dicha mini cesta además se mostrará información referente al la cesta actual, el numero de artículos y el total de la cesta.

Como contrapunto comentar que la cesta podrá ser utilizada por cualquier usuario que se conecte a la Web, teniendo como opciones, borrar, añadir productos o modificar cantidades, siendo necesaria la identificación mediante el proceso de entrar al sistema para llevar a cabo el proceso de compra hasta el final.

La realización del pedido, se llevará a cavo exclusivamente por clientes logados o por usuarios invitados recién registrados que hayan escogido la opción de tramitar el pedido en su cesta de la compra. Con el cliente logado o registrado quiere decir que ya tenemos todos los datos necesarios de manera que únicamente abra que dirigirlo a la Web de www.PayPal.com a través del botón Check Out para que procese el pago con su tarjeta de crédito a través de dicha página, por último devolver al cliente a nuestra página para que confirme el pago definitivo. En caso de no confirmar, el cliente siempre se podrá echar atrás y cancelar el pedido en última instancia.

A continuación se detallarán las funcionalidades más relevantes, entrando en detalle y comentando cual es su objetivo, quien es el actor principal que llevará a cabo la funcionalidad, precondiciones, poscondiciones, cual es el flujo normal y si lo hubiera el flujo alternativo.

6.2. Especificación de casos de uso

6.2.1. Modelo de casos de uso

Figura 18. Diagrama de casos de uso

Para poder llevar a cabo la creación de una tienda virtual y cumplir con todos y cada uno de los objetivos, en el diagrama de casos de uso se muestran todas funcionalidades a analizar así como los diferentes actores que interactuarán con el sistema. Es el primer paso a realizar en la creación de cualquier aplicación, permitiendo un análisis centrado en las necesidades del usuario. En nuestro caso podemos observar el comportamiento

del sistema cuando éste es expuesto por parte del usuario a una petición en concreto. Echando un vistazo a las funcionalidades a las que se hace mención en el diagrama, podemos destacar los siguientes grandes grupos, **gestión catálogo** (el cual estaría compuesto por alta producto, baja producto, modificación producto, alta categoría, baja categoría y modificación categoría), **gestión cliente** (alta cliente, baja cliente y modificación cliente), **entrar al sistema** (a destacar las cuatro posibles comportamientos Log-in, Log-out, recordar contraseña, registro), **usar cesta**, **usar catálogo**, **hacer pedido**, **gestión de ventas y buscar producto**.

Estas son a grandes rasgos, las funcionalidades mínimas que se han de implementar para poder hablar de tienda virtual y poder realizar con éxito compras vía Web.

6.2.2 Actores principales

Como bien representa el modelo de casos de uso, el sistema interactuara con un actor, éste tendrá tres roles claramente diferenciados administrador, cliente e invitado dependiendo de la funcionalidad a la que se quiera acceder. Los diferentes actores lanzan al sistema diferentes peticiones. Por último recalcar que diferentes actores pueden compartir peticiones al sistema, como claro ejemplo tenemos Entrar al sistema el cual comparten los tres roles.

6.2.3 Casos de uso

Entrar al sistema

- **Objetivos asociados:** Su función principal es la de interpretar el perfil de usuario el cual está accediendo al sistema con el fin de otorgarle permisos, funcionalidades o privilegios.
- **Descripción:** Tanto el administrador como los clientes o invitados entrarán en el sistema para que éste reconozca su perfil. El sistema a de facilitar el acceso.
- **Actor principal:** Invitado, cliente y administrador.
- **Precondiciones:** El usuario podrá entrar en el sistema en cualquier momento durante su visita. A tener en cuenta que el usuario tiene que haber estado dado

de alta en el sistema para poder entrar. Este caso de uso será el punto de partida de casi el resto de casos de usos.

- **Secuencia normal**
  1. Se realiza la conexión por parte del usuario a la Web.
  2. La aplicación mostrará en todo momento la opción para entrar al sistema.
  3. El sistema muestra el formulario de entrada al usuario pidiéndole su correo electrónico y su contraseña.
  4. El usuario introduce su correo electrónico y su contraseña y pulsa enviar.
  5. El sistema identifica al usuario como cliente y le enseña sus opciones de cuenta.
- **Secuencia alternativa**

  4.1. El usuario erró en la contraseña.

  4.1.1. El sistema avisa del error y vuelve al punto 3 dando la opción de pedir una nueva contraseña.

  4.2. El cliente no existe.

  4.2.1. El sistema avisa del error y vuelve al punto 3 dando la opción de darse de alta al usuario.
- **Poscondiciones:** El usuario tiene que haber entrado, siendo identificado por el sistema

Solicitar contraseña

- **Objetivos asociados:** Su función principal es la de facilitar al cliente una nueva contraseña para poder acceder al sistema.
- **Descripción:** Tanto el administrador como los clientes pueden pedir una nueva contraseña.
- **Actor principal:** Cliente y administrador.
- **Precondiciones:** El usuario ha de estar dado de alta en el sistema.
- **Secuencia normal**
  1. Se realiza la conexión por parte del usuario a la Web.
  2. La aplicación mostrará en todo momento la opción para entrar al sistema.
  3. El sistema muestra el formulario de entrada al usuario pidiéndole su correo electrónico.
  4. El usuario introduce su correo electrónico y pulsa enviar.

5. El sistema genera una nueva contraseña y la envía al usuario a su cuenta de correo electrónico.
6. El sistema avisa al usuario de que el proceso se ha realizado con éxito.

- **Secuencia alternativa**

  4.1.El cliente no existe.

  4.1.1. El sistema avisa del error y vuelve al punto 3 dando la opción de darse de alta al usuario.

- **Poscondiciones:** El usuario ha de recibir su nueva contraseña en su cuenta de correo electrónico.

Gestión Clientes

- **Alta cliente**
  - **Objetivos asociados:** Su función principal es la de insertar un nuevo cliente en el sistema.
  - **Descripción:** Únicamente los usuarios con rol de invitados serán los que puedan darse de alta
  - **Actor principal:** Invitado.
  - **Precondiciones:** El sistema ha de poder facilitar la opción de alta usuario para llevar a cabo el registro.
  - **Secuencia normal:**
    1. Se realiza la conexión por parte del usuario a la Web.
    2. La aplicación mostrará en todo momento la opción para darse de alta en el sistema.
    3. El sistema muestra el formulario de alta al usuario pidiéndole nombre, apellidos, dirección, población, provincia, código postal, país, DNI, mail y teléfono.
    4. El usuario introduce todos y pulsa enviar.
    5. El sistema comprueba los datos introducidos.
    6. El sistema genera una nueva contraseña y la envía al usuario a su cuenta de correo electrónico y le realiza el cifrado.
    7. El sistema avisa al usuario de que el proceso se ha realizado con éxito.

- **Secuencia alternativa:**

  5.1. Cualquiera de los datos introducidos son erróneos.

  5.1.1. El sistema avisa del error y vuelve al punto 3.

- **Poscondiciones:**

El usuario ha de recibir su nueva contraseña en su cuenta de correo electrónico.

Quedará almacenado en la base de datos, éste contará con un usuario y una contraseña para entrar en el sistema. Inmediatamente después del proceso de registro, el usuario quedará conectado en la sesión.

- **Baja cliente**
  - **Objetivos asociados:** Su función principal es la de eliminar un cliente de sistema.
  - **Descripción:** Únicamente los usuarios con rol de cliente serán los que puedan darse de baja.
  - **Actor principal:** Cliente y administrador.
  - **Precondiciones:** El sistema ha de poder facilitar la opción de baja cliente para que éste pueda darse de baja. El cliente se ha de haber registrado en el sistema. El cliente podrá cerrar sesión en cualquier momento.
  - **Secuencia normal**
    1. Se realiza la conexión por parte del cliente/administrador a la Web.
    2. El cliente entrará en el sistema.
    3. El sistema muestra la opción, en cualquier momento, de darse de baja al cliente.
    4. El cliente selecciona la opción darse de baja.
    5. El sistema pide al cliente su contraseña.
    6. El cliente introduce su contraseña y pulsa el botón.
    7. El sistema comprueba la contraseña.
    8. El sistema realiza un borrado lógico del cliente y lo saca de la sesión del navegador.
    9. El sistema avisa al cliente de que el proceso se ha realizado con éxito.
  - **Secuencia alternativa**

    2.1. El administrador entrará en el sistema.

2.1.1. El sistema muestra la opción dar de baja a clientes.

2.1.2. El administrador selecciona el cliente y pulsa eliminar.

2.1.3. El sistema realiza el borrado lógico del cliente elegido.

7.1. La contraseña es errónea.

7.1.1. El sistema avisa del error al cliente y vuelve al punto 5.

- **Poscondiciones:** El cliente ha sido borrado del sistema mediante un borrado lógico.

- **Modificar cliente**
  - **Objetivos asociados:** Su función principal es la de modificar cualquier dato del cliente haciéndoselo saber al sistema.
  - **Descripción:** Únicamente los usuarios con rol de cliente serán los que puedan modificar sus datos.
  - **Actor principal:** Cliente.
  - **Precondiciones:** El sistema ha de poder facilitar la opción de modificar cliente para que éste pueda modificarse sus datos. El cliente se ha de haber registrado en el sistema. El cliente podrá cerrar sesión en cualquier momento.
  - **Secuencia normal**
    1. Se realiza la conexión por parte del cliente a la Web.
    2. El cliente entrará en el sistema.
    3. El sistema muestra la opción, en cualquier momento, de darse modificar los datos del cliente.
    4. El cliente selecciona la opción modificar datos.
    5. El sistema muestra al cliente su nombre, apellidos, dirección, población, provincia, país, código postal, correo electrónico y teléfono.
    6. El cliente modifica los datos correspondientes además de la actual y futura contraseña.
    7. El sistema comprueba los datos introducidos.
    8. El sistema actualiza los cambios del cliente cifrando la nueva contraseña en la BBDD.
    9. El sistema avisa al cliente de que el proceso se ha realizado con éxito.

- **Secuencia alternativa**

  7.1. La contraseña es errónea.

  7.1.1. El sistema avisa del error al cliente y vuelve al punto 5.

- **Poscondiciones:** Los nuevos datos introducidos por el cliente han sido actualizados en la base de datos.

Gestión catálogo

- **Alta categoría**
  - **Objetivos asociados:** Su función principal es la de insertar una nueva categoría en el sistema. La categoría ha de ser irrepetible, de manera que no pueden coincidir dos categorías con el mismo nombre el la base de datos.
  - **Descripción:** Únicamente los usuarios con rol de administrador serán los que puedan dar de alta una categoría nuevo en el sistema.
  - **Actor principal:** Administrador.
  - **Precondiciones:** El sistema ha de poder facilitar la opción de alta categoría al administrador. No se aceptarán repeticiones de categorías. El administrador podrá cerrar sesión en cualquier momento.
  - **Secuencia normal**
    1. Se realiza la conexión por parte del Administrador a la Web.
    2. El usuario entra en el sistema como administrador.
    3. El sistema muestra las diferentes funcionalidades al administrador.
    4. El administrador selecciona la opción crear nueva categoría.
    5. El sistema muestra el formulario de alta categoría al Administrador pidiéndole el nombre de la categoría y el formulario de alta subcategoría donde aparecerá una lista de las categorías existentes en la BBDD éste introducirá el nombre, la categoría madre y una imagen.
    6. Si lo que desea es introducir una nueva categoría el administrador deberá introducir el nombre de la categoría, por el contrario si lo

que desea es insertar una nueva subcategoría, seleccionará del listado la categoría a la que pertenecerá.

7. El sistema comprueba los datos introducidos.
8. El sistema guardará la nueva categoría en la BBDD.

- **Secuencia alternativa**

  *se repetirán los pasos del 1 al 8 tantas veces como el administrador lo desee.

  7.1. Cualquiera de los datos introducidos son erróneos.

  7.1.1. El sistema avisa del error y vuelve al punto 5.

- **Poscondiciones:** Las nuevas categorías han de ser almacenadas en la BBDD.

- **Baja categoría**
  - **Objetivos asociados:** El objetivo es llegar a realizar la eliminación de las categorías existentes en la BBDD.
  - **Descripción:** Únicamente los usuarios con rol de administrador serán los que puedan borrar una categoría.
  - **Actor principal:** Administrador.
  - **Precondiciones:** El sistema ha de poder facilitar la opción de eliminar categorías al administrador. Una subcategoría con productos asociados no podrá ser eliminada. El administrador podrá cerrar sesión en cualquier momento.
  - **Secuencia normal**

    1. Se realiza la conexión por parte del Administrador a la Web.
    2. El usuario entra en el sistema como administrador.
    3. El sistema muestra las diferentes funcionalidades al administrador.
    4. El administrador escoge la opción borrar producto.
    5. El sistema muestra el formulario para llevar a cabo la eliminación de los productos al Administrador.
    6. El administrador selecciona la categoría del listado de categorías.
    7. El sistema muestra un listado con las subcategorías en el caso de que las tenga.

8. El administrador selecciona la categoría o subcategoría que desea eliminar y pulsa aceptar.
9. El sistema comprueba, en caso de eliminar una subcategoría si tiene productos asociados.
10. El sistema actualiza los cambios realizados en la BBDD.

- **Secuencia alternativa**

  *se repetirán los pasos del 1 al 10 tantas veces como el administrador lo desee

  9.1. La subcategoría no se puede eliminar, tiene productos asociados.

  9.1.1. El sistema avisa del error y vuelve al punto 5.

- **Poscondiciones:** La categoría seleccionada ha de ser eliminada de la BBDD.

- **Modificar categoría**
  - **Objetivos asociados:** El objetivo es llegar a realizar la modificación de cualquiera de las categorías existentes en la BBDD.
  - **Descripción:** Únicamente los usuarios con rol de administrador serán los que puedan modificar una categoría.
  - **Actor principal:** Administrador.
  - **Precondiciones:** El sistema ha de poder facilitar la opción de modificar categorías al administrador. El administrador podrá cerrar sesión en cualquier momento.
  - **Secuencia normal**
    1. Se realiza la conexión por parte del Administrador a la Web.
    2. El usuario entra en el sistema como administrador.
    3. El sistema muestra las diferentes funcionalidades al administrador.
    4. El administrador selecciona la opción modificar categoría.
    5. El sistema muestra el formulario para llevar a cabo las modificaciones de las categorías al Administrador.
    6. El administrador selecciona la categoría que desea modificar.
    7. El sistema muestra el nombre de la categoría así como un listado de todas sus subcategorías.

8. El administrador modifica el nombre de la categoría o de cualquiera de sus subcategorías si ésta las tuviera y pulsa aceptar.
9. El sistema comprueba los nuevos datos introducidos.
10. El sistema actualiza los cambios realizados en las categorías en la BBDD.

- **Secuencia alternativa**

  *se repetirán los pasos del 1 al 10 tantas veces como el administrador lo desee

  9.1. Cualquiera de los datos introducidos son erróneos.

  9.1.1. El sistema avisa del error y vuelve al punto 5.

- **Poscondiciones:** Las categorías modificadas han de ser actualizadas en la BBDD.

- **Alta producto**
  - **Objetivos asociados:** Su función principal es la de insertar un nuevo producto en el sistema.
  - **Descripción:** Únicamente los usuarios con rol de administrador serán los que puedan dar de alta un producto nuevo en el sistema.
  - **Actor principal:** Administrador.
  - **Precondiciones:** El sistema ha de poder facilitar la opción de alta producto al administrador. El nuevo producto no ha de existir y es necesaria la existencia de alguna categoría y subcategoría a la que poder asociarlo. El administrador podrá cerrar sesión en cualquier momento.
  - **Secuencia normal**

    1. Se realiza la conexión por parte del Administrador a la Web.
    2. El usuario entra en el sistema como administrador.
    3. El sistema muestra las diferentes funcionalidades al administrador.
    4. El administrador selecciona la opción alta producto.
    5. El sistema muestra el formulario de alta productos al Administrador pidiéndole categoría, subcategoría, nombre, descripción, imagen, stock, precio y si el producto se trata de una oferta.
    6. El usuario introduce todos los datos pedidos por el formulario y pulsa enviar.

7. El sistema comprueba los datos introducidos.
8. El sistema guardará el nuevo producto en la BBDD.

- **Secuencia alternativa**

  *se repetirán los pasos del 1 al 8 antas veces como el administrador lo desee

  7.1. Cualquiera de los datos introducidos son erróneos.

  71.1. El sistema avisa del error y vuelve al punto 5.

- **Poscondiciones:** Los productos han de ser almacenados en la BBDD.

- **Baja producto**
  - **Objetivos asociados:** El objetivo es llegar a realizar la eliminación de los productos escogidos por el administrador del catalogo de la tienda.
  - **Descripción:** Únicamente los usuarios con rol de administrador serán los que puedan borrar un producto del catálogo.
  - **Actor principal:** Administrador.
  - **Precondiciones:** El sistema ha de poder facilitar la opción de eliminar productos del catálogo al administrador. El administrador podrá cerrar sesión en cualquier momento.
  - **Secuencia normal**
    1. Se realiza la conexión por parte del Administrador a la Web.
    2. El usuario entra en el sistema como administrador.
    3. El sistema muestra las diferentes funcionalidades al administrador.
    4. El administrador escoge la opción modificar producto.
    5. El sistema muestra el formulario para llevar a cabo las modificaciones de los productos al Administrador.
    6. El administrador selecciona un producto de la tabla y pulsa modificar.
    7. El sistema muestra un formulario en el que aparecerán los datos actuales del producto nombre, descripción, stock, precio, imagen, si está de oferta o esta dentro del catalogo.
    8. El usuario activa la pestaña de “sacarlo del catalogo” y pulsa aceptar.
    9. El sistema comprueba los datos.
    10. El sistema realiza la baja lógica del producto en la BBDD sacándolo así del catalogo.

- **Secuencia alternativa:**

  *se repetirán los pasos del 1 al 10 tantas veces como el administrador lo desee.
- **Poscondiciones:** Se ha de realizar la baja lógica de los productos en la BBDD, en el futuro no saldrá en el catalogo.

- **Modificar producto**
  - **Objetivos asociados:** El objetivo es llegar a realizar la modificación de cualquiera de los datos de los productos existentes en la BBDD.
  - **Descripción:** Únicamente los usuarios con rol de administrador serán los que puedan modificar un producto.
  - **Actor principal:** Administrador.
  - **Precondiciones:** El sistema ha de poder facilitar la opción de modificar productos al administrador. El administrador podrá cerrar sesión en cualquier momento.
  - **Secuencia normal**
    1. Se realiza la conexión por parte del Administrador a la Web.
    2. El usuario entra en el sistema como administrador.
    3. El sistema muestra las diferentes funcionalidades al administrador.
    4. El administrador selecciona la opción modificar producto.
    5. El sistema muestra el formulario para llevar a cabo las modificaciones de los productos al Administrador.
    6. El administrador selecciona el producto y pulsa modificar producto.
    7. El sistema muestra los datos actuales pertenecientes al producto seleccionado, nombre, descripción, stock, precio, imagen, si es una oferta o está actualmente fuera de catalogo.
    8. El administrador modifica los datos que desea y pulsa el botón aceptar.
    9. El sistema comprueba los nuevos datos introducidos.
    10. El sistema actualiza los cambios realizados en el producto en la BBDD.
  - **Secuencia alternativa**

    *se repetirán los pasos del 1 al 10 tantas veces como el administrador lo desee

    9.1. Cualquiera de los datos introducidos son erróneos.

9.1.1 El sistema avisa del error y vuelve al punto 5.

- **Poscondiciones:** Las modificaciones del producto han de quedar almacenadas en la BBDD.

Usar cesta

- **Objetivos asociados:** El objetivo es llegar a realizar la inserción o eliminación de productos seleccionados en la cesta para su posterior compra.
- **Descripción:** Tendremos en cuenta que esta funcionalidad podrá ser utilizada por cualquier persona que se conecte a la Web y por los clientes.
- **Actor principal:** Invitado, cliente.
- **Precondiciones:** Es necesaria la existencia de un catalogo de productos, el usuario no ha de estar forzosamente registrado para su utilización. El cliente podrá cerrar sesión en cualquier momento.
- **Secuencia normal**
  1. Se realiza la conexión por parte del usuario a la Web.
  2. El usuario navega por el catalogo, encuentra un producto y lo añade a la cesta de la compra.
  3. El sistema mostrará la cesta actualizada con el producto anteriormente añadido. Dando la opción al usuario de aumentar la cantidad de dicho producto, actualizar la cesta o de eliminar el producto de la cesta. Igualmente permitirá seguir con la compra o tramitar el pedido.
  4. El usuario escoge la opción seguir comprando.
  5. El sistema le dirige al catálogo para seguir con el proceso.
  6. El usuario abandona la Web.
- **Secuencia alternativa:**

  *se repetirán los pasos del 1 al 6 tantas veces como el usuario lo desee.

  4.1. El usuario escoge modificar la cantidad del producto.

  4.1.1. El sistema comprueba los datos, informando de los posibles errores al usuario.

  4.2. El usuario escoge la opción borrar producto.

  4.2.1. El sistema eliminará el producto de la cesta.

  4.3. El usuario escoge la opción actualizar.

  4.3.1. El sistema actualiza la cesta.

4.4. El usuario escoge la opción tramitar pedido.

4.4.1. Se lanza el caso de uso “Entrar al sistema”. El sistema obligará al usuario a registrarse, puesto que está conectado como invitado y a partir de aquí es necesaria la identificación de usuario para la posterior tramitación del pedido.

- **Poscondiciones:** La cesta queda en óptimas condiciones para ser comprada y continuar con el proceso de “Hacer pedido”.

Usar catálogo

- **Objetivos asociados:** Consultar el catálogo de productos.
- **Descripción:** Uno de los principales objetivos de una tienda virtual es poder visualizar aquello que en un futuro se va a comprar. Por ello la existencia de esta funcionalidad.
- **Actor principal:** cualquier usuario que se conecto a la Web independientemente de su perfil.
- **Precondiciones:** Han de existir productos dados de alta en la BBDD así como las categorías a las que éstos pertenecen.
- **Secuencia normal:**
  1. Se realiza la conexión por parte del usuario a la Web.
  2. El sistema muestra en su página principal todos lo productos “ofertados” que existen y da la opción al usuario de buscar los productos a través de las diferentes categorías y subcategorías que actualmente hay en la BBDD que contengan algún producto.
  3. El usuario realiza la búsqueda de los productos y añade un determinado producto a la cesta.
  4. Se lanza el caso de uso “Usar cesta”.
  5. Se repiten los apartados del 1 al 5 tantas veces como el usuario lo desee.
- **Secuencia alternativa:**

4.1.El usuario utiliza el buscador para buscar productos.

4.1.1 El usuario introduce el nombre del producto en el buscador y pulsa utilizar el buscador.

4.1.2. El sistema realiza la búsqueda del producto e informa de los resultados al usuario.

4.1.3. Se lanza el caso de uso “Usar cesta”.

- **Poscondiciones:** El usuario tiene que haber navegado tanto tiempo y manera como quiera por el catalogo. El catalogo ha de mostrarse utilizando paginación.

Gestión ventas

- **Objetivos asociados:** Cambiar el estado de las ventas.
- **Descripción:** El administrador utilizará esta funcionalidad para llevar a cabo el cambio de estado de las ventas realizadas en función si el cliente ha pagado la venta o ha cancelado la compra.
- **Actor principal:** Administrador
- **Precondiciones:** Han de existir ventas con estado “EnCurso” en la BBDD.
- **Secuencia normal:**
  1. Se realiza la conexión por parte del administrador a la Web.
  2. El usuario entra en el sistema como administrador.
  3. El sistema muestra las diferentes funcionalidades al administrador.
  4. El administrador escoge la opción gestión de ventas.
  5. El sistema muestra el formulario para llevar a cabo las modificaciones de los estados de las ventas. En el formulario aparecerán todas las ventas que actualmente tienen su estado en “EnCurso”.
  6. El administrador selecciona la venta y pulsa en “Finalizada Pagada”.
  7. El sistema cambia el estado de la cesta a “FinalizadaPagada” y guarda la fecha actual.
- **Secuencia alternativa:**

  *se repetirán los pasos del 1 al 5 tantas veces como el usuario lo desee o existan ventas con el estado “EnCurso” en la BBDD.

  6.1. Administrador pulsa el botón Finalizada Impagada.

  6.1.1. El sistema cambia el estado de la cesta a “FinalizadaImpagada”, la fecha actual y devuelve el stock de los productos de la cesta a la BBDD.
- **Poscondiciones:** La información ha de quedar almacenada correctamente en la BBDD.

Hacer pedido

- **Objetivos asociados:** Llevar a cabo la compra de lo productos existentes en la cesta.
- **Descripción:** El objetivo principal de toda tienda virtual es conseguir que el cliente a través de los diferentes pasos realizados por la Web, consiga llevar a cabo una compra de la forma más fácil y clara posible.
- **Actor principal:** Cliente
- **Precondiciones:** El cliente ha de haber navegado por el catalogo, seleccionado los producto que le hayan interesado y depositarlos en la cesta de la compra (caso de uso usar cesta).
- **Secuencia normal:**
  1. Se lleva a cabo el caso de uso Usar cesta.
  2. El cliente escoge la opción tramitar pedido.
  3. El sistema muestra al cliente el formulario con los datos del cliente, los datos de la cesta y la forma de pago dándole la opción de cambiar cantidades o borrar productos. Aparecerán el botón PayPal y Cancelar pedido.
  4. El cliente pulsa el botón PayPal.
  5. El sistema dirige al cliente a www.PayPal.com.
  6. El cliente entra en www.PayPal.com para realizar el pago del pedido.
  7. Paypal notifica el pago a través de la cuenta PayPal del administrador.
  8. El administrador sirve el pedido.
  9. El administrador lanza el caso de uso Gestión de ventas.
- **Secuencia alternativa:**

  4.1 El cliente pulsa cancelar pedido.

  4.1.1. El sistema borra el contenido de la cesta y dirige al cliente a la página principal.

  4.2 El cliente cambia cantidades de los productos.

  4.2.1. El sistema comprueba el stock de los productos modificados

  4.2.2. En caso de error se avisa al usuario.

  4.3 El cliente pulsa borrar.

  4.3.1. El sistema elimina el producto de la cesta.

- **Poscondiciones:** Una vez hecho efectivo el pago por parte del cliente en www.paypal.com el administrador se compromete al envío del mismo al cliente. Toda la información de la venta así como sus posibles nuevos estados y de los productos, en especial el stock han de quedar correctamente actualizados en la base de datos.

7. Diseño

En esta fase se procederá al diseño de la aplicación, utilizando como herramienta principal los diferentes diagramas de clases, diseñados para cumplir con las funcionalidades que se han de implementar posteriormente. Teniendo en cuenta que la tecnología JavaServer Faces está basada en el patrón Modelo Vista Controlador, hemos decidido aplicar el patrón capas, ya que nos permitirá realizar una aplicación mantenible y ampliable. Otra característica del patrón capas es que nos permitirá realizar cambios en la base de datos sin que estos afecten al resto de la aplicación, hasta el punto que podemos cambiar de sistema gestor de bases de datos (SGBD) afectando únicamente a la capa encargada de interactuar directamente con los datos. Las capas serán Presentación, Aplicación (en nuestro proyecto, domotechonline), Dominio y Persistencia.

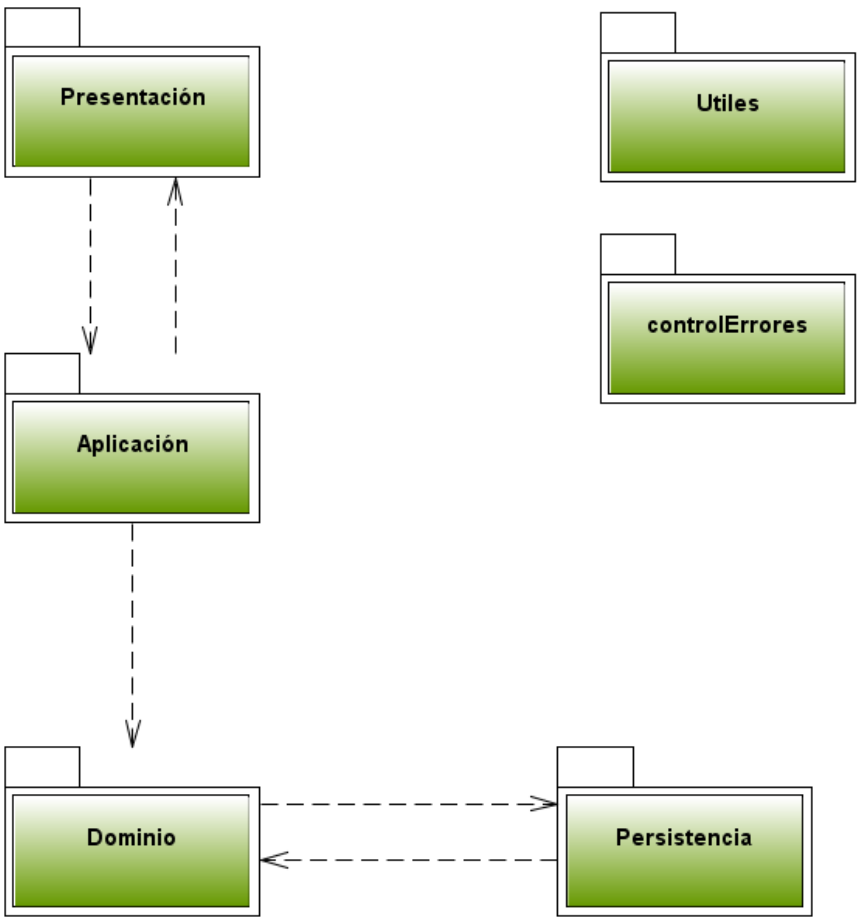

Figura 19. Diseño de paquetes de la aplicación

En la figura 19, se pueden observar los paquetes que componen la arquitectura de la aplicación, así como los acoplamientos entre las diferentas capas, dándonos pistas sobre cual va a ser el comportamiento de cada una de las capas. El aspecto más importante a destacar es que la capa Dominio, encargada de la lógica del negocio, está totalmente aislada de la Presentación, siendo únicamente consultada por la Aplicación y la Persistencia. La capa Aplicación y Dominio son paso obligatorio entre Presentación y Persistencia, de esta manera nunca tendrá acceso directo a los datos. Debido a la localización de las interfaces en el Dominio, siempre que la Aplicación desee hablar con Persistencia, deberá pasar por el Dominio. Por último, la Persistencia se acoplará con el Dominio para montar los objetos.

Haciendo una relación entre el patrón Modelo Vista Controlador y el patrón capas, llegamos a la conclusión de que la capa Presentación se corresponde con la Vista, la capa Aplicación lo hace con el Controlador y por último la capa Dominio lo hace con el Modelo.

7.1 Diseño del Dominio

La capa Dominio se corresponde con los objetos reales, tiene como principal objetivo encapsular los datos referentes a cada objeto y facilitarlos a las capas que los soliciten. Hemos dividido la capa Dominio en sub-capas, creando paquetes internos dentro del Dominio obteniendo el siguiente diseño.

A continuación pasaremos a detallar la figura 20.

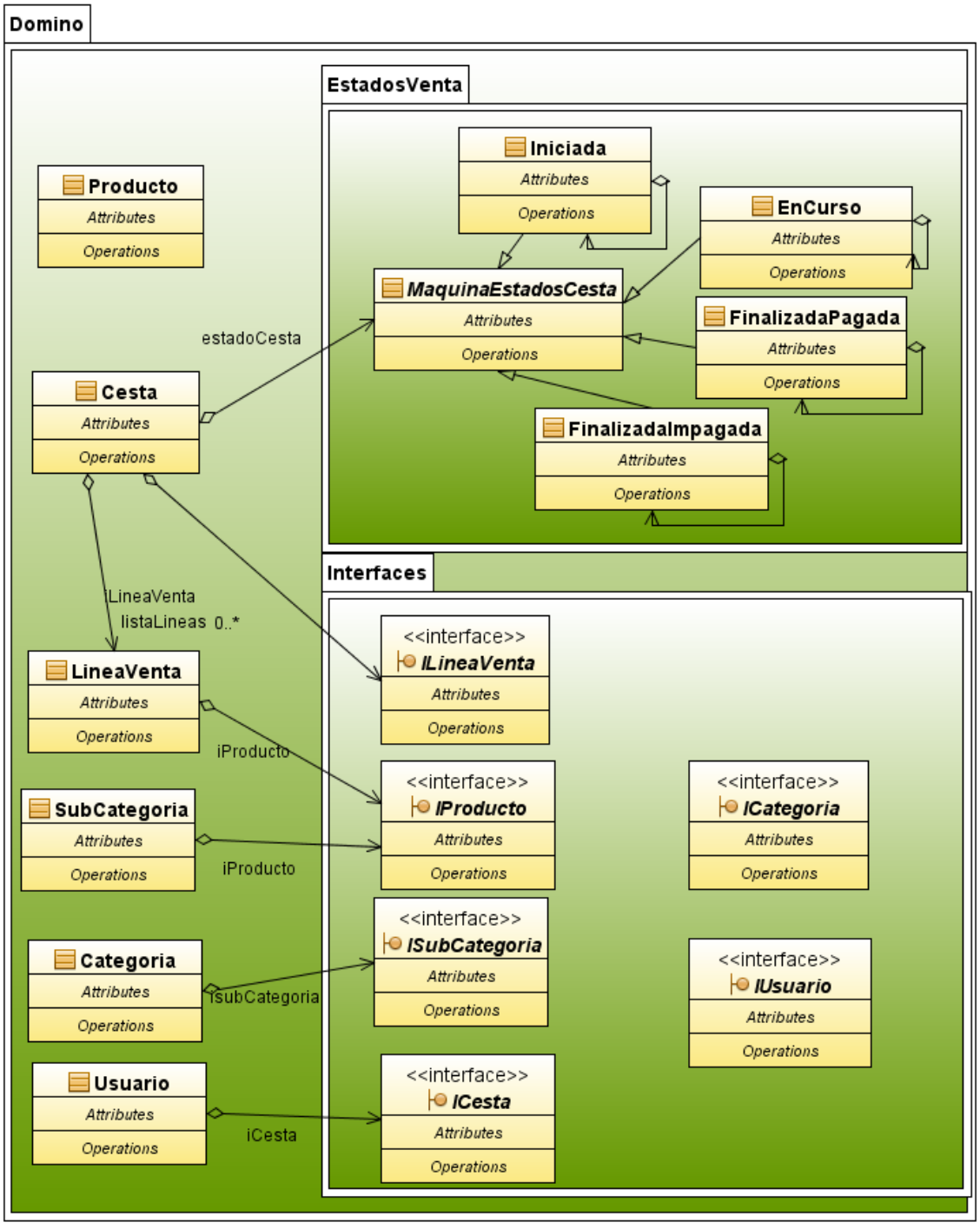

Figura 20. Diseño dominio

Como se puede apreciar, aparece la creación del paquete EstadosVenta y el paquete Interfaces los cuales contienen la máquina de estados y las diferentes interfaces.

La máquina de estados se encargará de ir actualizando el estado de la cesta. Las interfaces son las encargadas de conectar el Dominio y la aplicación con la Persistencia.

La cesta contiene un atributo de la clase MaquinaEstadosCesta, el estado de la cesta será de cualquiera de las clases singleton que extiende de la MaquinaEstadosVenta, la razón de que sean singleton es que la cesta únicamente puede tener un estado en cada momento.

Vamos a ver cada una de las clases en detalle.

7.1.1. Clase cesta

El siguiente diagrama UML se corresponde con la clase Cesta, podemos observar las clases y las capas con las que interactúa así como los métodos y los atributos que la componen.

Destacaremos los atributos de la clase idCesta, fechaCreacion, idUsuario, total, estado, una instancia a la interfaz ILineaVenta y listaLineas, una colección de objetos de la clase LineaVenta. También podemos observar el tipo de datos utilizados en la clase, int, double, String, ArrayList y Timestamp.

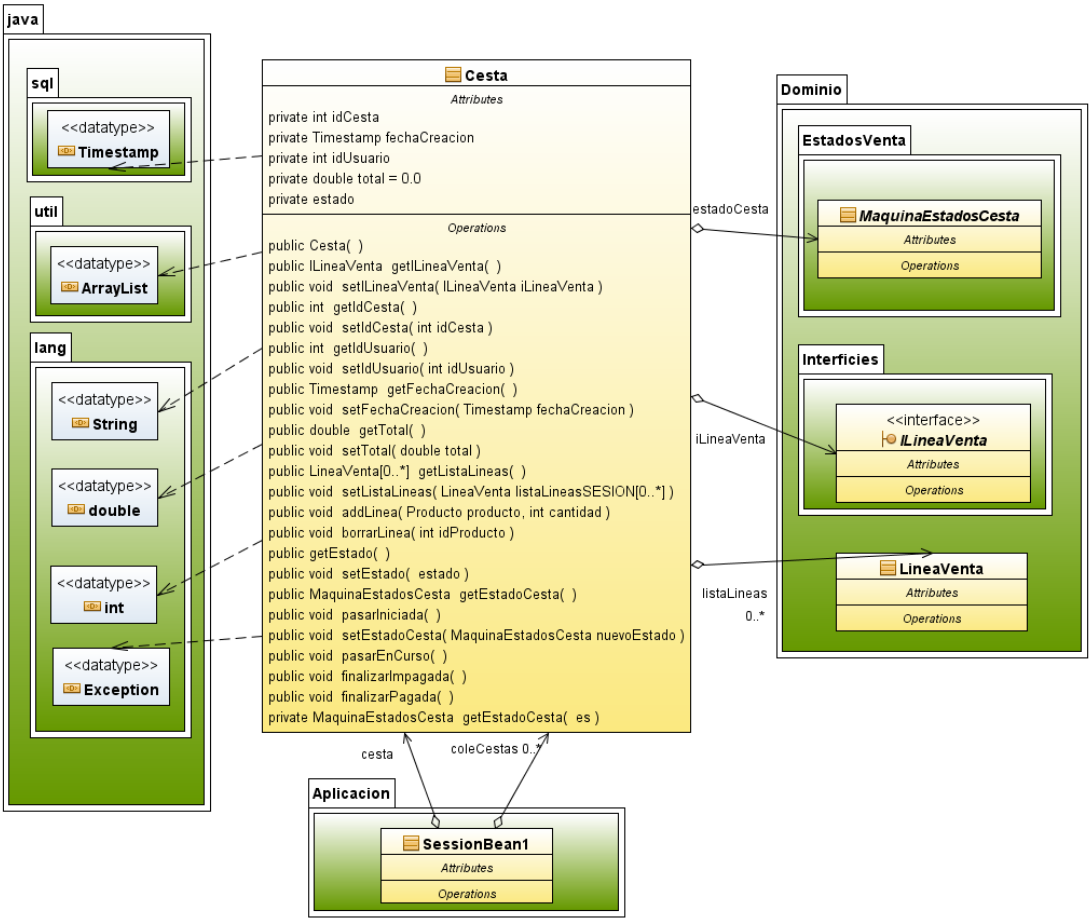

Figura 21. Clase cesta

En cuanto a métodos, los referentes a la máquina de estados, el método addLineaVenta() y borrarLineaVenta(), los cuales nos permiten añadir y borrar líneas a la cesta que está actualmente en sesión. Podemos observar como la clase SessionBean1 tiene dos referencias a la clase cesta, coleCestas y cesta.

7.1.2. Clase LineaVenta

En éste caso la clase LineaVenta es utilizada por la clase Cesta ya que ésta alberga una colección. Destacaremos el atributo de la interfaz IProducto por el cual se accede a la Persistencia así como los atributos idLineaVenta, cantidad, idProducto, idCesta y total. Cada uno de los atributos cuenta con sus métodos get y set.

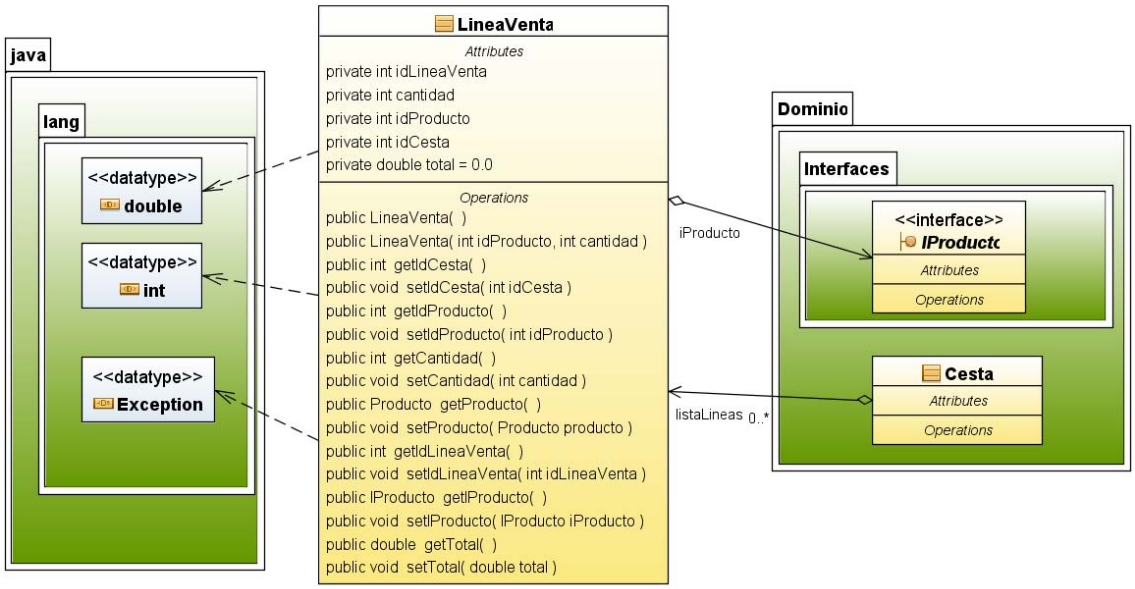

Figura 22. Clase LineaVenta

7.1.3. Clase producto

De la clase producto interesan atributos como el idProducto, nombre, descripción, imagen, precio, estoc, idSubCategoria, oferta y borrado.

Será la clase SessionBean1 de la capa Aplicación la que se acople con la clase producto, ya que tiene diferentes atributos de esta clase, listaProductosEncontrados, listaProductos, listaTODOSProductos, listaProductosOFERTA y producto.

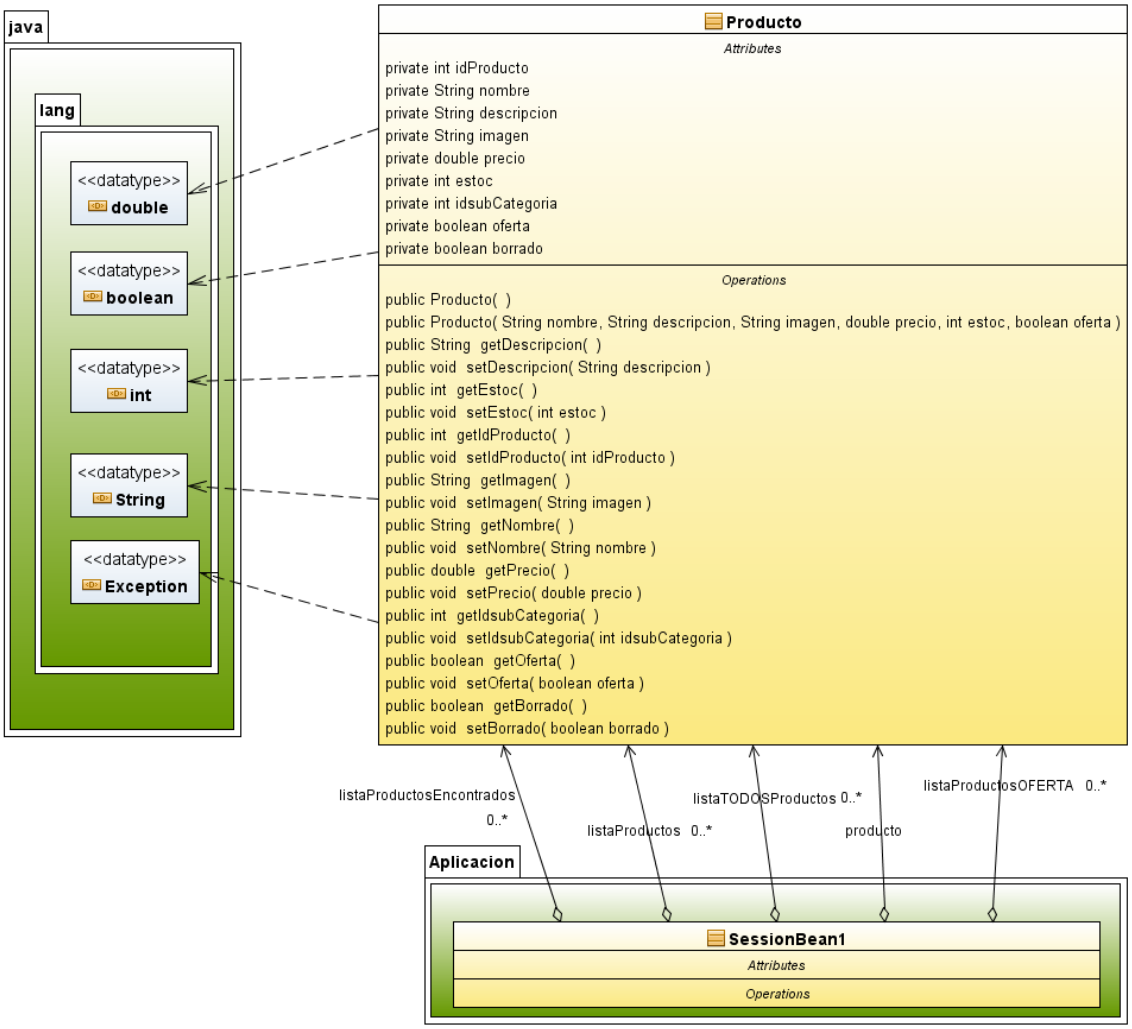

Figura 23. Clase producto

7.1.4. Clase categoría

La clase Categoría cuenta con tres atributos, idCategoria, nombre y un tercer atributo de tipo ISubCategoria, interfaz localizada en paquete Interfaces de la capa Dominio. En

este caso igual que en las clases Cesta o Producto será la clase SessionBean1 de la capa Aplicación será la que se acople con la clase Categoría.

A destacar los métodos redefinidos de la súper clase, Option necesarios para poder utilizar los componentes de JavaServer Faces.

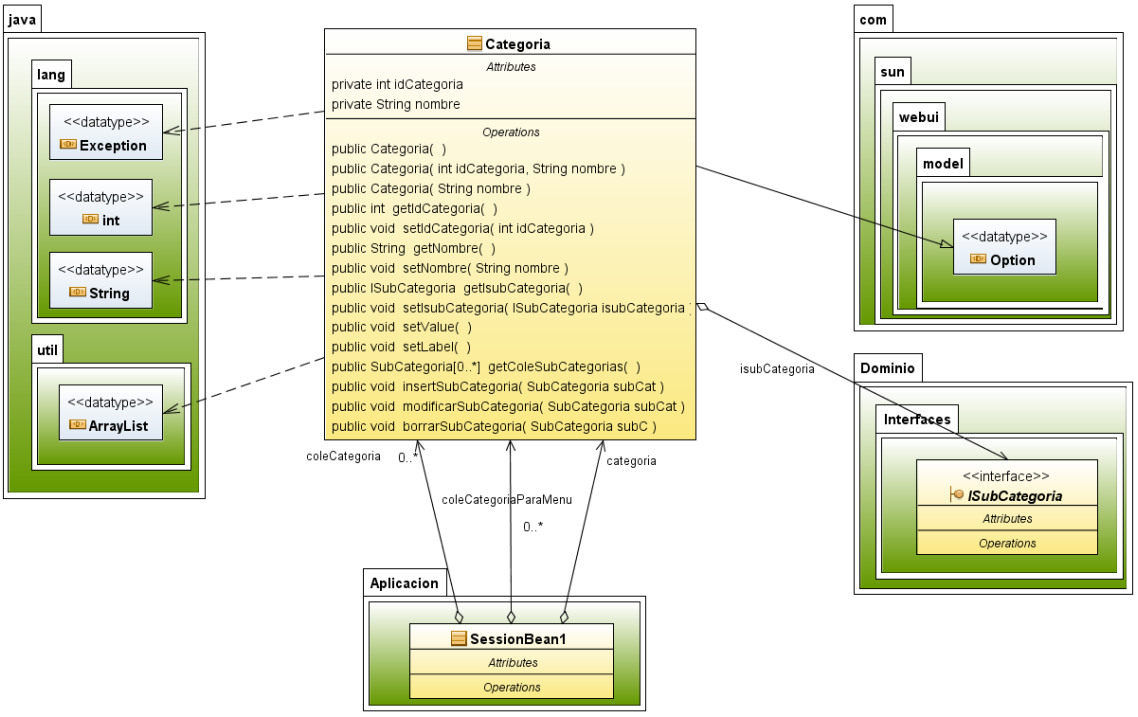

Figura 24. Clase categoría

7.1.5. Clase subcategoría

La clase Subcategoría al igual que la clase Categoría extiende a la clase Option para poder mostrar la información en la Presentación. Contiene los atributos idSubCategoria, nombre, imagen y idCategoria además de un atributo iProducto de la interfaz de la capa Dominio IProducto.

Los métodos getColeSubCategorias(), insertSubCategoria(), modificarSubCategoria() y borrarSubCategoria() utilizarán el atributo de la interfaz IProducto para llevar a cabo funcionalidades como alta, baja y modificación de SubCategoría (Véase figura 25).

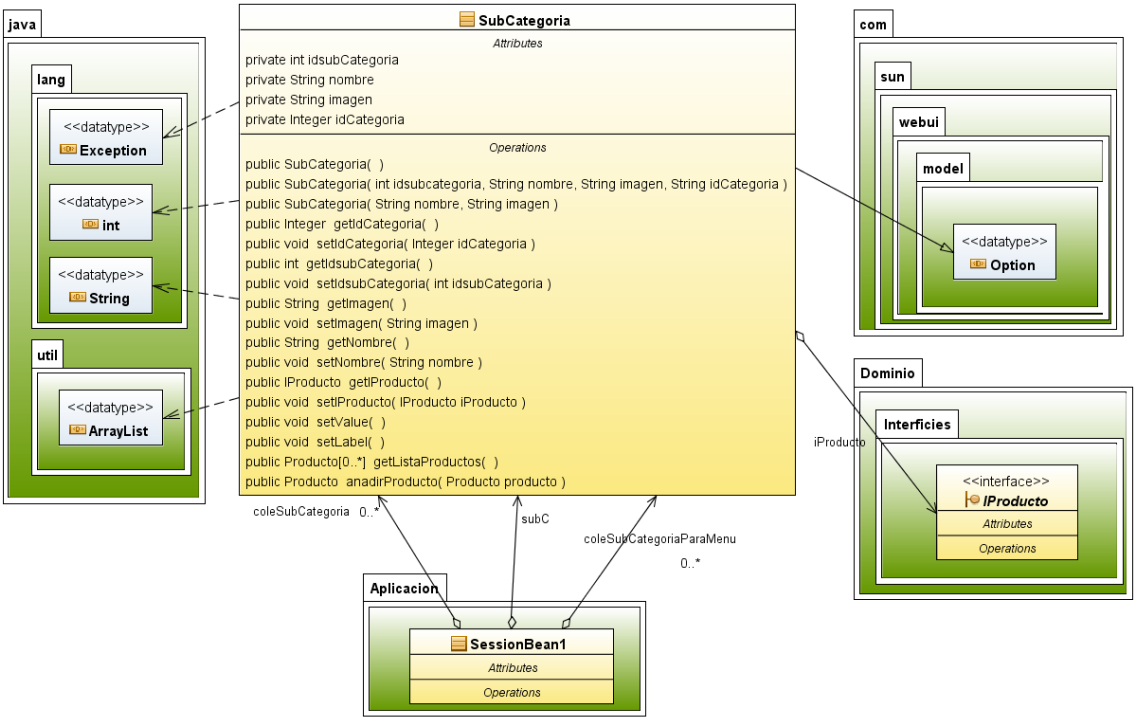

Figura 25. Clase Subcategoría

7.1.6. Clase usuario

Nuevamente la clase SessionBean1 de la Aplicación será la que se acople con el Dominio en este caso con la clase Usuario. La interfaz ICesta estará presente en la clase Usuario mediante el atributo iCesta. Gracias a este atributo y el método GuardarDatosCesta() se accederá a la base de datos para guardar la cesta comprada por el cliente. La clase usuario estará presente en los casos de uso alta, baja y modificación usuario.

Los atributos son idUsuario, nombre, apellidos, DNI, dirección, población, provincia, país, códigoPostal, email, contraseña, teléfono, fechaBaja y rol. Todos estos atributos tendrán sus respectivos métodos get y set (Véase figura 26).

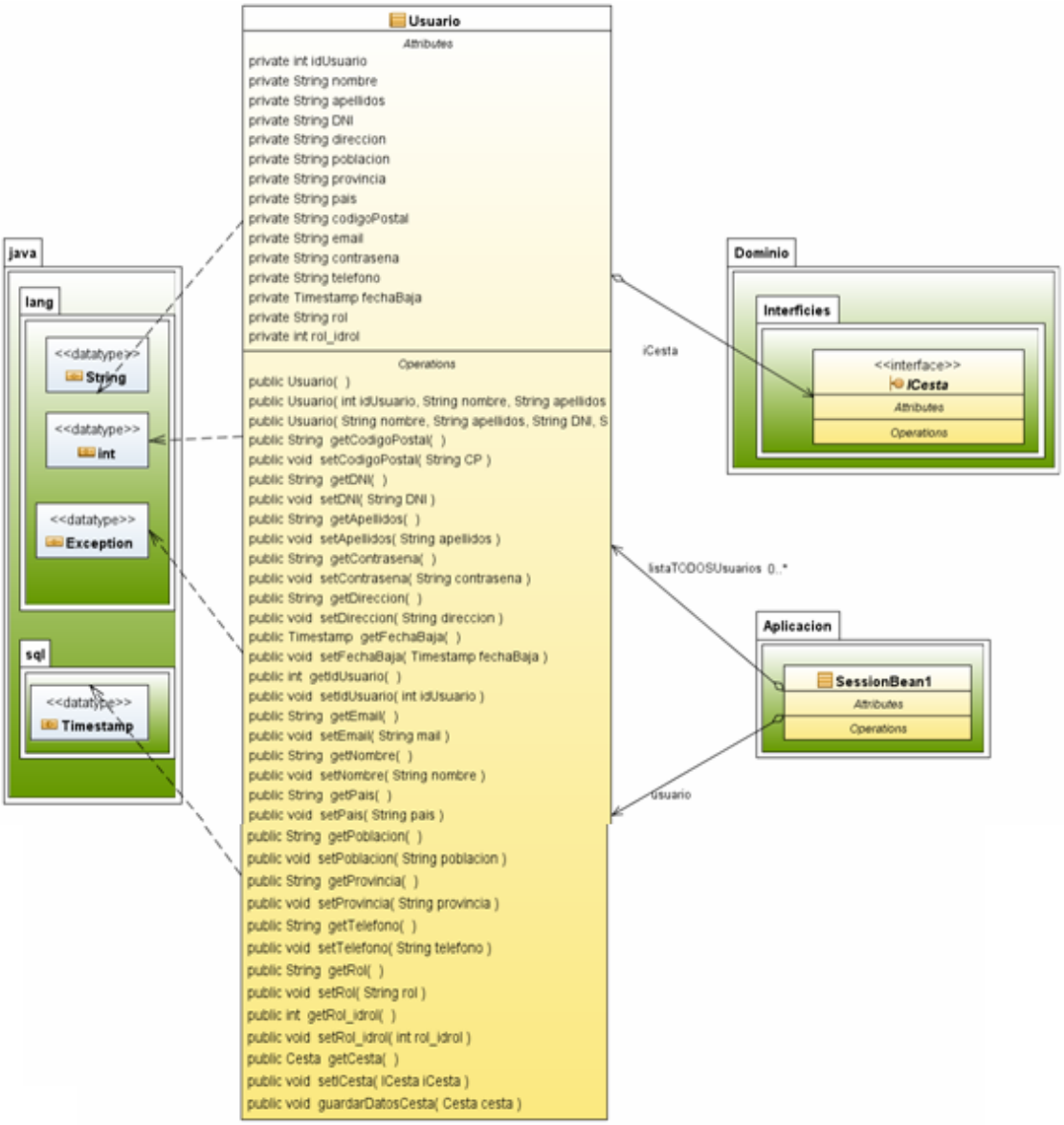

Figura 26. Clase usuario

7.2. Diseño Persistencia

La capa Persistencia tiene como principal objetivo aislar la base de datos del resto de la aplicación, consiguiendo el mínimo acoplamiento posible con el resto de las capas. Gracias a esta capa, las modificaciones en la base de datos supondrían los mínimos cambios en los métodos de la aplicación.

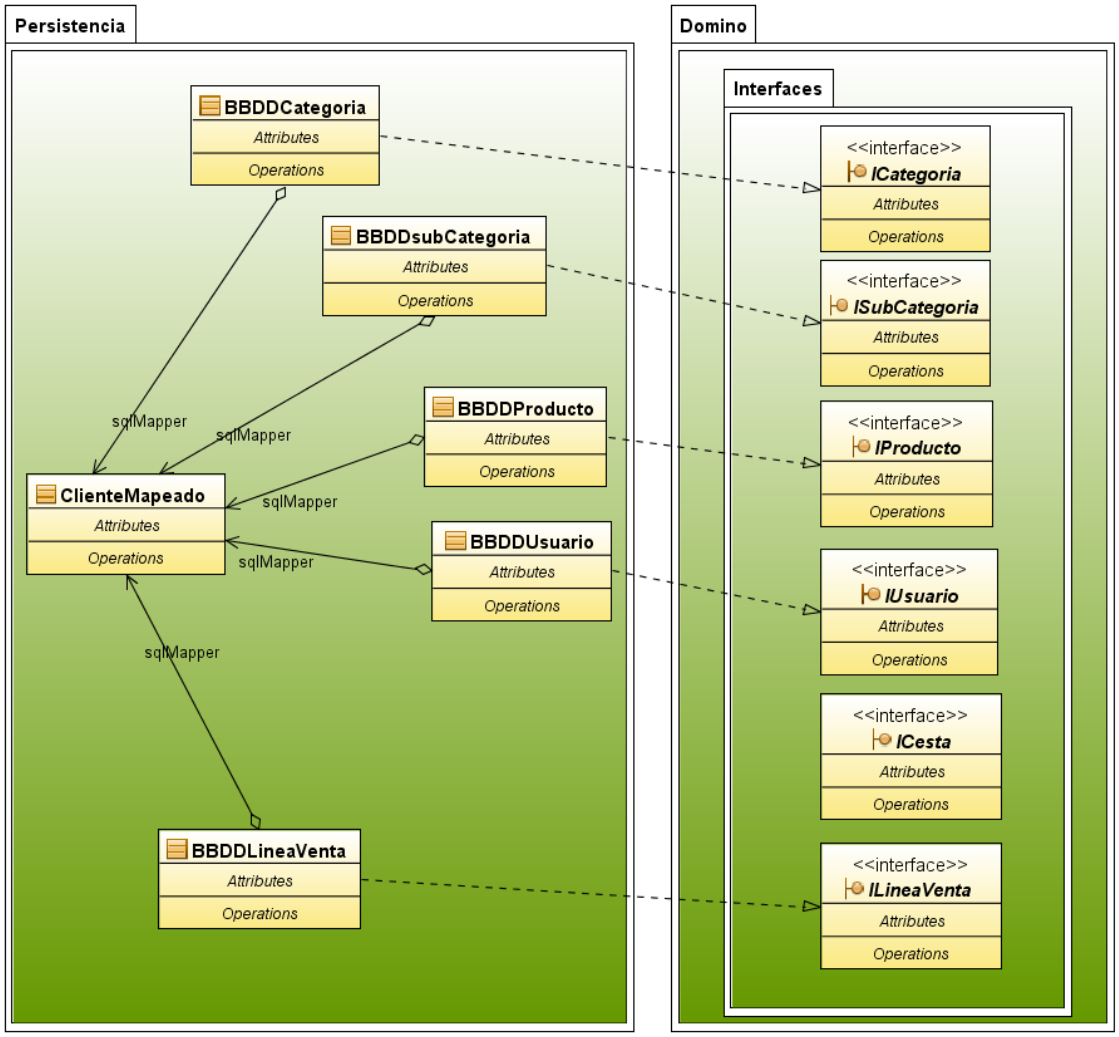

Figura 27. Diseño Persistencia

Cada una de las clases de la capa Persistencia, tiene un atributo de la clase ClienteMapeado, la cual se encarga de llamar, gracias a la librería iBatis, al fichero SqlMapConfig.xml con el que se obtiene la conexión a la base de datos así como el mapeo de los ficheros .xml donde se albergan las consultas a la base de datos. Por lo tanto habrá un fichero .xml por cada una de las clases de la capa Persistencia ubicados en el paquete Persistencia.sql.

Las clases de la capa Persistencia implementan las interfaces de la capa Dominio, siendo éstas la única manera de acceder a los datos.

7.3. Patrones de diseño utilizados

Durante el diseño de la aplicación, y teniendo en cuenta las funcionalidades a desarrollar, consideramos que la utilización de patrones de diseño nos podría ser de gran ayuda. Anteriormente se ha hecho referencia a patrones tales como el patrón capas o el Modelo Vista Controlador, en éste punto nos centraremos en la máquina de estados también llamado patrón estados y en el patrón fachada. El objetivo de la utilización de los patrones de diseño es la búsqueda de soluciones a problemas comunes en el desarrollo de software.

7.3.1. Máquina de estados [14]

Permite que un objeto modifique su comportamiento cada vez que cambie su estado interno. Hemos decidido utilizar este patrón ya que es condición indispensable que el objeto cesta, cambie su estado en función de las acciones producidas por los clientes.

Para llevar a cabo la implementación de éste patrón es necesario realizar un estudio previo de los posibles estados que puede llegar a tener la cesta. Ésta puede estar “Iniciada”, “EnCurso”, “FinalizadaPagada” o “FinalizadaImpagada”.

Como bien indica el nombre del estado, siempre que se cree una cesta nueva, lo hará con le estado de Iniciada. En el momento en que el cliente acepta el pago y sus condiciones, la cesta pasa de estar Iniciada a estar EnCurso.

Una vez se recibe el pago a través de PayPal de la cesta por parte del cliente, se producirá el cambio a FinalizadaPagada, si por el contrario no se lleva a cabo el pago, el estado al que se pasará es FinalizadaImpagada. El estado de la cesta quedará almacenado en la bases de datos una vez finalizada la venta.
Los estados y sus transiciones quedan descritos en el siguiente diagrama de estados.

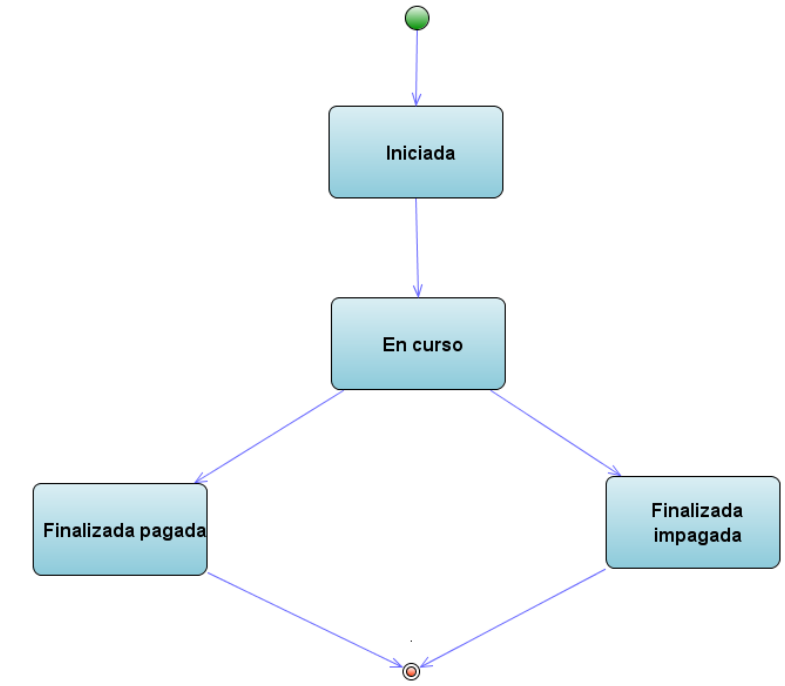

Figura 28. Máquina de estados Cesta

A la hora de diseñar la máquina de estados para la cesta, hemos creído conveniente que su ubicación fuera en un paquete llamado EstadosVenta dentro de la capa Dominio. Éste patrón cuenta con dos elementos claramente destacados, en primer lugar una clase abstracta por la que se accederá al cambio de estado y en sugundo lugar tantas clases como estados existan. Las clases estados son singletón y extienden a la clase abstracta.

Echemos un vistazo al diagrama de clases de la máquina estados (Véase figura 29).

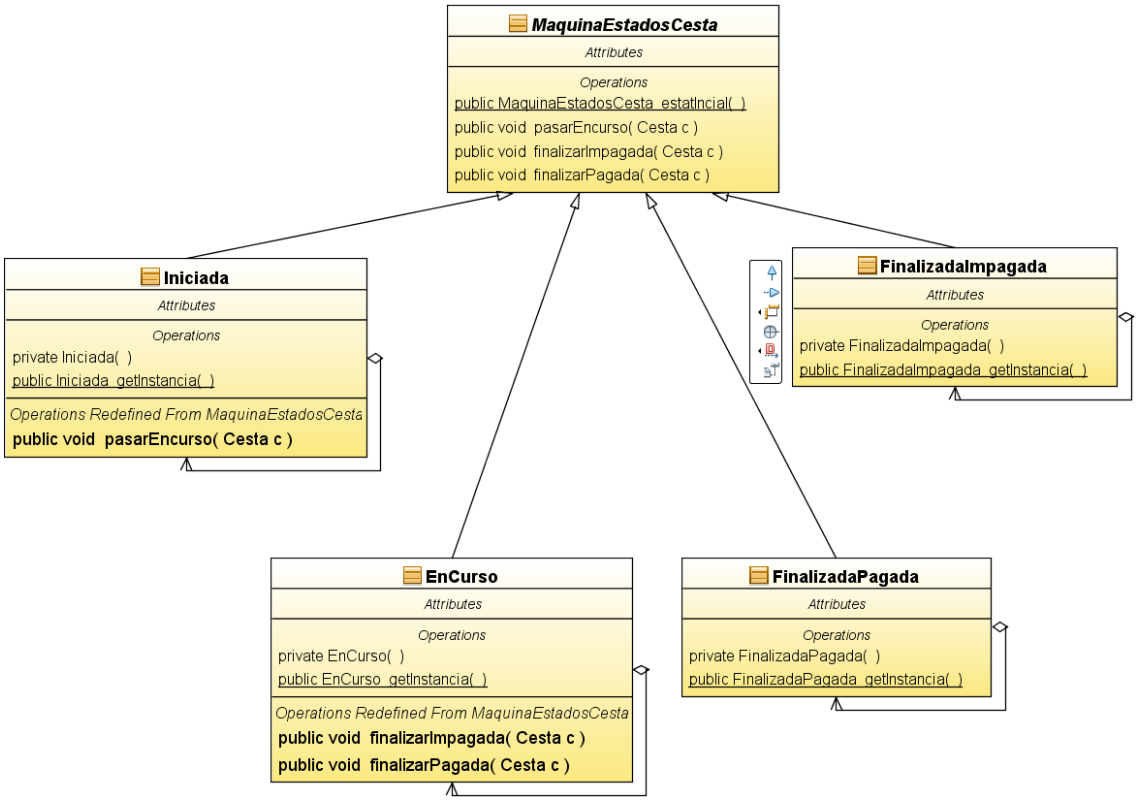

Figura 29. Diagrama de clases de la máquina de estados

Para cambiar el estado de la cesta, nos dirigiremos a los métodos implementados en la clase Cesta, los cuales utilizan el atributo estadoCesta para acceder al patrón estados.

7.3.2. Patrón fachada: FactoríaRegistre [14]

La FactoríaRegistre estará ubicada en la capa Aplicación y será instanciada por el controlador SessionBean1 de la misma capa. Este patrón nos permite el acceso a los datos sin tener que acoplarnos a la Persistencia ya que en la clase FactoriaRegistre existen métodos que nos devuelven instancias de las interfaces del Dominio, las cuales están implementadas por los registros de la Persistencia (Véase Figura 30).

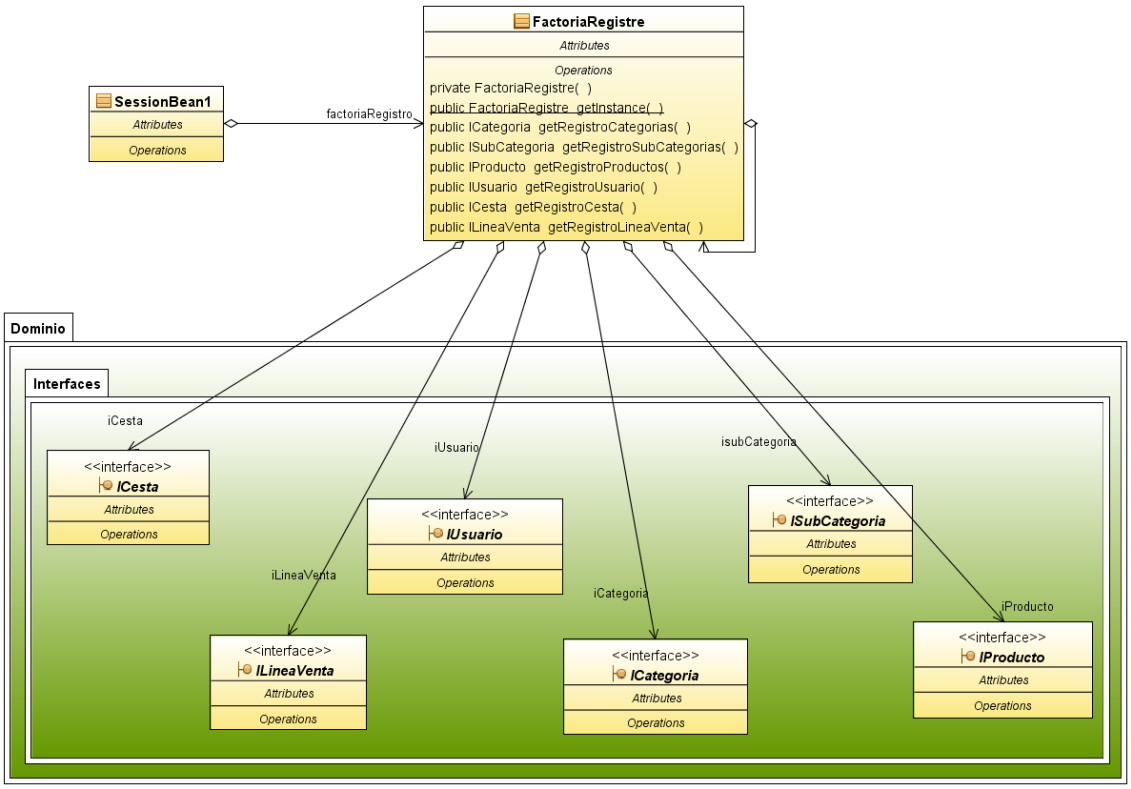

Figura 30. Patrón fachada

7.4. Diseño de la base de datos

La base de datos es el lugar donde la información quedará almacenada tras la ínter actuación entre los usuarios y el propio sistema. Es sin duda una pieza clave de la aplicación puesto que sin un gestor de bases de datos, la información se sin dejar constancia de los posibles cambios producidos. Como ya se ha comentado en la fase de planificación, el sistema gestor de bases de datos a utilizar es MySQL [6]. Veamos el modelo entidad-relación en el que quedan claramente descritas las tablas y sus relaciones a implementar (Véase figura 31).

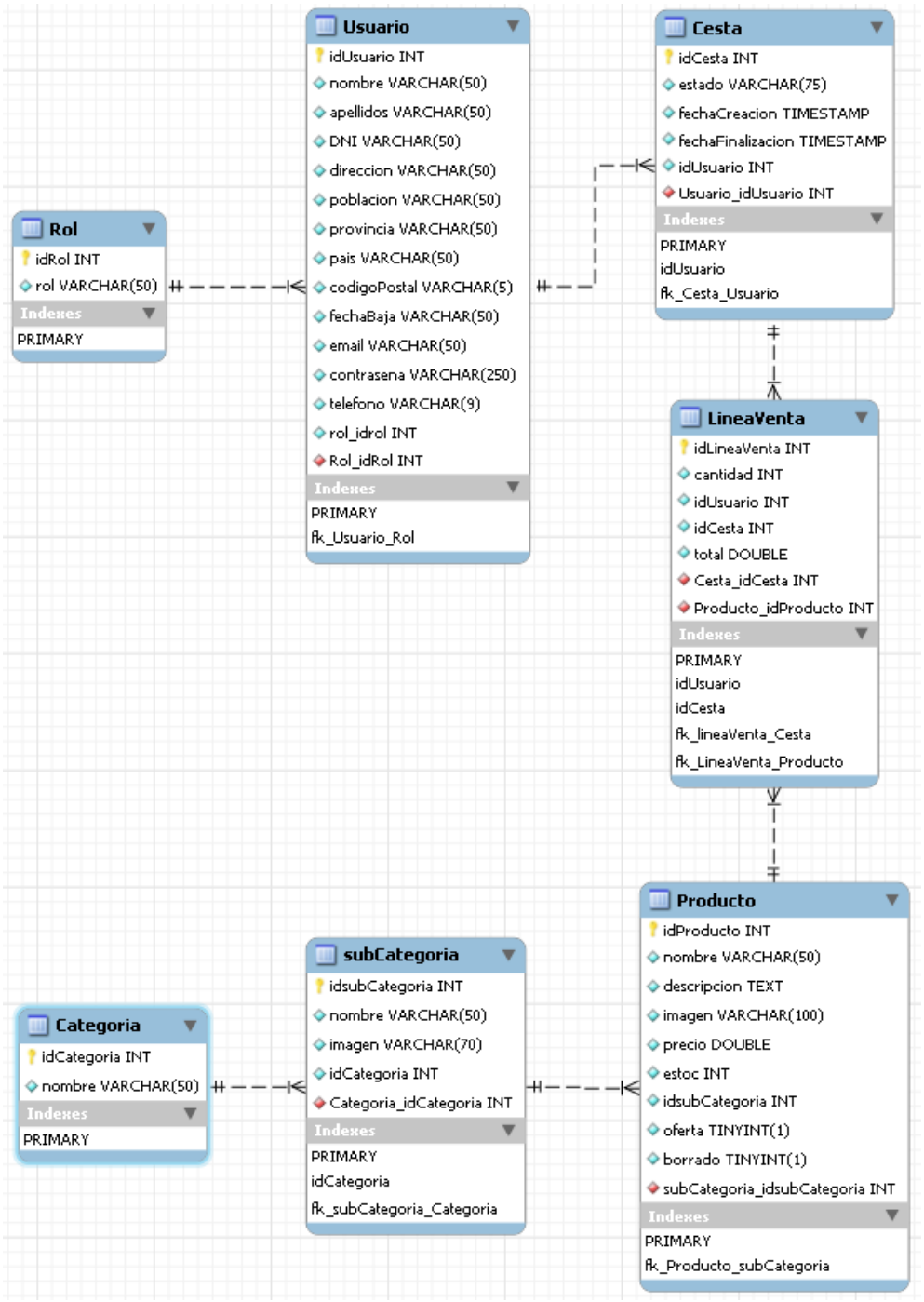

Figura 31. Diseño de la base de datos

Como podemos observar en el modelo de la base de datos, cada una de las tablas descritas, tienen su homónima en el Dominio a excepción de la tabla rol que se creyó conveniente pasarla por alto. Para llevar a cabo la implementación de una tienda virtual, la base de datos como mínimo precisa de las tablas Categoría, subCategoría, Producto, LineaVenta, Cesta y Usuario. Los atributos son los marcados con un diamante azul, las

claves primarias las podemos identificar por el logo de la llave amarilla y las claves foráneas por el diamante rojo.

Las restricciones son:

- Un usuario tendrá un único rol, aunque en la aplicación existen dos roles a almacenar en la base de datos cliente y administrador.
- Un usuario puede tener muchas cestas asociadas.
- Una cesta puede contener muchas líneas de venta.
- Una línea de venta pertenecerá a una única cesta y tendrá un único producto asociado.
- Un producto pertenecerá por un lado a una línea de venta y por otro a una subcategoría.
- Una subcategoría puede tener muchos productos.
- La subcategoría pertenecerá a una categoría pero la categoría puede tener muchas subcategorías asociadas.

Se llevará a cabo forzosamente el borrado lógico de información debido a las relaciones entre las tablas, es el caso de:

- No es permitido la existencia de LíneaVenta sin producto una vez creadas.
- No es permitido la existencia de cestas sin LíneaVenta una vez creadas.
- No es permitido la existencia de subcategorías sin productos una vez creadas.
- No es permitida la existencia de categorías sin subcategorías una vez creadas.

Algunos tipos de datos a comentar serían INT, DOUBLE, VARCHAR, TINYNT cuyo comportamiento es de boolean de cara a la aplicación y TIMESTAMP para los datos tipo fecha.

7.5. Diseño del paquete Útiles

El paquete Útiles es independiente al patrón capas ya que puede servir de apoyo a todos y cada uno de los paquetes de la aplicación. En nuestro caso, únicamente se acopla con las clases de la Presentación para el control de los diferentes formularios, y con la Aplicación, en la que se necesitarán los métodos de las clases Cifrado, GeneradorContrasena y sendMail. A continuación se explicarán brevemente cada una de las clases:

- **Cifrado.java**

La contraseña se cifrará a través de esta clase, el almacenamiento de la contraseña en la base de datos será cifrado.

- **GeneradorContrasena.java**

Para llevar a cabo el caso de uso Solicitar Contraseña y alta Cliente, es necesario la generación automática de una contraseña para enviársela al cliente de la tienda a su cuenta de correo electrónico. Para ello se utilizarán los métodos de la clase del paquete java.util.Random.

- **sendMail.java**

Su objetivo principal es enviar la contraseña previamente generada por la clase generadorContrasena.java, siendo pues condición indispensable para llevar a cabo las funcionalidades descritas en la clase anterior.

7.6. Diseño del paquete controlErrores

JavaServer Faces, cuenta con una metodología a la hora de validar y convertir datos de los formularios. Para validar los formularios, hay que tener un conocimiento previo sobre cuales son los validadores estándar que JSF provee, así como la manera de lanzar los mensajes.

En nuestro caso se ha optado por la utilización de validadores estándar como por ejemplo validateDoubleRange, el cual valida un valor double dentro de un rango predefinido. Los mensajes de error provenientes de las validaciones estándar están descritos en el fichero ErroresBundle.properties dentro del paquete controlErrores.

Pero en algunas ocasiones hemos tenido que crear nuestras propias reglas de validación, es el caso por ejemplo del control de stock o las direcciones de correo. Para ello, se ha creado la clase controlErrores.java,, la cual se caracteriza por implementar la interfaz Validator. Dicha implementación obliga a implementar un método llamado validate() el cual lanzará las excepciones cuando se produzca el error de validación.

Una vez creada la clase, lo único que tenemos que hacer es tener una referencia, con su respectivo get y set, en el controlador de la Aplicación encargado de gestionar los estados y atributos de las paginas .jsp de la Presentación y redefinir el atributo "validatorExpression" del componente a validar. Aunque más adelante se entrará en detalle, podemos ver a modo de ejemplo como redefinimos el atributo "validatorExpression" de un componente textField.

```
<webuijsf:textField binding="#{altaUsuario.textFieldCP}" id="textFieldCP" required="true"
    style="left: 216px; top: 264px; position: absolute" toolTip="Código."
    validatorExpression="#{altaUsuario.control.validate}"/>
```

Figura 32. Control de validación para un componente de textField

8. Implementación

En esta fase llevaremos a cabo la implementación de la aplicación, teniendo muy presente tanto lo detallado en la fase de Análisis como en la fase de Diseño. Procederemos a la programación de las diferentes clases y páginas jsp que componen las diferentes capas Presentación, Aplicación, Dominio y Persistencia. Intentáremos explicar cual ha sido el proceso a seguir, así como las partes clave en la implementación, para ello empezaremos por la capa Presentación e iremos recorriendo el flujo normal hasta llegar a la capa Persistencia dejando constancia de los posibles cambios producidos en los datos.

Que mejor manera para empezar, que ver uno de los casos de uso implementados desde su inicio. El caso de uso que mostraremos a continuación es Entrar al sistema. Hemos escogido esta funcionalidad porque se lleva a cabo el flujo normal de los datos entre las diferentes capas que componen la aplicación.

8.1. Archivos de configuración

Toda aplicación JavaServer Faces tiene un archivo de configuración llamado **faces-config.xml** contiene recursos de los bean y las reglas de navegación entre las paginas de la aplicación. Además del archivo de configuración, existen archivos que son necesarios para el servlet, estos son el archivo **web.xml** y el archivo al que se redireccionará al usuario como página principal de entrada al sistema, Page1.jsp.

El archivo de configuración **faces-config.xml**, contiene los archivos de validación estándar, la declaración de los diferentes ámbitos, todas las clases asociadas a las páginas jsp y todas las reglas de navegación, veamos unos ejemplos:

```
<?xml version='1.0' encoding='UTF-8'?>

<!-- ========== FULL CONFIGURATION FILE ================================== -->
<faces-config version="1.2"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-facesconfig_1_2.xsd">

    <application>
        <message-bundle>controlErrores.ErroresBundle</message-bundle>
    </application>

    <managed-bean>
        <managed-bean-name>SessionBean1</managed-bean-name>
        <managed-bean-class>domotechonline.SessionBean1</managed-bean-class>
        <managed-bean-scope>session</managed-bean-scope>
    </managed-bean>
    <managed-bean>
        <managed-bean-name>ApplicationBean1</managed-bean-name>
        <managed-bean-class>domotechonline.ApplicationBean1</managed-bean-class>
        <managed-bean-scope>application</managed-bean-scope>
    </managed-bean>
    <managed-bean>
        <managed-bean-name>RequestBean1</managed-bean-name>
        <managed-bean-class>domotechonline.RequestBean1</managed-bean-class>
        <managed-bean-scope>request</managed-bean-scope>
    </managed-bean>
    <managed-bean>
        <managed-bean-name>altaUsuario</managed-bean-name>
        <managed-bean-class>domotechonline.altaUsuario</managed-bean-class>
        <managed-bean-scope>request</managed-bean-scope>
    </managed-bean>

            <navigation-rule>
                <from-view-id>/datosEnvioFormaPago.jsp</from-view-id>
                <navigation-case>
                    <from-outcome>cerrarSesion</from-outcome>
                    <to-view-id>/Page1.jsp</to-view-id>
                </navigation-case>
                <navigation-case>
                    <from-outcome>modificarUSU</from-outcome>
                    <to-view-id>/modificacionUsuario.jsp</to-view-id>
                </navigation-case>
                <navigation-case>
                    <from-outcome>cancelarPedido</from-outcome>
                    <to-view-id>/Page1.jsp</to-view-id>
                </navigation-case>
                <navigation-case>
                    <from-outcome>cuentaUsu</from-outcome>
                    <to-view-id>/cuentaUsuario.jsp</to-view-id>
                </navigation-case>
                <navigation-case>
                    <from-outcome>pedidoOK</from-outcome>
                    <to-view-id>/Page1.jsp</to-view-id>
                </navigation-case>
            </navigation-rule>

        </faces-config>
```

Figura 33. Aspecto del archivo faces-config.xml

8.2. Página jsp de la capa Presentación

Para empezar, veamos cual es el código de una página jsp en blanco utilizando nuestro marco de trabajo.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    Document   : Page4
    Created on : 18-ene-2009, 12:09:23
    Author     : romculda
-->

<jsp:root version="2.1" xmlns:f="http://java.sun.com/jsf/core"
xmlns:h="http://java.sun.com/jsf/html"
xmlns:jsp="http://java.sun.com/JSP/Page"
xmlns:webuijsf="http://www.sun.com/webui/webuijsf">
<jsp:directive.page contentType="text/html;charset=UTF-8" pageEncoding="UTF-8"/>
<f:view>
    <webuijsf:page id="page1">
        <webuijsf:html id="html1">
            <webuijsf:head id="head1">
                <webuijsf:link id="link1" url="/resources/stylesheet.css"/>
            </webuijsf:head>
            <webuijsf:body id="body1" style="-rave-layout: grid">
                <webuijsf:form id="form1">
                </webuijsf:form>
            </webuijsf:body>
        </webuijsf:html>
    </webuijsf:page>
</f:view>
</jsp:root>
```

Figura 34. Aspecto página jsp

Visual Web JSF codifica mediante lenguaje xml sus páginas como podemos observar en la primera línea del código. También utiliza una librería de componentes específica, podemos ver la importación resaltada en el código anterior. Todas las referencias a las diferentes librerías utilizadas estarán en el tag que se encuentra resaltado en primer lugar. Los componentes a utilizar en nuestras páginas estarán situados dentro formulario, resaltado en segundo lugar.

A continuación, la página que se le muestra al usuario de la aplicación referente al caso de uso Entrar al sistema.

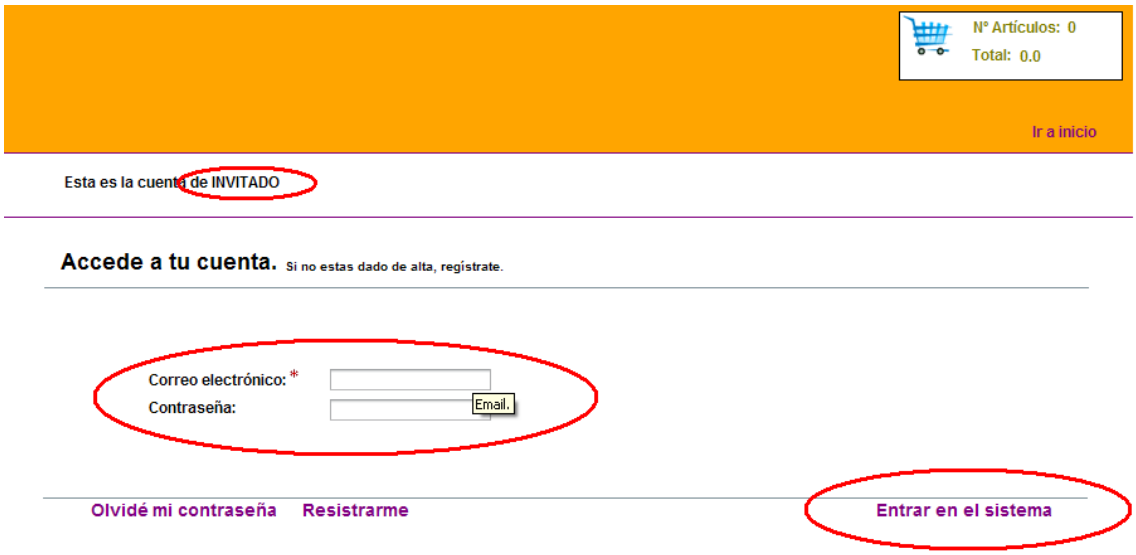

Figura 35. Caso de uso Entrar al sistema

Vemos que consta de 4 links que nos llevarán a otras funcionalidades (Ir a inicio, Olvidé mi contraseña, Registrarme y Entrar en el sistema), dos componentes textField y un componente label cuyo comportamiento es dinámico y que se nos muestra con el texto "INVITADO".

El código de la página Logarse.jsp de nuestra aplicación perteneciente a los componentes resaltados en la imagen anterior sería:

- El texto dinámico al que se hacía referencia anteriormente, se crea gracias a el componente de la librería xmlns:h="http://java.sun.com/jsf/html", esto lo podemos ver ya que se corresponden mediante la variable h. El cometido de este componente es visualizar, a través de la propiedad value, el nombre del cliente que ha entrado en el sistema, esto es posible gracias a la existencia del atributo usuarioSession en el controlador SessionBean1 de la capa Aplicación. Destacar el formato por el cual se llaman a las variables de la Aplicación en JSF (#{variable}).

```
<h:outputText value="Esta es la cuenta de #{SessionBean1.usuarioSesion}"/>
```

- Los campos de texto así como los labels serán componentes de la librería xmlns:webuijsf="http://www.sun.com/webui/webuijsf". La propiedad binding es la encargada de relacionar el componente, con un atributo del bean asociado a la jsp en

la capa Aplicación. También destacar la propiedad for del label3, creando dependencia con el textFieldEmail y forzando así a que sea campo obligatorio. Por otro lado la propiedad validatorExpression es por la que se llevará a cabo la validación del campo textFieldEmail, hace referencia al método validate del objeto control, cuya referencia está en la clase asociada.

```
<webuijsf:textField binding="#{Logarse.textFieldEmail}" id="textFieldEmail"
required="true" toolTip="Email."
validatorExpression="#{Logarse.control.validate}"/>

   <webuijsf:passwordField binding="#{Logarse.passwordFieldContrasena}"
   id="passwordFieldContrasena" toolTip="Contraseña."/>

<webuijsf:label for="textFieldEmail" id="label3" requiredIndicator="true"
text="Correo electrónico:"/>

<webuijsf:label id="label4" text="Contraseña:"/>
```

- El componente hyperlink pertenece a la misma librería, será el que mediante su propiedad actionExpression se ponga en contacto con el bean asociado a la jsp de la capa Aplicación a través del método hyperlinkEntrar_action().

```
<webuijsf:hyperlink actionExpression="#{Logarse.hyperlinkEntrar_action}"
id="hyperlinkEntrar" text="Entrar en el sistema"
toolTip="Pulsa para entrar en el sitema."/>
```

- Aunque no se le muestra en primera instancia al usuario, el componente message (también de la librería de nuestro framework), cambiará su estado cumpliendo con las normas de validación mostrando el error de validación producido. A destacar la propiedad for la cual asocia el mensaje de error con el textFieldEmail.

```
<webuijsf:message for="textFieldEmail" id="message2" showDetail="false"
showSummary="true"/>
```

- Para acabar el componente messageGroup se activará cuando se envíen mensajes globales del sistema desde los bean de la capa Aplicación, ya que tiene la propiedad showGlobalOnly activada.

```
<webuijsf:messageGroup binding="#{Logarse.messageGroup}" id="messageGroup"
showGlobalOnly="true"/>
```

8.3. Bean asociado en la capa Aplicación

El bean asociado a la página jsp anterior es Logarse.java, podemos observar que se trata de una clase java la cual extiende a AbstractPageBean. Continuando con el caso de uso Entrar al sistema, el bean tiene los siguientes atributos y el método hyperlinkEntrar_action(), acción del componente hyperlinkEntrar.

```
private StaticText staticTextUsuarioSesion = new StaticText();
private TextField textFieldEmail = new TextField();
private PasswordField passwordFieldContrasena = new PasswordField();
private Message message1 = new Message();
private MessageGroup messageGroup = new MessageGroup();
private controlErrores control;
```

Figura 36. Atributos clase Logarse.java

Observemos la existencia del atributo control de la clase controlErrores.java del paquete Útiles, por el que se llevará a cabo la validación del componente textFielEmail de la página jsp.

```
public String hyperlinkEntrar_action() throws Exception {
1   String mail=(String)this.textFieldEmail.getText();
    String pass=(String)this.passwordFieldContrasena.getText();

    if(pass==null || pass.equals("")){
        error("La contraseña es necesaria para entrar en el sistema.");
        return null;
    }
2   Cifrado ci=new Cifrado();
    pass = ci.getEncoded(pass);

    Usuario usu=this.getSessionBean1().getIUsuario().getUsuario(mail, pass); 3
        if(usu==null){
4       error("Validación incorrecta, vuelve a intentarlo.");
        return null;
        }
        else{
            this.getSessionBean1().setUsuario(usu); 5
        }
    if(this.getSessionBean1().getUsuario().getRol().equals("administrador")){
        return "adminlogado";
    }
    if(!this.getSessionBean1().getCesta().getListaLineas().isEmpty()){
        return "verCesta";
    }

  return "logado"; 6
}
```

Figura 37. Acción hyperlinkEntrar_action()

1- Capturamos el contenido de los componentes textField de la página jsp a partir de los atributos en el bean. Esto es gracias a la propiedad binding del componente.

2- Una vez obtenida la contraseña introducida por el usuario de la aplicación, la codificamos a través del método getEncoded() de la clase Cifrado en el paquete Útiles.

3- Todos los bean de la capa Aplicación tiene métodos por los que se obtienen referencias a los diferentes ámbitos Sesión, Aplicación y petición, será por aquí por donde realizaremos los cambios en los atributos de la sesión, mediante los métodos get y set. En éste caso la sesión tiene una instancia de la interfaz IUsuario por la que accedemos para coger el usuario de la base de datos con el mail y la contraseña introducidos.

```
public interface IUsuario {

     ArrayList<Usuario> getTODOSUsuarios()throws Exception;
     Usuario getUsuario(String email,String contrasena)throws Exception;
     boolean getUsuarioPorMail(Usuario usuario)throws Exception;
     void insertUsuario (Usuario usu) throws Exception ;
     void darBajaUsuario(Usuario usuario) throws Exception ;
     void modificarUsuario(Usuario usuario)throws Exception;
     void modificarContrasena(String correo,String contrasena)throws Exception;

}
```

Figura 38. Métodos interface IUsuario

Esta interfaz estará implementada por la clase BBDDUsuario de la capa Persistencia, para el desarrollo del caso de uso, nos centraremos en el método getUsuario().

```
public Usuario getUsuario(String email,String contrasena)throws Exception{
    Usuario usuario=new Usuario();
    usuario.setEmail(email);
    usuario.setContrasena(contrasena);

    usuario=(Usuario) sqlMapper.getInstancia().queryForObject("getUsuario", usuario);
    if(usuario!=null){
        usuario.setICesta(new BBDDCesta());
    }
    return usuario;
}
```

Figura 39. Implementación método getUsuario()

En el método anterior, devolveremos el usuario que se ha registrado. El atributo sqlMapper de la clase ClienteMapeado será el medio por el que se llevarán a cabo las futuras consultas. Más adelante explicaremos la metodología de iBATIS en la Persistencia.

4- En caso de que no hayan coincidencias en la base de datos, llamaremos al método error(), lanzando el error y retornando un null para acabar con el hilo de ejecución. Tanto los métodos que nos devuelven los diferentes ámbitos como el método error(), están implementados por las clases de la librería appbase.jar facilitada por el framework una vez que se crea el proyecto. El error lanzado le llegará al usuario a través del componente MessageGroup al que su propiedad showGlobalOnly le obliga a sacar únicamente los mensajes producidos por este método.

Figura 40. Respuesta del sistema frente a la validación errónea

5- Mediante el método setUsuario() le decimos a la sesión cual es el usuario que ha entrado en el sistema. A partir de aquí la página Logarse.jsp actualizará el value del componente:

```
<h:outputText value="Esta es la cuenta de #{SessionBean1.usuarioSesion}"/>
```

6- Por último observemos que el método hyperlinkEntrar_action() devuelve un String, es aquí cuando entran en juego las reglas de navegación descritas en el archivo de configuración **faces-config.xml** para la página Logarse.jsp**.** Más concretamente a través del caso “logado”:

```
<navigation-rule>
    <from-view-id>/Logarse.jsp</from-view-id>
    <navigation-case>
        <from-outcome>altausuario</from-outcome>
        <to-view-id>/altaUsuario.jsp</to-view-id>
    </navigation-case>
    <navigation-case>
        <from-outcome>logado</from-outcome>
        <to-view-id>/cuentaUsuario.jsp</to-view-id>
    </navigation-case>
    <navigation-case>
        <from-outcome>olvidoPass</from-outcome>
        <to-view-id>/olvidoContrasena.jsp</to-view-id>
    </navigation-case>
    <navigation-case>
        <from-outcome>page1</from-outcome>
        <to-view-id>/Page1.jsp</to-view-id>
    </navigation-case>
    <navigation-case>
        <from-outcome>index</from-outcome>
        <to-view-id>/Page1.jsp</to-view-id>
    </navigation-case>
    <navigation-case>
        <from-outcome>logarse</from-outcome>
        <to-view-id>/Logarse.jsp</to-view-id>
    </navigation-case>
    <navigation-case>
        <from-outcome>cuentaUsu</from-outcome>
        <to-view-id>/cuentaUsuario.jsp</to-view-id>
    </navigation-case>
    <navigation-case>
        <from-outcome>adminlogado</from-outcome>
        <to-view-id>/espacioAdministrador.jsp</to-view-id>
    </navigation-case>
    <navigation-case>
        <from-outcome>irInicio</from-outcome>
        <to-view-id>/Page1.jsp</to-view-id>
    </navigation-case>
    <navigation-case>
        <from-outcome>verCesta</from-outcome>
        <to-view-id>/cesta.jsp</to-view-id>
    </navigation-case>
</navigation-rule>
```

Figura 41. Reglas de navegación para la página Logarse.jsp

Obsérvese que en la página Logarse.jsp existen tantas opciones de navegación a otras páginas de la capa Presentación como posibles <navigation-case> hayan declarados. En el caso que estamos desarrollando la regla de navegación nos lleva a la página cuentaUsuario.jsp

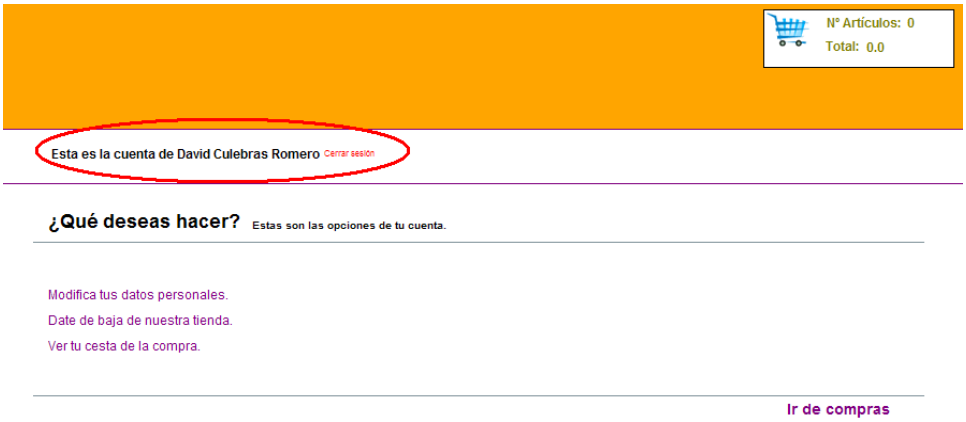

Figura 42. Respuesto del sistema frente a la validación correcta

8.4. El ámbito de sesión

Éste es el ámbito más utilizado. SessionBean1.java, es el controlador de la capa Aplicación que contiene todos y cada uno de los atributos a los que se accederá a través de cualquiera de las páginas jsp de la Presentación. Esto es posible gracias a los métodos get y set de los objetos del Dominio existentes en la clase. Veamos los atributos:

```
  //  OBJETOS DEL DOMINIO
private Categoria categoria;
private SubCategoria subC;
private Producto producto;
private Cesta cesta;
private Usuario usuario;

//    INSTACIAS DE LAS INTERFACES
private ICategoria iCategoria;
private IUsuario iUsuario;
private ICesta iCesta;
private IProducto iProducto;

//      ATRIBUTO DE LA FACTORIAREGISTRE
 private FactoriaRegistre factoriaRegistro;

//COLECCIONES A UTILIZAR POR LOS COMPONENTES DE PRESENTACIÓN
private ArrayList<Categoria> coleCategoria;
private ArrayList<SubCategoria> coleSubCategoria;
private ArrayList<Categoria> coleCategoriaParaMenu;
private ArrayList<SubCategoria> coleSubCategoriaParaMenu;
private ArrayList<Producto> listaTODOSProductos;
private ArrayList<Producto> listaProductos;
private ArrayList<Producto> listaProductosOFERTA;
private ArrayList<Usuario> listaTODOSUsuarios;
private ArrayList<Producto> listaProductosEncontrados;
private ArrayList<Cesta> coleCestas;
```

Figura 43. Atributos del controlador SessionBean1.java

Para poder utilizar las instancias de las interfaces del Dominio desde la SessionBean1, es necesario cargar los diferentes registros que las implementan. Por ésta razón, nada más iniciar la aplicación, desde el constructor del controlador SessionBean1.java se llama al siguiente método:

```
private void cargarRegistros() {
    System.setProperty("RegistroCategorias", "Persistencia.BBDDCategoria");
    System.setProperty("RegistroSubCategorias", "Persistencia.BBDDsubCategoria");
    System.setProperty("RegistroProductos", "Persistencia.BBDDProducto");
    System.setProperty("RegistroUsuarios", "Persistencia.BBDDUsuario");
    System.setProperty("RegistroCesta", "Persistencia.BBDDCesta");
    System.setProperty("RegistroLineaVenta", "Persistencia.BBDDLineaVenta");
}
```

Figura 44. Cargamos los registros

Una vez cargados los registros y continuando en el contructor, a través de la FactoriaRegistre obtenemos las instancias de las interfaces IUsuario, ICesta, IProducto, ICategotia.

```
this.iCategoria = factoriaRegistro.getInstance().getRegistroCategorias();
this.iProducto = factoriaRegistro.getInstance().getRegistroProductos();
this.iUsuario = factoriaRegistro.getInstance().getRegistroUsuario();
this.iCesta = factoriaRegistro.getInstance().getRegistroCesta();
```

A partir de aquí cada uno de los bean asociados a las paginas jsp, podrán llevar a cabo las funcionalidades gracias a los atributos que residen en la SessionBean1.java.

En el caso de uso Entrar al sistema, la clase Logarse.java, necesita de los métodos de la instancia de IUsuario para llevar a cabo la captura del usuario que ha introducido el mail y la contraseña.

8.5. Clase factoriaRegistre

Su objetivo es desacoplar la Aplicación de la Persistencia. A través de los métodos getRegistroX() se obtienen instancias de las interfaces del Dominio implementadas por los registros de la Persistencia. Esta clase es singleton ya que únicamente podemos tener un objeto en tiempo de ejecución.

```
public class FactoriaRegistre {
private static  FactoriaRegistre instance=null;


    private ICategoria iCategoria;
    private ISubCategoria isubCategoria;
    private IProducto iProducto;
    private IUsuario iUsuario;
    private ICesta iCesta;
    private ILineaVenta iLineaVenta;



    private FactoriaRegistre()throws Exception {
       iCategoria    = (ICategoria) Class.forName(System.getProperty("RegistroCategorias")).newInstance();
       isubCategoria = (ISubCategoria) Class.forName(System.getProperty("RegistroSubCategorias")).newInstance();
       iProducto=(IProducto)Class.forName(System.getProperty("RegistroProductos")).newInstance();
       iUsuario=(IUsuario)Class.forName(System.getProperty("RegistroUsuarios")).newInstance();
       iCesta=(ICesta)Class.forName(System.getProperty("RegistroCesta")).newInstance();
       iLineaVenta=(ILineaVenta)Class.forName(System.getProperty("RegistroLineaVenta")).newInstance();
    }
    public static synchronized FactoriaRegistre getInstance()throws Exception{
        if(instance==null) instance = new FactoriaRegistre();
        return instance;
    }
    public ICategoria getRegistroCategorias (){ return this.iCategoria;}
    public ISubCategoria getRegistroSubCategorias (){return this.isubCategoria;}
    public IProducto getRegistroProductos(){return this.iProducto;}
    public IUsuario getRegistroUsuario(){return this.iUsuario;}
    public ICesta getRegistroCesta(){return this.iCesta;}
    public ILineaVenta  getRegistroLineaVenta(){return this.iLineaVenta;}
}
```

Figura 45. Clase FactoriaRegistre.java

8.6. Validación de los formularios

Existen dos formas de validar formularios en JSF, la primera utilizando los validadores estándar que facilita la tecnología y la segunda crearte tus propias reglas de validación.

- **Validación estándar**

Como ejemplo de validación estándar tenemos el validador doubleRangeValidator el cual a sido utilizado para controlar la cantidad introducida por el usuario en el textField de la página cesta.jsp. La llamada al validador se produce en la propiedad validatorexpression del textField.

```
<webuijsf:textField binding="#{cesta.textFieldCantidad}" columns="1"
    converter="#{cesta.integerConverter1}" id="textFieldCantidad" label="" required="true"
    text="#{LV.cantidad}" validatorExpression="#{cesta.doubleRangeValidator.validate}"/>
```

La validación estándar consiste en declarar el archivo de propiedades ErroresBundle.properties del paquete controlErrores en el archivo de configuración de la aplicación **faces-config.xml**

```
<application>
    <message-bundle>controlErrores.ErroresBundle</message-bundle>
</application>
```

Tras la declaración, tenemos que redefinir los posibles mensajes de error que queremos hacerle llegar al usuario una vez se produzca un fallo en la validación.

```
javax.faces.component.UIInput.REQUIRED=Campo obligatorio
javax.faces.converter.BigIntegerConverter.BIGINTEGER=Introduce un número positivo.
javax.faces.converter.DoubleConverter.DOUBLE=Introduce un número positivo.
javax.faces.converter.IntegerConverter.INTEGER=Introduce un número positivo.
javax.faces.validator.DoubleRangeValidator.NOT_IN_RANGE=Introduce un número positivo del 1 al 10.
errorContrasena=La contraseña es errónea.
javax.faces.validator.DoubleRangeValidator.MINIMUM=El estoc ha de ser igual o mayor a 1.
java.lang.NumberFormatException=El precio a de ser un número mayor a 0 (99.9)
```

Figura 46. Archivo erroresBundle.properties

Esta forma de proceder sirve también para las conversiones de los campos en los formularios como podemos observar en el archivo.

- **Creación de reglas de validación propias**

En el caso de uso que estamos siguiendo de ejemplo, el campo a validar es el de la dirección de correo electrónico introducido.

```
<webuijsf:textField binding="#{Logarse.textFieldEmail}" id="textFieldEmail"
required="true" toolTip="Email."
validatorExpression="#{Logarse.control.validate}"/>
```

La propiedad del componente textField validatorExpression, hace referencia al atributo control de la clase Logarse.java, controlador asociado para la página Logarse.jsp, en la clase de la Aplicación, tendremos los get y set respectivos de este atributo.

La clase controlErrores.java del paquete controlErrores ha de implementar la interfaz Validator. Es obligatoria la implementación del método validate(). A este método le llegan el contenido y el nombre del componente donde se está realizando la validación. Se llevará a cabo la comprobación del contenido de cada campo a validar y en caso de que la validación sea errónea se lanzará una ValidatorExcepcion en la que se encapsulará el mensaje de error.

Veamos el contenido de la clase referente al caso de uso Entrar al sistema.

```
public class controlErrores implements Validator {

  public controlErrores() {
  }

  public void validate(FacesContext context, UIComponent component,
        Object value) throws ValidatorException {

    if (component.getId().equals("textFieldEmail")) {
      if (!this.validaMail((String) value)) {
       throw new ValidatorException(
        new FacesMessage("Comprueba la dirección de correo (qqq@qqq.com)."));
       }
    }
  }

  public boolean validaMail(String mail) {
    mail = mail.toLowerCase();
    Pattern p = Pattern.compile("^\\w+([\\.-]?\\w+)*@\\w+([\\.-]?\\w+)*(\\.\\w{2,3})+$");
    Matcher m = p.matcher(mail);
    if (m.find()) {
        return true;
    } else {
        return false;
    }
  }
}
```

Figura 47. Clase controlErrores.java

8.7. Implementación Persistencia

Cuando la Persistencia es requerida por parte de la Aplicación para llevar a cabo gestiones con los datos en la base de datos, lo hace a través de las interfaces del Dominio. Esto significa que cada clase de la Persistencia implementa dichas interfaces. La utilización de iBATIS en la gestión de la Persistencia, obliga a tener un archivo xml por cada una de las tablas de la base de datos. En estos archivos, están descritas las consultas a realizar a la base de datos. Veamos el contenido de uno de estos archivos.

```
<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE sqlMap PUBLIC "-//ibatis.apache.org//DTD SQL Map 2.0//EN"
    "http://ibatis.apache.org/dtd/sql-map-2.dtd">
<sqlMap namespace="Categoria">

    <typeAlias alias="categoria" type="Dominio.Categoria"/>

    <insert id="insertCategoria" parameterClass="categoria">
     insert into categoria(nombre)
        values(#nombre#)
    </insert>

    <select id="getListaCategorias" resultClass="categoria">
      select idCategoria,nombre
      from categoria
    </select>

    <select id="getCategoria" resultClass="categoria">
      select idCategoria,nombre
      from categoria
      where idCategoria=#idCategoria#
    </select>

    <update id="modificarCategoria" parameterClass="categoria">
    UPDATE categoria SET
           categoria.nombre=#nombre#
    WHERE
          categoria.idCategoria=#idCategoria#
    </update>

    <delete id="borrarCategoria" parameterClass="int">
    delete from categoria where idCategoria = #id#
    </delete>

</sqlMap>
```

Figura 48. Archivo Categoría.xml

La conexión a la base de datos también esta codificada en un archivo xml, en nuestro caso lo hemos llamado SqlMapConfig.xml. Contendrá además de los datos necesarios para la conexión, el mapeo de todos los archivos xml utilizados para las consultas. Todos los archivos xml referentes a la Persistencia de los datos, se encuentran en el paquete Persistencia.sql.

```
<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE sqlMapConfig
    PUBLIC "-//ibatis.apache.org//DTD SQL Map Config 2.0//EN"
    "http://ibatis.apache.org/dtd/sql-map-config-2.dtd">

<sqlMapConfig>

    <transactionManager type="JDBC" >
        <dataSource type="SIMPLE">
            <property value="com.mysql.jdbc.Driver" name="JDBC.Driver" />
            <property value="jdbc:mysql://localhost/bbdddomotechonline" name="JDBC.ConnectionURL" />
            <property value="root" name="JDBC.Username" />
            <property value="culebras" name="JDBC.Password" />
        </dataSource>
    </transactionManager>

    <sqlMap resource="Persistencia/sql/Categoria.xml" />
    <sqlMap resource="Persistencia/sql/Usuario.xml" />
    <sqlMap resource="Persistencia/sql/SubCategoria.xml" />
    <sqlMap resource="Persistencia/sql/Producto.xml" />
    <sqlMap resource="Persistencia/sql/Cesta.xml"/>
    <sqlMap resource="Persistencia/sql/LineaVenta.xml"/>

</sqlMapConfig>
```

Figura 49. Archivo SqlMapConfig.xml

Las llamadas a los archivos xml se llevarán a cabo desde las clases de la Persistencia. Volviendo al ejemplo del caso de uso Entrar al sistema, vamos a ver como se recoge un usuario del cual sabemos su mail y su contraseña.

```
public Usuario getUsuario(String email,String contrasena)throws Exception{
    Usuario usuario=new Usuario();
    usuario.setEmail(email);
    usuario.setContrasena(contrasena);

    usuario=(Usuario) sqlMapper.getInstancia().queryForObject("getUsuario", usuario);
    if(usuario!=null){
        usuario.setICesta(new BBDDCesta());
    }
    return usuario;
}
```

Como se ha comentado anteriormente, el atributo sqlMapper contiene todos los métodos de la librería iBATIS. La metodología siempre es la misma, se llamará a la consulta de cualquiera de los archivos xml mediante el primer parámetro, mientras que el tipo de objeto a mapear se envía a través del segundo parámetro. En este caso iBatis montará y devolverá el objeto del Dominio gracias a los get y set de los atributos pertenecientes al usuario que tiene el mail y la contraseña.

8.8. Algunos componentes JSF

La presentación de los datos, es de suma importancia ya que sin éstos, el usuario no podría interactuar con el sistema. Visual Web JSF te proporciona una serie de componentes visuales a través de su librería, pero existen otros muchos componentes pertenecientes a otras librerías las cuales podemos importar a nuestro proyecto. El principal problema, en términos de presentación, con el que nos hemos encontrado ha sido la ignorancia del comportamiento de los componentes visuales. En éste apartado veremos cual es el código de algunos de ellos.

4- DataList de la librería xmlns:t="http://myfaces.apache.org/tomahawk".

```
<t:dataList id="menuCategorias" value="#{SessionBean1.coleCategoriaParaMenu}" var="categoria">
    <webuijsf:hyperlink actionExpression="#{Page1.hyperlinkCategoriaSeleccionada1_action}"
        binding="#{Page1.hyperlinkCategoriaSeleccionada1}"
       id="hyperlinkCategoriaSeleccionada1" text="#{categoria.nombre}"/>
</t:dataList>
```

Figura 50. Componente dataList

Éste componente se comporta como el típico for, lo que hace es recorrer la colección existente en el bean SessionBean1.java llamada coleCategoriasParaMenu. Crea una variable a partir de la propiedad var por la que podremos acceder a los datos del objeto del dominio.

En este caso, recorremos todas las categorías de la colección creando un hyperlink en el que el texto del link será el nombre de la categoría que en ese momento se está visitando en la colección.

Tras aplicar el CSS correspondiente este es el resultado:

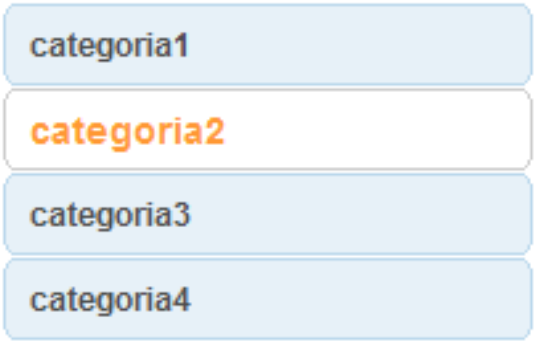

Figura 51. Menú categorias

2- Table de la librería xmlns:webuijsf="http://www.sun.com/webui/webuijsf".

Este componente pertenece al framework Visual Web JSF, utilizado a través de su paleta. Un table es el contnedor de tableRowGroup el cual alberga el conjunto de tableColummns y su comportamiento recuerda al de un for. Una de sus características es que permite paginación a través de las propiedades paginationButton y paginationControls. Éste componente se puede combinar con otros ya sean de ésta librería o de otras.

```
<webuijsf:table augmentTitle="false" binding="#{adminBajaCliente.tableClientes}"
id="tableClientes" paginateButton="true" paginationControls="true" title="Clientes">
    <webuijsf:tableRowGroup binding="#{adminBajaCliente.tableRowGroup1}"
    id="tableRowGroup1" rows="10" selected="#{adminBajaCliente.selectedState}"
    sourceVar="currentRow" sourceData="#{SessionBean1.listaTODOSUsuarios}">
        <webuijsf:tableColumn headerText="idUsuario" id="tableColumn1" sort="idUsuario">
            <webuijsf:staticText id="staticText1" text="#{currentRow.value['idUsuario']}"/>
        </webuijsf:tableColumn>

        <webuijsf:tableColumn headerText="nombre" id="tableColumn2" sort="nombre">
            <webuijsf:staticText id="staticText2" text="#{currentRow.value['nombre']}"/>
        </webuijsf:tableColumn>

        <webuijsf:tableColumn headerText="apellidos" id="tableColumn3" sort="apellidos">
            <webuijsf:staticText id="staticText3" text="#{currentRow.value['apellidos']}"/>
        </webuijsf:tableColumn>

        <webuijsf:tableColumn headerText="DNI" id="tableColumn4" sort="DNI">
            <webuijsf:staticText id="staticText4" text="#{currentRow.value['DNI']}"/>
        </webuijsf:tableColumn>

        <webuijsf:tableColumn headerText="email" id="tableColumn5" sort="email">
            <webuijsf:staticText id="staticText5" text="#{currentRow.value['email']}"/>
        </webuijsf:tableColumn>

        <webuijsf:tableColumn headerText="Selección" id="tableColumn6" selectId="radioButton1">
            <webuijsf:radioButton id="radioButton1" name="radioButton1"
            selected="#{adminBajaCliente.selected}"
            selectedValue="#{adminBajaCliente.selectedValue}"/>
        </webuijsf:tableColumn>

    </webuijsf:tableRowGroup>

</webuijsf:table>
```

Figura 52. Componente table

El componente tableRowGroup, entre otras cosas marca el número de filas de la tabla y la colección a iterar existente en el controlador SessionBean1.java así como la variable por la que se recorre la colección. A través de esta variable conseguimos los atributos

del objeto del Dominio existente en controlador La propiedad selected, hace referencia al método getSelectedState() del controlador asociado a la página jsp, utilizado para identificar la fila seleccionada marcada por el componente radioButton, que a su vez sus propiedades hacen referencia a los métodos getSelected() y getSelectedValue().

Los métodos por los que se consigue capturar la selección residen en la clase controladora asociada a la página jsp y los facilita el marco de trabajo.

```
/*********************************************************************
 *   REDEFINICION METODOS PARA LA SELECCION DE ELMENTOS EN LA TABLA
 *********************************************************************/
private TableSelectPhaseListener tablePhaseListener = new TableSelectPhaseListener();

public void setSelected(Object object) {
    RowKey rowKey = (RowKey) getValue("#{currentRow.tableRow}");
    if (rowKey != null) {
        tablePhaseListener.setSelected(rowKey, object);
    }
}

public Object getSelected() {
    RowKey rowKey = (RowKey) getValue("#{currentRow.tableRow}");
    return tablePhaseListener.getSelected(rowKey);

}

public Object getSelectedValue() {
    RowKey rowKey = (RowKey) getValue("#{currentRow.tableRow}");
    return (rowKey != null) ? rowKey.getRowId() : null;

}

public boolean getSelectedState() {
    RowKey rowKey = (RowKey) getValue("#{currentRow.tableRow}");
    return tablePhaseListener.isSelected(rowKey);
}
```

Figura 53. Redefinición de métodos para la selección de elementos

El resultado tras su utilización es el siguiente

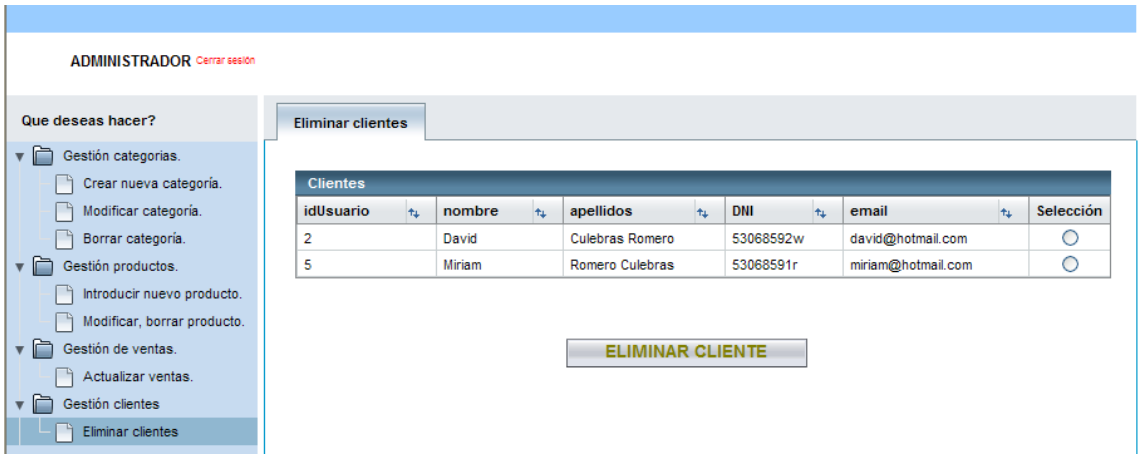

Figura 54. Resultados del componente table

4- DataTable de la librería xmlns:t="http://myfaces.apache.org/tomahawk".

Este componente se ha utilizado para mostrar el catalogo en la pagina principal de la aplicación.

```
<t:dataTable columnClasses="columna_abajoderechaImagen,columna_abajoderecha,columna_abajo"
     footerClass="standardTable_Header" headerClass="standardTable_Header"
     id="productosBusqueda" preserveDataModel="false" rows="5" styleClass="scrollerTable"
     value="#{SessionBean1.listaProductosEncontrados}" var="productosEncontrados">

    <h:column>
        <webuijsf:imageHyperlink actionExpression="#{Page1.imageHyperlinkImagenEncontrado_action}"
             binding="#{Page1.imageHyperlinkImagenEncontrado}" id="imageHyperlinkImagenEncontrado"
             imageURL="#{productosEncontrados.imagen}" toolTip="#{productoOfertado.nombre}"/>
    </h:column>

    <h:column>
        <webuijsf:label binding="#{Page1.labelIdProductoBusqueda}" id="labelIdProductoBusqueda"
            text="#{productosEncontrados.idProducto}" visible="false"/>
        <webuijsf:hyperlink actionExpression="#{Page1.hyperlinkNombreBuscado_action}"
            binding="#{Page1.hyperlinkNombreBuscado}" id="hyperlinkNombreBuscado"
            text="#{productosEncontrados.nombre}"/>

        <h:outputText value="#{productosEncontrados.descripcion}"/>
    </h:column>

    <h:column>
        <webuijsf:label binding="#{Page1.labelOfertaBuscado}" id="labelOfertaBuscado"
            text="Oferta!!" visible="#{productosEncontrados.oferta}"/>

        <h:outputText value="#{productosEncontrados.precio} €"/>

        <webuijsf:imageHyperlink actionExpression="#{Page1.imageHyperlinkImagenCarroBusqueda_action}"
             binding="#{Page1.imageHyperlinkImagenCarroBusqueda}" id="imageHyperlinkImagenCarroBusqueda"
             imageURL="/resources/cesta.gif" toolTip="Comprar #{productosEncontrados.nombre}"/>
    </h:column>

</t:dataTable>
```

Figura 55. Componente dataTable

Este componente cuenta una vez más con dos propiedades básicas, una para la colección a recorrer y otra para la variable en curso. También cuenta como en el caso anterior de columnas en la que se centrarán otros componentes.

La paginación se consigue gracias al componente dataScroll, de la misma librería, cuya propiedad for hace referencia a la tabla anterior.

```
<t:dataScroller fastStep="10" for="productosBusqueda" id="scroll_productosBusqueda1"
    pageCountVar="pageCount" pageIndexVar="pageIndex" paginator="true"
    paginatorActiveColumnStyle="font-weight:bold;" paginatorMaxPages="4"
    paginatorTableClass="paginator" styleClass="scroller">

    <f:facet name="first">
        <t:graphicImage border="1" url="/resources/primero.gif"/>
    </f:facet>
    <f:facet name="last">
        <t:graphicImage border="1" url="resources/ultimo.gif"/>
    </f:facet>
    <f:facet name="previous">
        <t:graphicImage border="1" url="resources/anterior.gif"/>
    </f:facet>
    <f:facet name="next">
        <t:graphicImage border="1" url="resources/siguiente.gif"/>
    </f:facet>

</t:dataScroller>
```

Figura 56. Paginación a partir del componente dataScroller

Nuevamente el resultado obtenido tras la aplicación de las hojas de estilo es el siguiente.

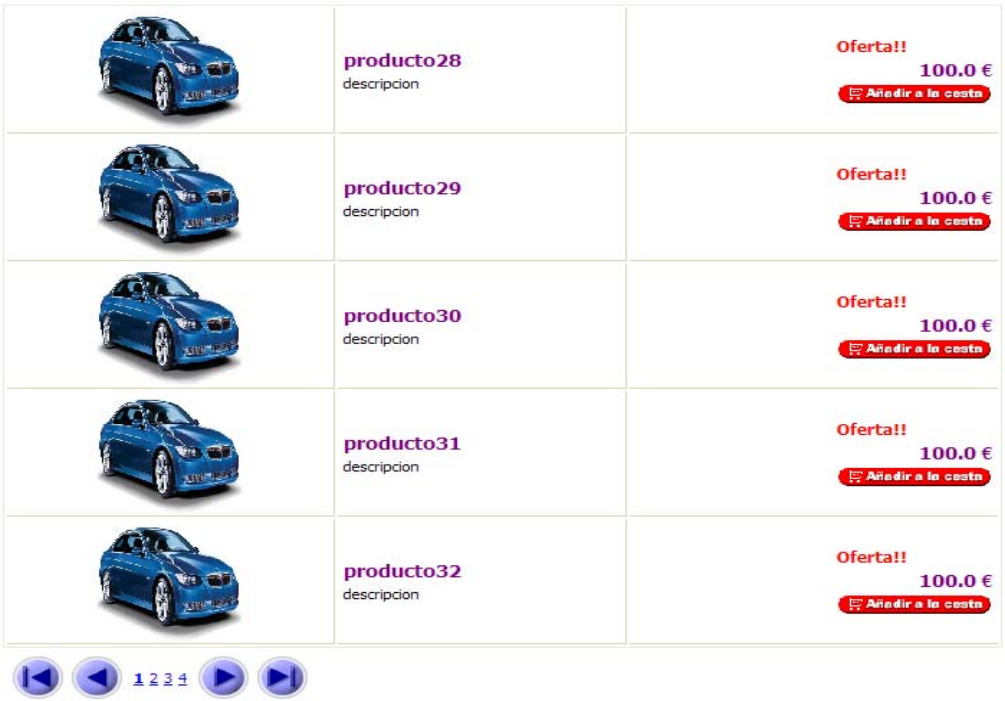

Figura 57. Resultados del componente table y de su paginación

4- DropDownList, librería xmlns:webuijsf="http://www.sun.com/webui/webuijsf".

```
<webuijsf:dropDown binding="#{modificarCategoria.dropDownCategorias}"
id="dropDownCategorias" items="#{SessionBean1.coleCategoria}"
valueChangeListenerExpression="#{modificarCategoria.dropDownCategorias_processValueChange}"/>
```

La propiedad ítems, es la encargada de recoger los valores de la colección, en este caso todas las categorías.

Por otro lado la propiedad valueChangeListenerExpression marca el destino de la acción cuando el usuario realiza la selección de algún ítem.

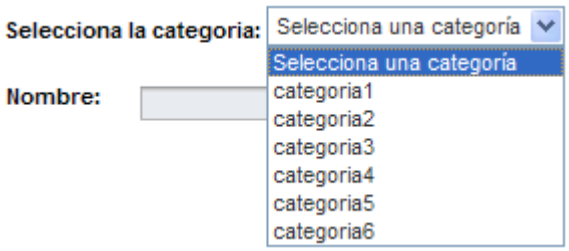

Figura 58. Resultado componente dropDown

9. Manual de usuario

• Rol cliente e invitado

La página principal de la aplicación mostrará el catalogo de productos que actualmente se encuentran a la venta. En primera instancia mostrará todos los productos que están de oferta. La clasificación de los productos en el catalogo es a partir de las diferentes categorías y subcategorías, en la página principal se puede acceder al menú de categorías.

En ésta primera página podemos observar la cesta que al principio estará vacía. Se podrá utilizar el buscador, acceder a los productos ofertados en todo momento y el cliente registrado podrá entrar en su cuenta a través de la opción “Mi cuenta” es aquí donde el usuario invitado podrá darse de alta en nuestra tienda.

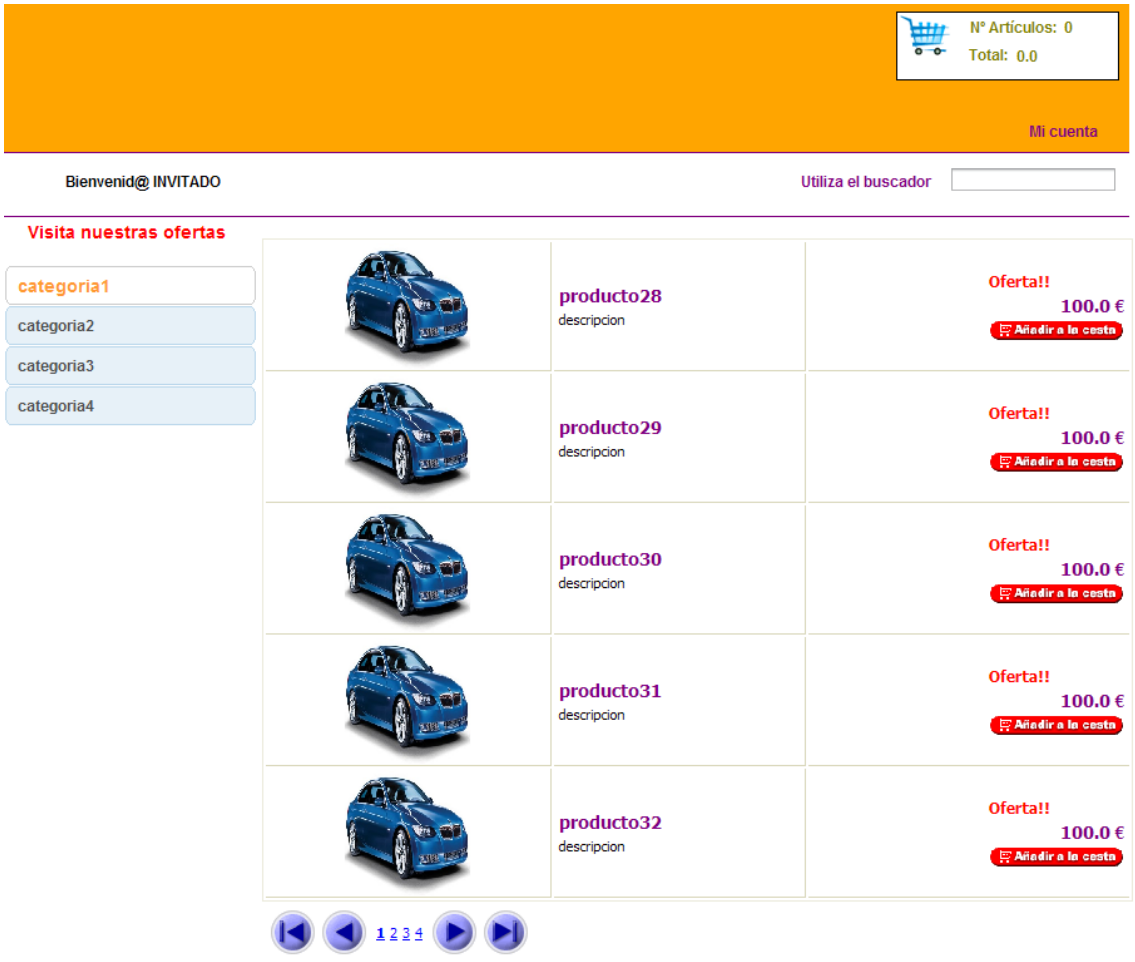

Figura 59. Página principal

Cada una de las categorías que aparecen en el menú, tienen subcategorías las cuales aparecen cuando pulsamos en una determinada categoría.

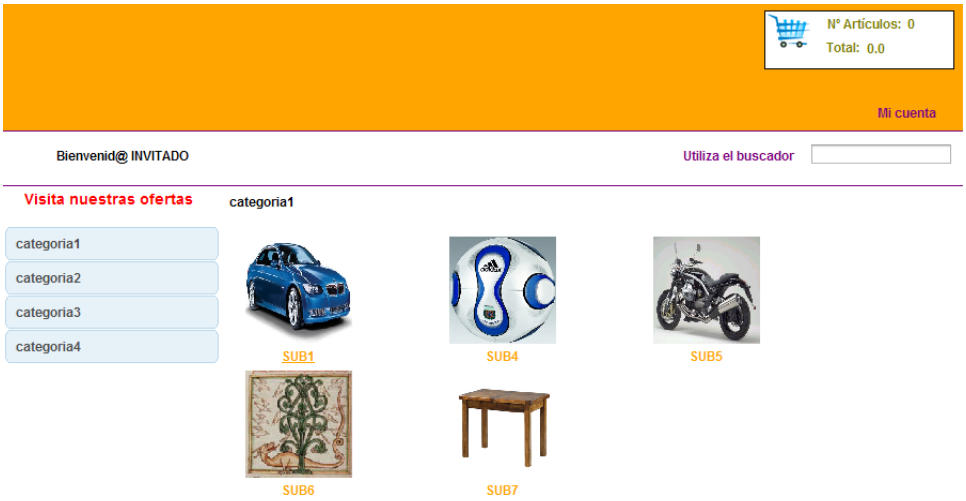

Figura 60. Menú subcategorias

Cada una de las subcategorías que salen en el catalogo cuentan con productos, si escogemos cualquiera de estas subcategorías aparecerán todos los productos que existen a la venta ordenados y catalogados mediante el paginador.

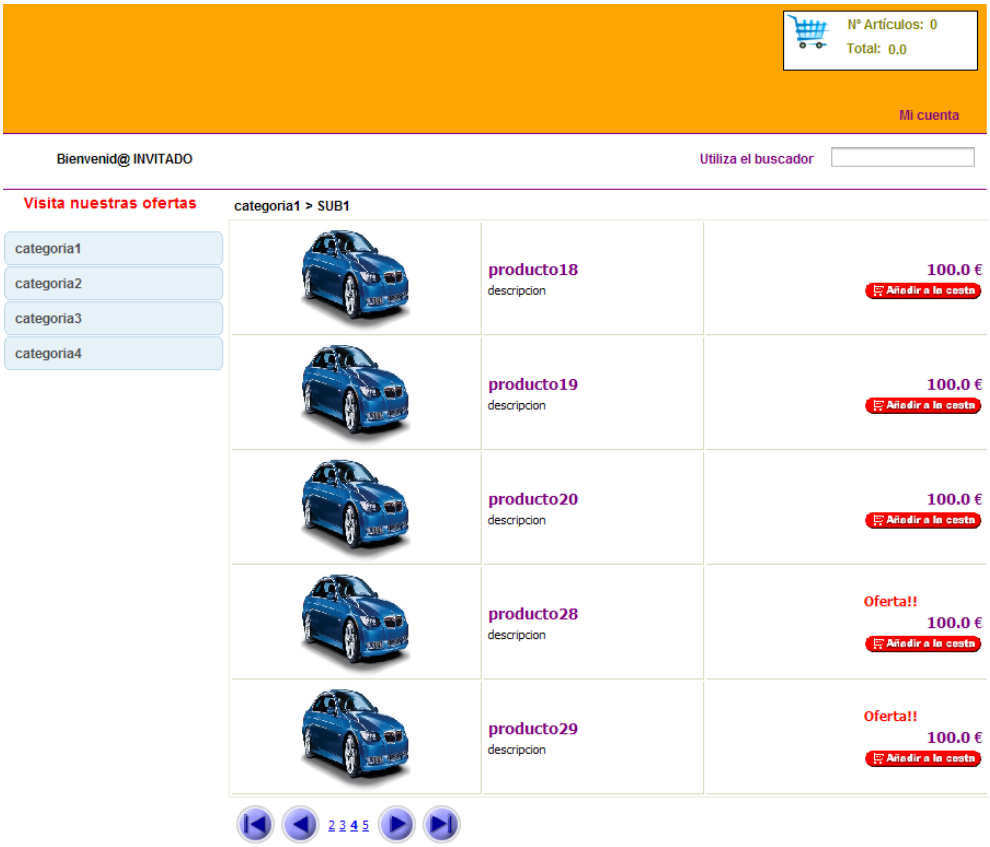

Figura 61. Productos pertenecientes a una subcategoría

Una vez el usuario añade un determinado producto a la cesta, ésta aparecerá en pantalla dejándole modificar la cantidad que desea comprar. La mini cesta estará constantemente visible y a través de ella el usuario podrá visitar la cesta siempre que lo desee. La cesta también informará al usuario de los posibles errores de validación.

Figura 62. Cesta

En el momento en que el usuario decide tramitar el pedido, se le obligará a logarse. En caso de que no sea cliente en nuestra tienda, se tendrá que registrar.

Figura 63. Entrar al sistema

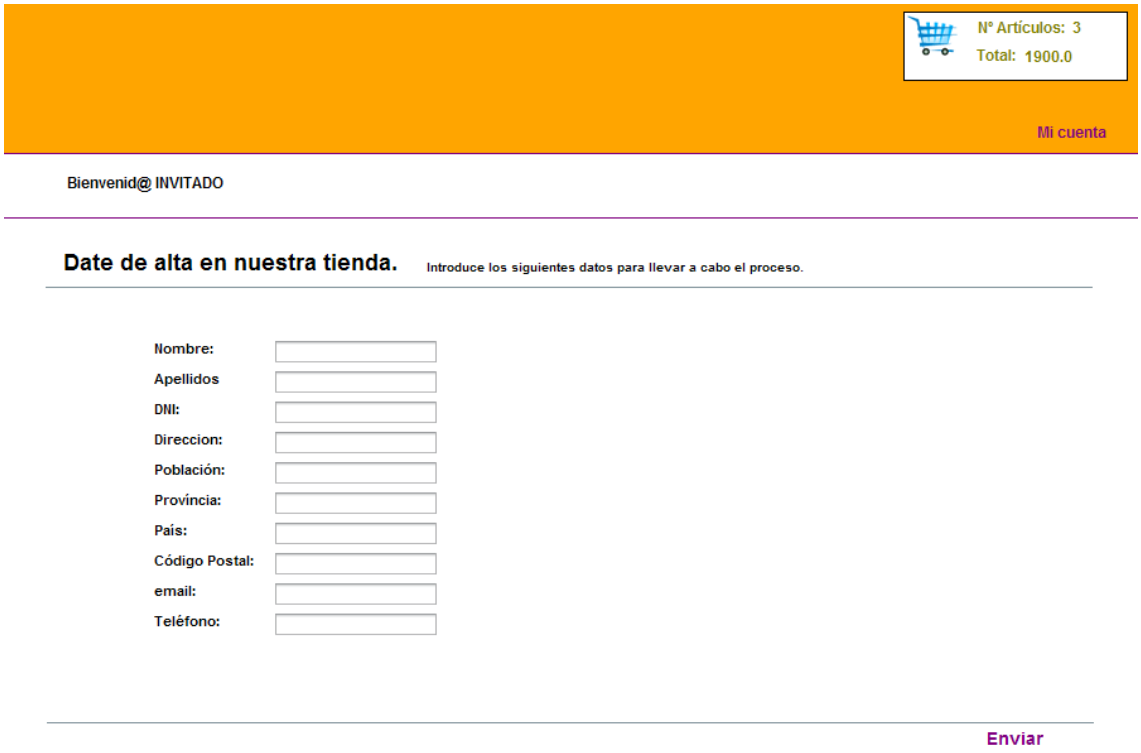

Figura 64. Alta usuario

Una vez logado, el cliente tendrá acceso a su cuenta de usuario don podrá ver la cesta que actualmente tiene en sesión, modificar sus datos personales, darse de baja de nuestra tienda seguir comprando o cerrar la sesión.

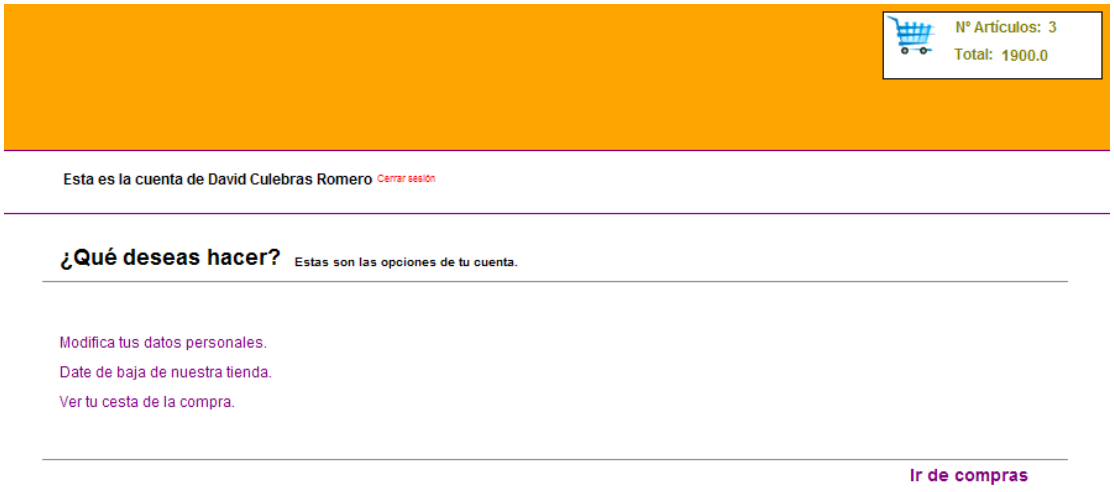

Figura 65. Cuenta de usuario

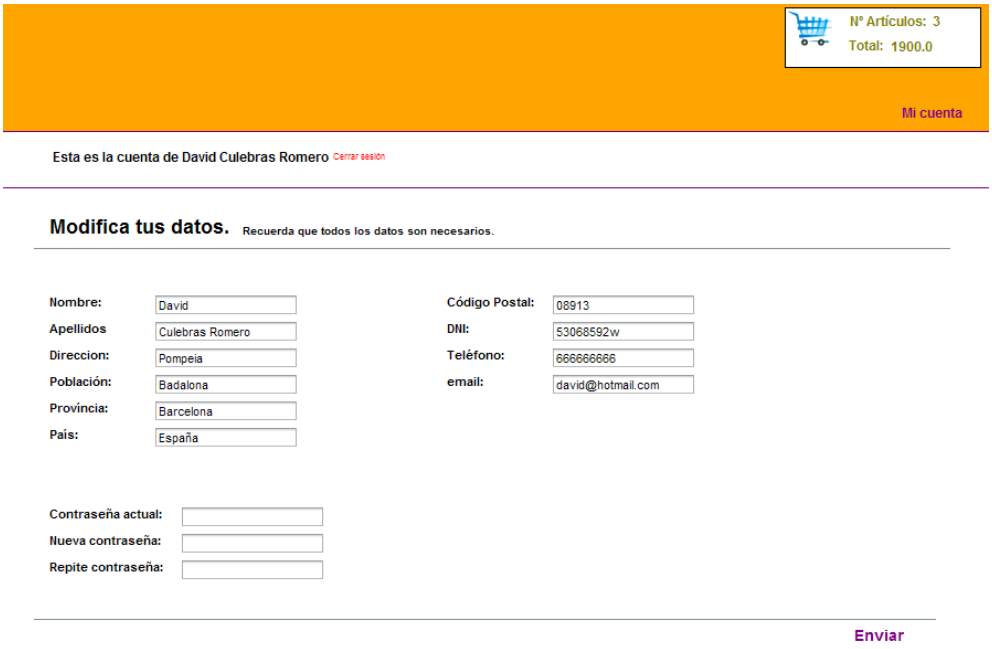

Figura 66. Modificación de usuario

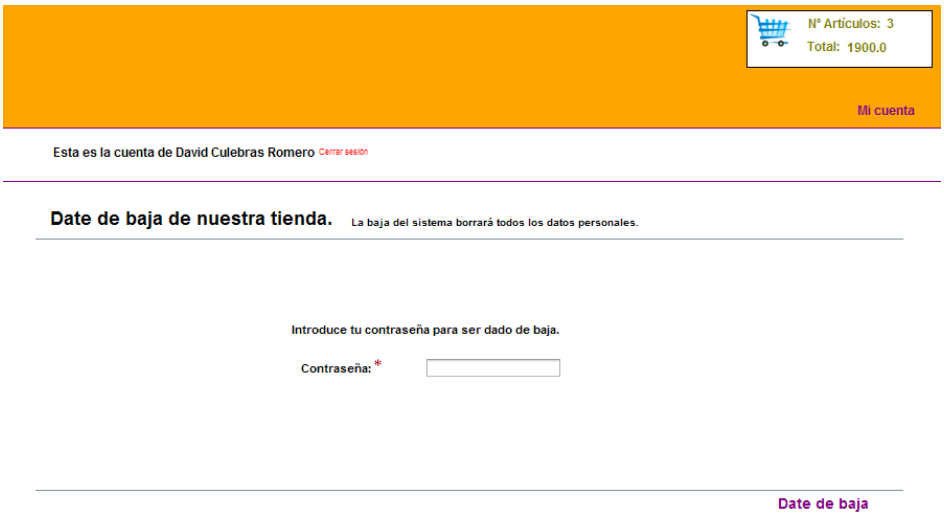

Figura 67. Borrar usuario

Por último una vez es identificado por el sistema, se le permitirá llevar a cabo la compra a través de la última página en la que aparecerá la opción de modificación de datos personales, actualizar cantidades de los productos, borrar productos y cancelar pedido. El sistema llevará a cabo el control de stock de cada producto informando al cliente de cual es el número de existencias en el almacén. En el momento en que se acepta la compra, el sistema dirige al cliente a al web de PayPal donde se llevará a cabo el pago del pedido.

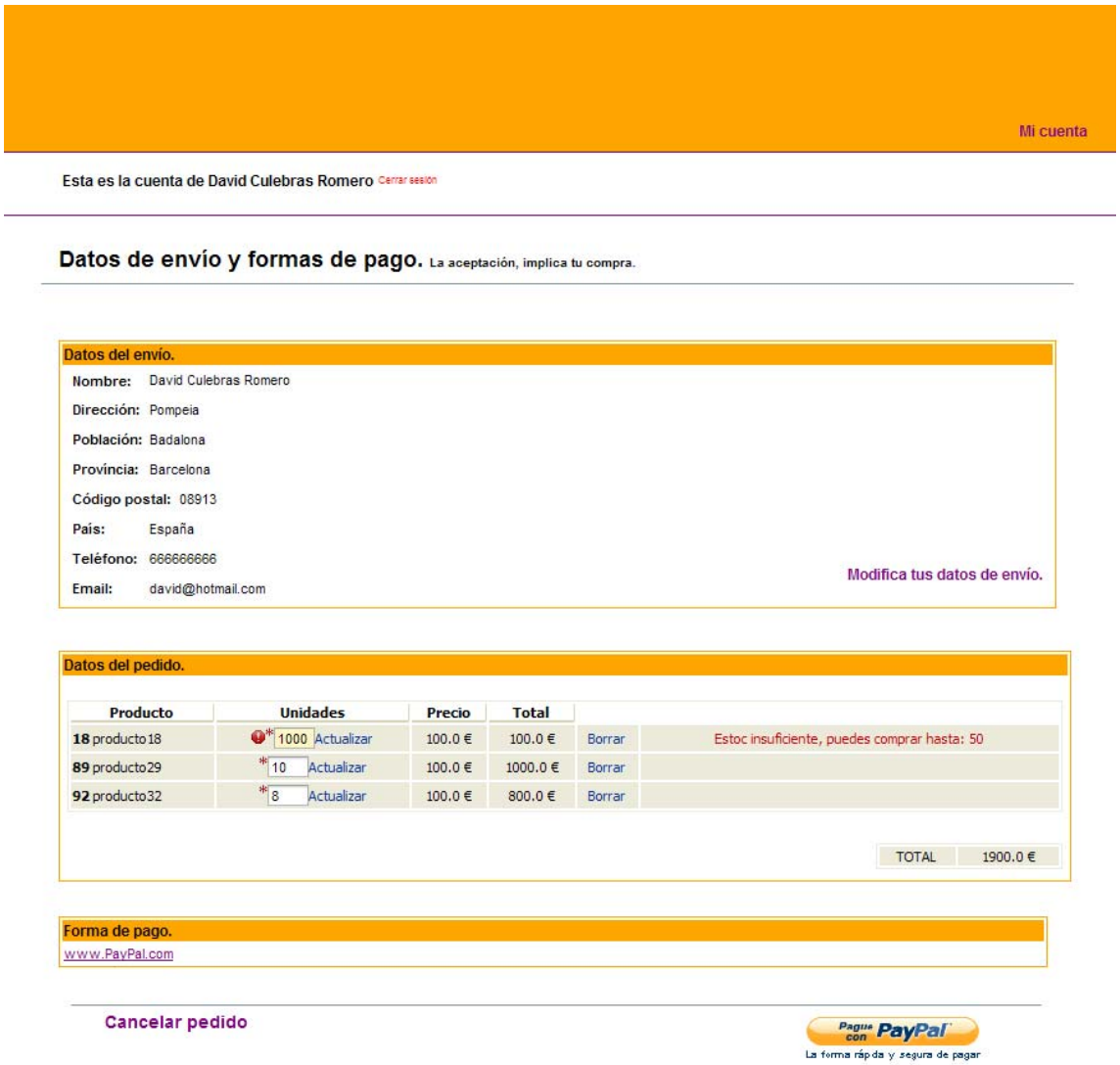

Figura 68. Hacer pedido

- **Rol administrador**

Como veremos a continuación el rol de administrador en la aplicación está diferenciado estéticamente por los colores que adopta la cuenta de administrador. En su cuenta, el administrador podrá llevar a cabo funcionalidades tales como gestionar el catalogo dando de alta nuevos productos, nuevas categorías así como borrándolos del catalogo. Además podrá gestionar las ventas y eliminar a clientes. En la página, todas estas funcionalidades aparecerán en el menú situado a la izquierda (Véase figura 69).

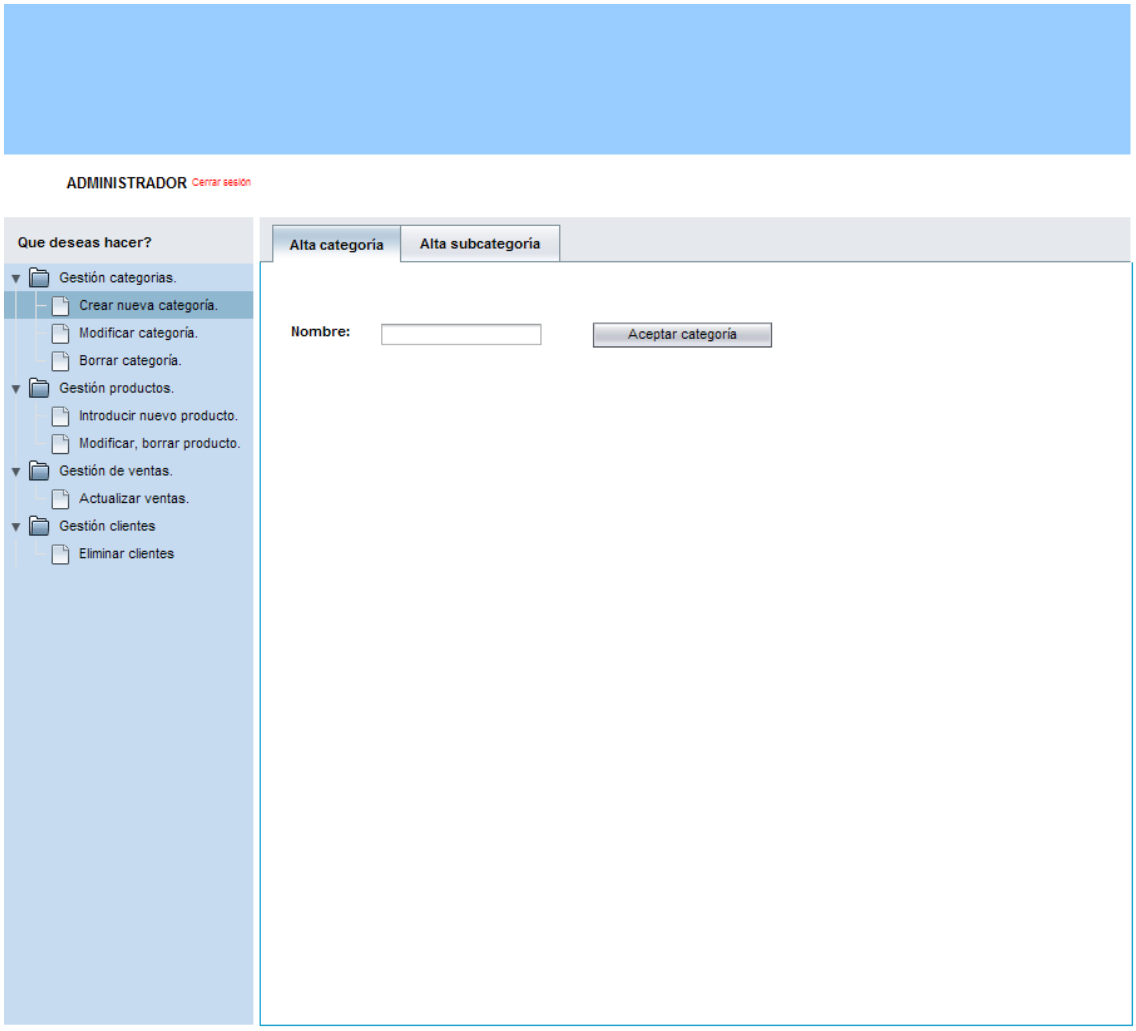

Figura 69. Cuenta administrador

10. Conclusiones

La elaboración del proyecto ha sido entretenida y muy laboriosa, le he tenido que dedicar mucho tiempo, sobretodo hasta pasar la curva de aprendizaje de la nueva tecnología. Antes de plantearme si quiera el proyecto, no había oído hablar de términos como iBATIS ni había entrado nunca a estudiar una tecnología relacionada con las aplicaciones Web. Puedo decir que se han cumplido los objetivos marcados en primera instancia, sobre todo el hecho de haber aprendido la tecnología JavaServer Faces.

En cuanto a experiencia adquirida, estoy muy satisfecho porque el proyecto me ha dado la oportunidad de aplicar conocimientos que se han dado durante la carrera. He aprendido a utilizar herramientas que anteriormente no había utilizado nunca. La puesta en marcha de este proyecto también me ha servido para entender y adaptar numerosas herramientas realizadas por terceros.

Los dos principales marcos de trabajo utilizados Visual Web JSF e iBATIS se han aprendido lo suficiente como para llevar acabo las funcionalidades típicas de una tienda virtual, cumpliendo así con otro de los objetivos marcados en un principio.

Para terminar comentar que me siento satisfecho con el trabajo realizado.

11. Posibles ampliaciones

El único aspecto quizás que se me haya pasado un poco por alto ha sido la aplicación de técnicas para incrementar la estética de la página. Me hubiera gustado poder aprender algo más de maquetación Web. Una posible propuesta de futuro sería lograr un aspecto atractivo, accesible y usable de la tienda virtual.

En cuanto a funcionalidades, el proyecto cumple con las funciones básicas de una tienda virtual, si es cierto que quizás para una tienda modesta se de por finalizado pero hay muchos aspectos que no se han llevado a cabo como por ejemplo políticas de descuentos.

12. Glosario

Framework: En el desarrollo de software, es una estructura de soporte definida mediante la cual otro proyecto de software puede ser organizado y desarrollado.

IDE, entorno de desarrollo integrado: Entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica GUI

JSF, JavaServer Faces: JavaServer Faces (JSF): Framework para aplicaciones Java basadas en web que simplifica el desarrollo de interfaces de usuario en aplicaciones Java EE.

Visual Web JSF: Nuevo Framework de Java mediante el cual se pueden generar páginas Web visualmente.

iBatis: framework de código abierto basado en capas desarrollado por Apache Software Foundation, que se ocupa de la capa de Persistencia, se sitúa entre la lógica de Negocio y la capa de la Base de Datos.

MySQL: sistema de gestión de base de datos (SGBD) relacional, multihilo y multiusuario.

JavaBeans: son un modelo de componentes creado por Sun Microsystems para la construcción de aplicaciones en Java.

MVC, Modelo Vista Controlador: es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos.

DAO, Data Access Object: es un componente de software que suministra una interfaz común entre la aplicación y uno o más dispositivos de almacenamiento de datos, tales como una Base de datos o un archivo

12. Bibliografía

[1] http://www.microsoft.com/spanish/msdn/netframework/framework20_InformacionCaract.mspx (Enero 2009)

[2] http://java.sun.com/javaee/ (Enero2009)

[3] http://www.consultec.es/DocInformes/j2ee%20VS%20NET.pdf (Enero 2009)

[4] http://www.sicuma.uma.es/sicuma/Formacion/documentacion/JSF.pdf (Enero 2009)

[5] http://es.wikipedia.org/wiki/Modelo_Vista_Controlador (Enero 2009)

[6] http://www.netbeans.org (Enero 2009)

[7] http://ibatis.apache.org/javadownloads.cgi (Enero 2009)

[8] http://www.netbeans.org/kb/60/web/helloweb.html (Enero 2009)

[9] http://wiki.netbeans.org/NavegandoPaginasVisualJSF (Enero 2009)

[10] http://www.netbeans.org/kb/60/web/intro.html (Enero 2009)

[11] http://www.netbeans.org/kb/docs/web/fileupload.html (Enero 2009)

[12] http://ibatis.apache.org/javadownloads.html (Enero 2009)

[13] http://dev.mysql.com/downloads/mysql/5.1.html#downloads (Enero 2009)

[14] Craig Larman, Uml y Patrones. Una introducción al análisis y diseño orientado y al proceso unificado. Prentice Hall, segunda edición 2003.

[15] Dossier Proyectos UPC , Otoño / Primavera 2007,2008

[16] http://www.programacion.com/java/tutorial/javamail Enero 2009

14. Anexo I

En primer lugar, para enviar emails es necesario un servidor SMTP (Single Mail Transmision Protocol). Normalmente, nuestro administrador de redes nos da la dirección de este servidor,si no es así, lo podemos bajar fácilmente de Internet. Otra opción es obtener una cuenta de un servidor externo i conectarnos para enviar emails.

Esta es la forma que hemos utilizado en este proyecto, ya que al no tener un dominio registrado, al enviar correos desde un servidor instalado para nosotros haría que otros servidores lo detectaran como correo spam.

Para la realización de este proyecto hemos utilizado una cuenta de gmail pero se podría cambiar por otra cuenta o instalar un servidor SMTP registrando el dominio y enviarlo desde aquí.

El siguiente paso a realizar es descargar e instalar el API de JavaMail, que contiene el conjunto de clases que nos permitirán conectar con el servidor SMTP y enviar emails.

También necesitamos la ultima versión de JAF Activation. Al ser una librería nativa, podemos encontrarla en la pagina de Sun.

Para descargar el API JavaMail:
http://java.sun.com/products/javamail/downloads/index.html

Para descargar JAF Activation,
http://java.sun.com/products/javabeans/glasgow/jaf.html

Los archivos .zip obtenidos de las descargas anteriores contienen mucha documentación y los archivos .jar que necesitamos. Tenemos que sacar los archivos .jar y modificar nuestro CLASSPATH para incluirlo. Todos los IDE permiten añadir referencias a los archivos .jar y es el propio IDE el que se encarga de generar la variable CLASSPATH correcta en el momento de compilar. Una vez realizado todo esto, ya podemos utilizar las funciones y las clases de las librerías que hemos añadido en nuestro proyecto, si existe alguna duda, siempre nos podemos mirar el API.

15. Anexo II (Contenidos de CD-ROM)

1. Carpeta con la documentación.
    - Primeras páginas e índices.
    - Documentación del proyecto.

2. Carpeta con los Scripts necesarios para la puesta en marcha de la BBDD en MySQL.
    - CREACION_TABLAS_BBDD.txt (Archivo de creación de las tablas.)
    - INSERCIONES.txt (Archivo de inserciones en las tablas.)

3. Carpeta con la aplicación.
    - Código fuente.
    - Librerías.

4. Software necesario.
    - Netbeans 6.1
    - MySql Server 6.0
    - MySql Administrador.
    - MySQL Workbench 5.0 OSS