UNIVERSIDAD
DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR
DE INGENIERÍA INFORMÁTICA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

INGENIERIA TECNICA INFORMATICA DE SISTEMAS

APLICACIÓN WEB DE UNA EMPRESA DE PRODUCTOS HORTOFRUTÍCOLAS

Realizado por

Juan José González Ruiz

Dirigido por

Dr. José Galindo Gómez

Departamento

Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA

UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

Ingeniería Técnica Informática (Sistemas)

Reunido el tribunal examinador en el día de la fecha, constituido por:

Presidente/a Dº/Dª. _____

Secretario/a Dº/Dª._____

Vocal Dº/Dª. _____

para juzgar el proyecto Fin de Carrera titulado:

"**Aplicación Web de una empresa de productos hortofrutícolas"**

del alumno/a Dº/Dª. Juan José González Ruiz

dirigido por Dº/Dª. José Galindo Gómez

ACORDÓ POR _____OTORGAR LA CALIFICACIÓN DE _____

Y PARA QUE CONSTE, SE EXTIENDE FIRMADA POR LOS COMPARECIENTES DEL TRIBUNAL, LA PRESENTE DILIGENCIA.

Málaga, a de del 200_

El/La Presidente/a El/La Secretario/a El/La Vocal

Fdo: Fdo: Fdo:

ÍNDICE

Agradecimientos

Con este proyecto pongo fin a una parte de mi vida en la universidad. Por fin he realizado uno de mis sueños que veía tan lejano y difícil, y aun así no me lo creo.

Quiero dar un agradecimiento:

- A mis padres, hermano y mi novia por su esfuerzo, y su dedicación a que terminara esta carrera.

- A mis amigos y compañeros sobre todo a Oscar que fue el que me recomendó que realizara el proyecto con estas tecnologías.

- A mi tutor D. José Galindo por su dedicación y confianza en mi proyecto.

- A D. Sergio Gálvez por permitirme usar su laboratorio de la facultad.

Mis más sinceras gracias a todos.

CAPITULO 1:

Introducción

En este capítulo, voy a dar una breve introducción del funcionamiento y la tecnología utilizada en esta aplicación, así como una descripción de todos los capítulos del que consta el proyecto.

1.1 Funcionamiento y tecnología

El desarrollo de este proyecto era la implementación de un servicio web, que realizara la función de comercio electrónico aplicado a la comercialización de productos hortofrutícolas.
Para ello nos hemos decantado por una tecnología web basado en Java, en concreto J2EE (Java 2 Enterprise Edition), la cual nos ha permito poder utilizar todas las ventajas que posee el lenguaje de programación Java.

Dentro de los dos sectores fuertes que hoy en día implementa la tecnología J2EE, el primero basado en EJB (Enterprise Java Bean) desarrollado por Sun y el segundo basado JavaBean, hemos preferido el segundo, ya que nos permite un mayor control sobre el desarrollo del software, así como estructuras más simples y más manejables.

La aplicación ofrecerá productos hortofrutícolas a través de la web, donde los usuarios podrán registrarse y realizar pedidos, así como el envío de los productos a su casa, o la recogida del producto en el sitio.

Hemos desarrollado esta aplicación porque hemos considerado que es un producto que no se ofrece por la web, aunque nuestra aplicación, con la tecnología utilizada se puede modificar o ampliar para ofrecer otros productos diferentes o parecidos.
Este sistema de información lo hemos analizado utilizando la metodología Métrica 3, y desarrollado utilizado la programación orientas a objetos y las tecnologías basada en J2EE.

1.2 Estructura de la memoria

En este apartado describimos los diferentes capítulos del que consta nuestra memoria.

- Capitulo 2. Herramientas
  En este capitulo describimos las herramientas utilizada para el desarrollo del sistema de información tanto los framework como el lenguaje de programación o el software utilizado para el desarrollo del sistema de información.
  Los diferentes software y frameworks lo detallamos a continuación.

    - Hibernate: mapeador de base de datos relacional. Nos permite acceder a la base de datos relacional y mapear para crear objetos que nos permitirán utilizarlos en nuestro sistema de información.

    - Spring: Framework de desarrollo basado en Java, implementado en nuestro sistema de información en la capa de la lógica de negocio.

    - JSF: Framework de desarrollo basado en Java, implementado en nuestro sistema de información en la capa de presentación.

    - Otras herramientas: Otros programas como PostgreSQL para la gestión de los datos, MagicDraw y StartUML para el diseño UML y Tomcat y Apache como servidor web de nuestra aplicación.

- Capitulo 3.Aplicación de la Métrica 3.
  Este capítulo es el encargado de toda la metodología empleada para el análisis, diseño y desarrollo del sistema de información. El capítulo consta de cuatro apartados generales, que son:

    - Estudio de Viabilidad del Sistema (EVS). Se encarga del análisis concreto de las necesidades del cliente para proponer una solución a corto plazo teniendo en cuenta varias restricciones, ya sean económicas, legales y operativas.

- Análisis del Sistema de Información (ASI). En este proceso se obtiene una especificación detallada del sistema de información que satisfaga las necesidades de información de los usuarios y sirva de base para el posterior diseño del sistema.

- Diseño del Sistema de Información (DSI). Se define la arquitectura del sistema, así como el entorno tecnológico que le va a dar soporte junto con una especificación detallada de los componentes del sistema de información. En este proceso, también, se generan todas las especificaciones de construcción necesarias, así como una descripción técnica del plan de pruebas, definición de requisitos de implantación y el diseño de procedimientos de migración y carga de inicial de datos si es necesario.

- Construcción del Sistema de Información (CSI). Se genera el código necesario de todos los componentes del sistema de información, junto con los procedimientos de operación y seguridad, así como el desarrollo de las pruebas necesarias.

- Capitulo 4. Manual de usuario.

  En este apartado mostramos el manual de instalación y manejo del programa.

  Se ha realizado un manual de usuario lo más simple posible de tal manera que sea fácil manejar la aplicación.

  El manual de instalación es un poco más complejo de entender pero siguiendo los pasos y descargando los programas necesarios se puede realizar la instalación sin problemas.

- Conclusiones y líneas futuras.

  Aquí se comenta todo lo aprendido durante el desarrollo del proyecto, así como las conclusiones al finalizar la realización de éste. En este apartado también se detallan algunos aspectos ampliables a la aplicación desarrollada.

- Referencias.

  Aquí se hace un desglose tanto de la bibliografía utilizada como páginas de Internet y artículos relacionados con los temas utilizados en el desarrollo del proyecto, ya sean temas informáticos o especializados en la materia de la aplicación.

- Apéndice A. Diagramas.

  Diagramas desarrollados durante la aplicación de la metodología Métrica 3.En este apartado se especifican de forma breve y se muestran imágenes de los diferentes diagramas utilizados para el análisis y desarrollo de la aplicación. Estos diagramas se explican a lo largo del desarrollo de la metodología Métrica 3 en el capítulo tres.

- Apéndice B. Glosario.

  Información sobre todos los términos de las diferentes tecnologías utilizadas en nuestra aplicación.

Capítulo 2:
Herramientas
(*Hibernate*, *Spring*, *JSF*)

2.1 Ideas Generales

Para el desarrollo de esta aplicación Web, hemos utilizado una serie de herramientas que nos han permitido la división de nuestra aplicación, en distintas capas, dicha capas las podemos clasificar en:

- **Capa de Persistencia**.

  La capa de Persistencia maneja el almacenamiento de los datos de sus entidades de negocio en una base de datos. Maneja también la creación de entidades de dominio (Cliente, Pedido, etc.) leyendo los datos de una base de datos y construyendo la entidad de dominio. En otras palabras, es un constructor de sus entidades de dominio. La herramienta que utilizaríamos en este capa seria *Hibernate*, ya que es un buen mapeador ORM, el cual no reduce el coste de desarrollo de esta capa y su comunicación entre el modelo Entidad-Relación de la Base de Datos y el modelo de objeto de nuestra aplicación hecha en Java.

- **Capa de negocio.**

  Se denomina capa de negocio (e incluso de lógica del negocio) porque es aquí donde se establecen todas las reglas que deben cumplirse en nuestra aplicación. Esta capa se comunica con la capa de presentación, para recibir las solicitudes y presentar los resultados, y con la capa de persistencia, para solicitar al gestor de Base de Datos como almacenar o recuperar datos. En esta capa utilizaremos *Spring*, ya que es un *Framework* bastante útil para el diseño de esta capa, debido a su gran funcionalidad a la hora de comunicarse con la capa de persistencia (*Hibernate*) y la capa de presentación (*JSF).*

- **Capa de presentación.**

  Es la que ve el usuario (también se la denomina "capa de usuario"), presenta el sistema al usuario, le comunica la información y captura la información del usuario en un mínimo de proceso (realiza un filtrado previo para comprobar que no hay errores de formato). Esta capa se comunica únicamente con la capa de negocio. También es conocida como interfaz grafica y debe tener la característica de ser "amigable" (entendible y fácil de usar) para el usuario. Hemos utilizado el *framework JSF* para esta capa, ya que es un *framework* bastante robusto en la capa de presentación, y se interconecta bastante bien con las demas capas de nuestra aplicación.

2.2 Capa de persistencia (*Hibernate*)

Hibernate es una ORM de libre distribución, de las más maduras y completas. Su uso está muy extendido y además está siendo desarrollada de forma muy activa. Una característica muy importante que distingue *Hibernate* de otras soluciones al problema de la persistencia, es que la clase *Hibernate* persistente puede utilizarse en cualquier contexto de ejecución, es decir, no se necesita un contenedor especial para ello [2][3].

Esto nos va a permitir que mediante *Hibernate* guardemos y recuperemos los objetos de la Base de Datos, ya que el hace la conversión de las atributos de dichos objetos a los atributos de las tablas de la base de datos mediante archivos XML.

Hibernate busca solucionar el problema de la diferencia entre los dos modelos usados hoy en día para organizar y manipular datos, El usado en los lenguajes de programación (orientación a objetos) y el usado en las bases de datos (modelo relacional). Para lograr esto permite al desarrollador detallar cómo es su modelo de datos, qué relaciones existen y qué forma tienen.

2.2.1 Características de Hibernate

Entre las características de *Hibernate* las más destacadas son [13]:

- Permite a la aplicación manipular los datos de la base operando sobre objetos, con todas las características de la POO (Programación Orientada a Objetos).

- Convierte los datos entre los tipos utilizados por Java y los definidos por SQL.

- Genera las sentencias SQL y libera al desarrollador del manejo manual de los datos que resultan de la ejecución de dichas sentencias, manteniendo la portabilidad entre todas las bases de datos con un ligero incremento en el tiempo de ejecución.

- Es flexible en cuanto al esquema de tablas utilizado, para poder adaptarse a su uso sobre una base de datos ya existente. También tiene la funcionalidad de crear la base de datos a partir de la información disponible.

- Ofrece también un lenguaje de consulta de datos llamado **HQL** (*Hibernate Query Language*), dicho lenguajes de consulta de datos permite enviarle sentencias de consulta, recuperando atributos de la columnas convertidos en objeto, que serán con los que trabajemos en nuestra lógica de negocio.

2.2.2 Arquitectura

La Figura 2.1 muestra los roles de las interfaces *Hibernate* más importantes en las capas de persistencia y de negocio de una aplicación J2EE. La capa de negocio está situada sobre la capa de persistencia, ya que la capa de negocio actúa como un cliente de la capa de persistencia [13].

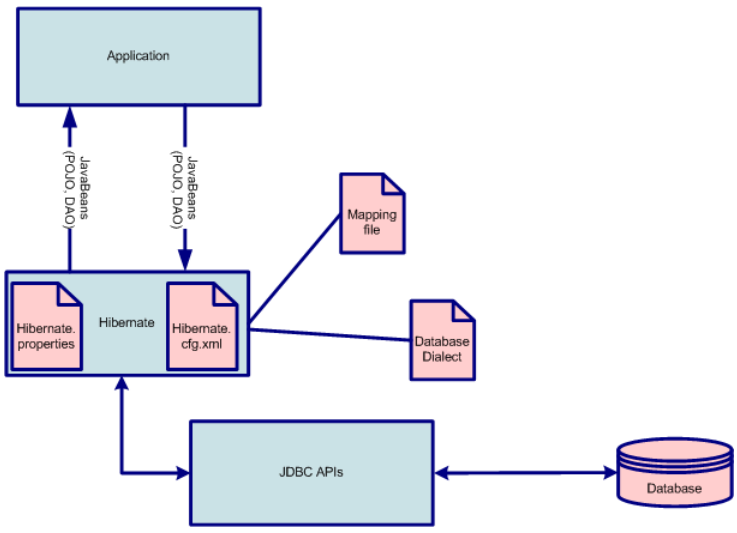


Figura 2.1 Vista de alto nivel del API de *Hibernate* en una arquitectura por capas

Hibernate consta de una serie de interfaz que permita la conexión y configuración entre la Base de Datos y nuestra Lógica de Negocio, las interfaces pueden clasificarse como sigue:

- *Configuration* interfaces llamada por el código de la infraestructura de la aplicación para configurar *Hibernate.*

- *Callback* interfaces que permiten a la aplicación reaccionar ante determinados eventos que ocurren dentro de la aplicación, tales como Interceptor, Lifecycle, y Validatable.

- *UserType, CompositeUserType, e IdentifierGenerator* interfaces que permiten extender las funcionalidades del mapeado de *Hibernate*.

- La interfaz *SessionFactory* permite obtener instancias *Session*. Es una interfaz no "ligera", y debe realizarse entre muchos hilos de ejecución. Típicamente hay una única *SessionFactory* para toda la aplicación, creada durante la inicialización de la misma. Sin embargo, si la aplicación accede a varias Bases de Datos, se necesitará una *SessionFactory* por cada base de datos.

- La interfaz *Configuration* se utiliza para configurar y "arrancar" *Hibernate*. La aplicación utiliza una instancia de *Configuration* para especificar la ubicación de los documentos que indican el mapeado de los objetos y propiedades específicas de *Hibernate*, y a continuación crea la *SessionFactory.*

- La interfaz *Query* permite realizar peticiones a la Base de Datos y controlar cómo se ejecuta dicha petición (query). Las peticiones se escriben en HQL, o en el dialecto SQL nativo de la Base de Datos que estemos utilizando. Una instancia *Query* se utiliza para enlazar los parámetros de la petición, limitar el número de resultados devueltos por la petición, y para ejecutar dicha petición.

- Los ficheros de configuración de mapeo de *Hibernate* utiliza un objeto denominado Type. Un objeto *Type* de *Hibernate* hace corresponder un tipo Java con un tipo de

- una columna de la base de datos. Todas las propiedades persistentes de las clases persistentes, incluyendo las asociaciones, tienen un tipo *Hibernate* correspondiente. Este diseño hace que *Hibernate* sea altamente flexible y extensible.

2.2.3 Implementación

En esta aplicación el uso que damos a *Hibernate*, es el desarrollo de la capa de persistencia, creando las clases DAO (DATA ACCESS OBJECT), que son clases que implementa métodos de consulta a la base de datos, los cuales nos permite mediante Hibernate guardar, consultar y recuperar datos convertidos a objetos de la Base de Datos.

Hibernate mapea los datos de la Base de Datos, a objetos mediante unos ficheros XML, que indica los atributos de las tablas de nuestra base de datos y su relación con los atributos de una clase que tiene relación con esta tabla.

Ejemplo de fichero XML (Figura 2.2), en este caso seria el fichero de productos, dicho fichero consta de una serie de atributos los cuales serian:

- *Id name* nombre del atributo en nuestra clase.
- *type* el tipo de dicho atributo.
- *column name* nombre de la columna de la tabla de la base de datos.
- *length* longitud de dicho atributo en la base de datos.
- *class name* nombre de la clase que hemos creado y su dirección dentro de nuestra aplicación.
- *table* tabla de la base de datos que esta relacionada con la clase anterior.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<!--
    Mapping file autogenerated by MyEclipse Persistence Tools
-->
<hibernate-mapping>
    <class name="com.plantilla.pojo.Producto" table="productos" schema="public">
        <id name="idproducto" type="java.lang.Long" >
            <column name="idproducto" />
            <generator class="assigned" />
        </id>
        <property name="nombreProducto" type="java.lang.String">
            <column name="nombre_producto" length="25" not-null="true"/>

        </property>
```

```
        <property name="variedad" type="java.lang.String">
            <column name="variedad" length="20" />
        </property>
        <property name="precio" type="java.lang.String">
            <column name="precio" length="10" not-null="true" />
        </property>
        <property name="oferta" type="java.lang.Boolean">
            <column name="oferta" not-null="true" />
        </property>
        <property name="icono" type="java.lang.String">
            <column name="icono" not-null="true" />
        </property>
        <property name="foto" type="java.lang.String">
            <column name="foto" length="25" />
        </property>
         <property name="tipo" type="java.lang.String">
            <column name="tipo" length="8" not-null="true" />
        </property>
         <property name="ecologico" type="java.lang.Boolean">
            <column name="ecologico" />
        </property>
        <property name="procedencia" type="java.lang.String">
            <column name="procedencia" length="20" />
        </property>
        <property name="descripcion" type="java.lang.String">
            <column name="descripcion" />
        </property>
    </class>
</hibernate-mapping>
```

Figura 2.2. Producto.xml.hbm

En este ejemplo se crearía una clase Producto.class (Figura 2.3) relacionada con el fichero de mapeo anterior, dicha clase posee toda los atributos del fichero de mapeo, con sus respectivos métodos *getter* y *setter* para poder acceder a los atributos

```
public class Producto implements java.io.Serializable {

        private String poneroferta;
        private boolean ecologico;
        private String ponerecologico;
        private String procedencia;
        private String descripcion;

        public boolean isEcologico() {
                return ecologico;
        }

        public void setEcologico(boolean ecologico) {
                this.ecologico = ecologico;
        }

        public boolean getEcologico(){
                return this.ecologico;
        }

        public String getProcedencia() {
                return procedencia;
        }

        public void setProcedencia(String procedencia) {
                this.procedencia = procedencia;
        }

        public String getDescripcion() {
                return descripcion;
        }

        public void setDescripcion(String descripcion) {
                this.descripcion = descripcion;
        }
```

```
	public String getPoneroferta() {
		if (this.getOferta()==false){
			this.setPoneroferta("no");
		}else{
			this.setPoneroferta("si");
		}
		return this.poneroferta;
	}

	public void setPoneroferta(String poneroferta) {

		if (poneroferta.compareTo("si")==0){
			this.setOferta(true);
			this.poneroferta="si";
		}else{
			this.setOferta(false);
			this.poneroferta="no";
		}
	}

	public String getPonerecologico() {
		if (this.getEcologico() == false) {
			this.ponerecologico = "no";
		} else
			this.ponerecologico = "si";
		return this.ponerecologico;
	}

	public void setPonerecologico(String ponerecologico) {
		if (ponerecologico.compareTo("si") == 0) {
			this.setEcologico(true);
		} else {
			this.setEcologico(false);

		}
	}
```

Figura 2.3. Clase Producto

Todo esto ficheros anteriores no servirían sin el fichero de configuración de la conexión a la base de datos, este fichero se llamaría *Hibernate.xml* o *Hibernate.config.xml* (Figura 2.4), el cual contiene todos los parámetros de configuración de nuestra conexión a nuestra base de datos y las propiedades de dicha conexión.

```
<bean id="sessionFactory"
class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
    <property name="dataSource" ref="tiendafrutaDS" />
    <property name="mappingResources">
      <list>
			<value>com/plantilla/hbm/Cliente.hbm.xml</value>
			<value>com/plantilla/hbm/Listapedidos.hbm.xml</value>
			<value>com/plantilla/hbm/Pedido.hbm.xml</value>
			<value>com/plantilla/hbm/Producto.hbm.xml</value>
      </list>
    </property>
      <property name="hibernateProperties">
    <props>
      <prop key="hibernate.hbm2ddl.auto">validate</prop>
      <prop key="hibernate.show_sql">true</prop>
<prop key="hibernate.dialect">
org.hibernate.dialect.PostgreSQLDialect</prop>
      <prop key="hibernate.connection.autocommit">true</prop>
    </props>
</property>
</bean>

<bean id="tiendafrutaDS"
```

```
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
      <property name="driverClassName" value="org.postgresql.Driver"/>
<property name="url" value="jdbc:postgresql://localhost:5432/hortobd"/>
<property name="username" value="postgres"/>
<property name="password" value="marauder"/>
</bean>
</beans>
```

Figura 2.4. Fichero de configuración XML

2.2.4 Conclusión

Unos de los problemas mas grandes en la programación era el coste de desarrollar aplicaciones con conexión a las Base de Datos, debido a la complejidad de implementar el código que permitiera conectarse a la base de datos realizando los procesos de abrir una conexión, realizar los procesos pertinentes, y cerrar la conexión, lo cual traía un coste bastante considerable tanto en rendimiento como en implementación, si a eso la añadíamos el problema de tratar esos datos en nuestra aplicaciones aun lo hacia mas complicado.

Con el desarrollo de Lenguaje Orientados a Objeto, la complejidad de desarrollar aplicaciones se ve reducida con su filosofía orientada a Objetos, pero aun así seguíamos teniendo el mismo problema a la hora de conectarse a las Base de Datos para obtener, consultar o recuperar datos, aquí es donde entra la idea del ORM (*Object Relation Mapping*) como una solución a dicho problema, permitiendo conectar nuestra aplicación orientada a objeto con una base de datos relacional, realizando la función de un interfaz entre la base de datos y la aplicación sin que el usuario tenga que implementar complejo y costosos métodos que le permita la interconexión entre la base de datos y la aplicación.

Hay bastantes ORM pero uno de mas conocido y utilizado en el mundo Java es Hibernate, debido a su facilidad de configuración y también a su característica técnicas que permite ser cargado en un contenedor Web como *TomCat* o en servidores de aplicaciones (*Jboss*, *WebSphere*, etc…), por eso el hecho de que mi aplicación implemente este ORM.

2.3 Capa de negocio (*Spring*)

El *Spring Framework* (también conocido simplemente como *Spring*) es un framework de código abierto de desarrollo de aplicaciones para la plataforma Java, aunque hay también una versión para *.NET*. *Spring Framework* se ha popularizado en la comunidad de programadores de Java al considerársele una alternativa y sustituto del modelo de *Enterprise JavaBean*(*EJB*) mas costoso y complicado internamente aparte de necesitar un servidor de aplicaciones donde tener que depositar nuestra aplicación, este modelo es el utilizado por *Sun MicroSystem* aunque no es muy favorecido por cierta mayoría de desarrolladores que opinan que son aplicaciones muy costosas de diseñar y mantener, por eso una alternativa a este modelo de programación es el utilizado por *Spring*. Por su diseño el *framework* ofrece mucha libertad a los desarrolladores en Java con soluciones muy bien documentadas y fáciles de usar para las prácticas comunes en la industria.

Mientras que las características fundamentales de este *framework* pueden emplearse en cualquier aplicación hecha en Java, existen muchas extensiones y mejoras para construir aplicaciones basadas en Web por encima de la plataforma empresarial de Java (*Java Enterprise Platform*).

2.3.1 Características

Entre las muchas características de *Spring* destacamos [5] [16]:

1. Una potente gestión de configuración basada en *JavaBeans*, aplicando los principios de Inversión de Control (*IoC*). Esto hace que la configuración de aplicaciones sea rápida y sencilla. Ya no es necesario tener *singletons* ni ficheros de configuración, una aproximación consistente y elegante. Estas definiciones de *beans* se realizan en lo que se llama el contexto de aplicación.

2. Una capa genérica de abstracción para la gestión de transacciones, permitiendo gestores de transacción añadibles (*pluggables*), y haciendo sencilla la demarcación de transacciones sin tratarlas a bajo nivel.

3. Una capa de abstracción JDBC que ofrece una significativa jerarquía de excepciones (evitando la necesidad de obtener de SQLException los códigos que cada gestor de base de datos asigna a los errores), simplifica el manejo de errores, y reduce considerablemente la cantidad de código necesario.

4. Integración con *Hibernate, JDO e iBatis SQL Maps* en términos de soporte, implementaciones DAO y estrategias con transacciones. Especial soporte a *Hibernate* añadiendo convenientes características de *IoC*, y solucionando muchos de los comunes problemas de integración de *Hibernate*. Todo ello cumpliendo con las transacciones genéricas de *Spring* y la jerarquía de excepciones DAO.

5. Funcionalidad *AOP* (Programación Orientada a Objetos), totalmente integrada en la gestión de configuración de *Spring*. Se puede aplicar *AOP* a cualquier objeto gestionado por *Spring*, añadiendo aspectos como gestión de transacciones declarativa. Con *Spring* se puede tener gestión de transacciones declarativa sin *EJB*.

6. Un *framework* MVC (Model-View-Controller), construido sobre el núcleo de *Spring*. Este *framework* es altamente configurable vía interfaces y permite el uso de múltiples tecnologías para la capa vista como pueden ser *JSP, Velocity, Tiles, iText o POI*. De cualquier manera una capa modelo realizada con *Spring* puede ser fácilmente utilizada con una capa Web basada en cualquier otro *framework* MVC, como *Struts, WebWork , Tapestry o JSF*.

2.3.2 Arquitectura

La arquitectura [16] en capas de *Spring* ofrece mucha de flexibilidad. Toda la funcionalidad está construida sobre los niveles inferiores. Por ejemplo se puede utilizar la gestión de configuración basada en *JavaBeans,* sin utilizar el *framework* MVC o el soporte AOP.

La arquitectura de *Spring* esta basada en un patrón de diseño llamado "***Dependency Injection***" y en ***IoC(Inversion of Control).***

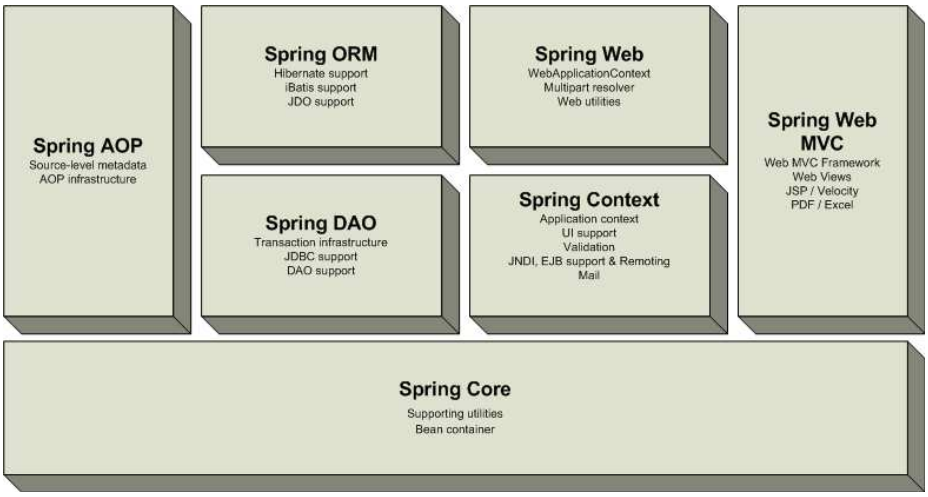


Figura 2.5 Arquitectura de Spring

IoC es un concepto que utiliza *Spring* y que se basa en que cierta clase A que llama a métodos de otra clase B, no tiene por que inicializar un objeto de dicha clase A para utilizar dicho método, sino que un agente externo en este caso el contenedor de *Spring* realizaría esa función y solamente la clase A tendría que ejecutar el método directamente.

Esta filosofía se basa en la dependencia que hay entre los distintos componentes de una aplicación ya que cierto componente pueden depender de cierto método de otros componente, de esta manera un agente intermediario permitiría la conexión de dicho componente sin que ambos se tengan que inicializar o destruir dentro de su contexto ya que esta funciones las realizaría dicho agente intermedio.

El patrón *Dependency Injection* se basa en la idea anteriormente comentada, es un patrón de diseño orientado a objetos, en el que se inyectan objetos a una clase en lugar de ser la propia clase quien cree el objeto, consideremos *la Figura sprintA* la cual es un diagrama de clases, la *clase Foo* depende de una instancia de la *clase Bar*, para realizar algún tipo de procesamiento, tradicionalmente en la *clase Foo* se tendría la sentencia *Bar bar=new Bar();* para crear el objeto *bar*, usando *Dependency Injection*, una instancia de *Bar* (o bien una subclase) es proporcionada a la clase *Foo* en tiempo de ejecución por algún proceso externo, es decir *la clase Foo* no llama a crear el objeto *Bar* si no que el proceso externo le proporciona el objeto *Bar* a *la clase Foo*, es por eso que se define a *Dependency Injection* con la frase: *"No me llames, yo te llamo"*.

Spring soporta varios tipos de *Dependecy Injection*, pero en si estos son los más utilizados:

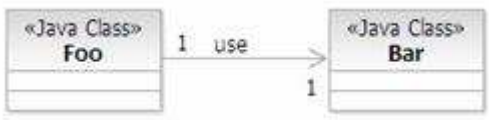


Figura 2.6 Dependencia entre clases

1. *Setter Inyection*:en este tipo la inyección de dependencia es aplicada por medio de métodos *JavaBeans setters*, que a la vez tiene un *setter* respectivo.

2. Constructor *Inyection*: esta inyección es a través de los argumentos del constructor.

2.3.3 Núcleo de *Spring*

Esta parte es la que provee la funcionalidad esencial del *framework*, está compuesta por el *BeanFactory*, el cual utiliza el patrón de Inversión de Control y configura los objetos a través de Inyección de Dependencia, el núcleo de *Spring* es el paquete org.springframework.beans el cual esta diseñado para trabajar con *JavaBeans*.

El *BeanFactory* es unos de los componentes principales de *Spring*, *BeanFactory* es de propósito general, ya que puede crear muchos tipos diferentes de *Beans*, Los *Beans* pueden ser llamados por su nombre y se encargan de manejar las relaciones entre objetos.

También tiene la propiedad de aportar objetos de dos modos diferentes:

- *Singleton*: Existe únicamente una instancia compartida de un objeto con un nombre particular, que puede ser regresado o llamado cada vez que se necesite. Este es el método mas común y el mas usado.

- *Prototype*: (*non-sigleton*): en este método cada vez que se realiza un regreso o una llamada, se crea un nuevo objeto independiente.

La implementación de *BeanFactory* más usada en la tecnología de *Spring* sería la org.springframework.beans.factory.xml.XmlBeanFactory, que carga las definiciones de cada *Bean* que se encuentra guardado en un archivo XML que consta de:

- Id (nombre con el que se conoce a la clase).
- clase (tipo de *bean*).
- *Singleton* o *Prototype* (modos del *Bean* antes mencionados), propiedades, con sus atributos name, value y ref, argumentos del constructor, métodos de inicialización y métodos de destrucción.

Ejemplo de una posible definición de un *bean* en un fichero XML ver Figura 2.7

```
<beans>
    <bean id="Ejemplo" class="src.Miejemplo" singleton="true"/>
      <property name="driverClassName"value="com.Postgres.jdbc.Driver"/>
</beans>
```

Figura 2.7 Definición de un bean

Para cargar dicho fichero XML se haría de la siguiente manera:

```
BeanFactory Fac=new XmlBeanFactory(new FileInputStream ("bean.xml"))
```

Una vez que la definición es cargada, únicamente cuando se necesite el *Bean* se creara una instancia dependiendo de sus propiedades, para tomar un *Bean* de un *Factory* se usa el método getBean(), dándole el nombre del *Bean* a obtener.

```
MyBean mybean=(MyBean) factory.getBean("mybean")
```

2.3.4 Implementación

En nuestra aplicación el uso que le damos a *Spring* es en la capa de negocios, este *framework* nos va a permitir enlazar la capa de presentación con la capa de persistencia, también va a realizar toda la lógica de negocio de nuestra aplicación con el uso de *bean* (clase java con atributos, métodos *setter* y *getter* para obtener y modificar dichos atributos). Toda esta implementación se va a definir mediante el uso de un fichero de configuración llamado *ApplicationContext.xml*.

El fichero *ApplicationContext* es una subinterfaz de *BeanFactory*, por lo que todo lo que realiza el *BeanFactory* lo realiza el *ApplicationContext*.

Características del *ApplicationContext*:

- Localización y reconocimiento automático de las definiciones de los *Beans*
- Cargar múltiples contexto
- Contexto de herencia
- Búsqueda de mensajes para encontrar su origen
- Acceso a recursos
- Propagación de eventos, para permitir que los objetos de la aplicación puedan publicar y opcionalmente registrarse para ser notificado de los eventos.
- Agrega soporte para internacionalización (i18n)

En nuestra aplicación hemos definido nuestro *AplicationContext.xml* el cual hemos definido en la Figura 2.8

```
<bean id="ref_transactionManager"
   class="org.springframework.orm.hibernate3.HibernateTransactionManager">
        <property name="sessionFactory" ref="sessionFactory" />
</bean>

<bean id="txProxyTemplate" abstract="true"
        class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean">
<property name="transactionManager"
        ref="ref_transactionManager" />
                <property name="transactionAttributes">
                        <props>
                                <prop key="save*">PROPAGATION_REQUIRED</prop>
                                <prop key="remove*">PROPAGATION_REQUIRED</prop>
                                <prop key="delete*">PROPAGATION_REQUIRED</prop>
                                <prop key="update*">PROPAGATION_REQUIRED</prop>
                                <prop key="*">PROPAGATION_REQUIRED</prop>
                        </props>
                </property>
        </bean>


        <bean id="ProductoDAO"
                class="com.plantilla.persistencia.ProductoDAO">
                <property name="sessionFactory">
                        <ref bean="sessionFactory" />
                </property>
        </bean>

        <bean id="PedidoDAO" class="com.plantilla.persistencia.PedidoDAO">
                <property name="sessionFactory">
```

```
                        <ref bean="sessionFactory" />
                </property>
        </bean>

        <bean id="ListapedidosDAO"
                class="com.plantilla.persistencia.ListapedidosDAO">
                <property name="sessionFactory">
                        <ref bean="sessionFactory" />
                </property>
        </bean>


        <bean id="clienteDAO"
                class="com.plantilla.persistencia.ClienteDAO">
                <property name="sessionFactory">
                        <ref bean="sessionFactory" />
                </property>
        </bean>


        <bean id="clienteServicioImpl" parent="txProxyTemplate">
                <property name="target">
                        <bean

        class="com.plantilla.negocio.implementacion.ClienteServicioImpl">
                                <property name="clienteDao" ref="clienteDAO" />
                        </bean>
                </property>
        </bean>

        <bean id="productoServicioImpl" parent="txProxyTemplate">
                <property name="target">
        <bean

        class="com.plantilla.negocio.implementacion.ProductoServicioImpl">
                                <property name="productoDao" ref="ProductoDAO" />
                        </bean>
                </property>
        </bean>




<bean id="pedidoServicioImpl" parent="txProxyTemplate">
  <property name="target">
        <bean   class="com.plantilla.negocio.implementacion.PedidoServicioImpl">
        <property name="pedidoDao" ref="PedidoDAO" />
<property name="listapedidosDao"  ref="ListapedidosDAO" />
        </bean>
  </property>
</bean>
</beans>
```

Figura 2.8 Fichero ApplicationContext.xml

2.3.5 Conexión de *Spring* con las Bases de Datos

Spring no posee un ORM [16] propio para evitar tener que utilizar JDBC para la conexión a la base de datos, pero si provee de una API para la conexión con los distintos ORM existente en el mercado entre ellos *Hibernate, Oracle Toplink, iBatis etc...*

Entre las ventajas que ofrece *Spring* con herramientas ORM son:

- Manejo de sesión, *Spring* abstrae a programador de la programación del ORM dándole una forma más eficiente, sencilla y segura de manejar sesiones aunque un poco más costosa.

- Manejo de recursos: se pueden manejar y configurar los ficheros de configuración del ORM con la base de datos, de una forma más fácil y aprovechando dichos fichero para su uso en *Spring*, y con lo cual en nuestra lógica de negocios.

- Manejo de transacciones integrado.

- Envolver excepciones: todas las excepciones son gestionadas por *Spring* para tener que evitar las declaraciones y los catch en cada segmento.

2.3.6 *Spring* DAO

El patrón DAO (DATA ACCESS OBJECT) es una de los patrones mas importante y usados en aplicaciones J2EE, y la arquitectura de acceso a datos de *Spring* provee de un buen soporte para dicho patrón [16].
Spring provee de dos opciones para llevar a cabo el acceso, conexión y manejo de base de datos, utilizar herramientas ORM o utilizar JDBC, la elección de cada una de ellas es libre.

El programador podrá elegir cada una de las herramientas en función de la complejidad de la aplicación, para aplicaciones simple se esta utilizado JDBC y para aplicaciones mas complejas se esta utilizando ORM.

El uso de JDBC hace que muchas veces se repita el mismo código en distintos sitios de nuestra aplicación para crear la conexión, buscar información, procesar resultados y cerrar la conexión, lo cual provoca cierto fallos si el desarrollador se le olvida cualquiera de alguno de estos parámetros, mientras que utilizando *Spring* con una ORM no pasaría esto, ya que utiliza una API que abstrae al desarrollador de tener que realizar todos los pasos anteriormente mencionados.

En el caso de nuestra aplicación nos hemos decantado por la primera opción creando así unos DAO que implementa *Spring* sobre *Hibernate* ver Figura 2.8.

2.3.7 Conclusión

Spring es perfecto para ser incrustado en nuestra lógica de negocio, ya que da una fuerte funcionalidad entre las distintas capas de nuestra aplicación, sin que se cree una dependencia total entre las distintas capas y sobre todo con este *framework*, de tal manera que es posible retirarlo sin tener que cambiar líneas de código, Lo único que habría que hacer, lógicamente es añadirle funcionalidad, bien con otro *framework* o con nuestro código, a parte este *framework* posee una fuerte comunidad de desarrollo detrás que esta constantemente mejorándolo.

```
public class ProductoDAO extends HibernateDaoSupport {
     private static final Log log = LogFactory.getLog(ProductoDAO.class);
     // property constants


     protected void initDao() {
            // do nothing
     }

     public void save(Producto transientInstance) {
            log.debug("saving Productos instance");
            try {
                   getHibernateTemplate().save(transientInstance);
                   log.debug("save successful");
            } catch (RuntimeException re) {
                   log.error("save failed", re);
                   throw re;

            }
     }

     public void update(Producto transientInstance) {
            log.debug("updating Productos instance");
            try {
                   getHibernateTemplate().update(transientInstance);
                   log.debug("save successful");
            } catch (RuntimeException re) {
                   log.error("save failed", re);
                   throw re;

            }
     }


     public void delete(Producto persistentInstance) {
            log.debug("deleting Productos instance");
            try {
                   getHibernateTemplate().delete(persistentInstance);
                   log.debug("delete successful");
            } catch (RuntimeException re) {
                   log.error("delete failed", re);
                   throw re;
            }
```

```
        }

        public Producto findById(java.lang.Long id) {
                log.debug("getting Producto instance with id: " + id);
                try {
                        Producto instance = (Producto) getHibernateTemplate().get(
                                        "com.plantilla.pojo.Producto", id);
                        return instance;
                } catch (RuntimeException re) {
                        log.error("get failed", re);
                        throw re;
                }
        }
}
```

Figura 2.8 Ejemplo de ProductoDAO.class de nuestra aplicación

Spring también posee una fuerte conectividad entre las capas de presentación y la capa de persistencia, con lo cual es perfecto para nuestro desarrollo como interfaz de conexión entre estas capas.

Con su tecnología de *IoC* nos permite acceder directamente a los métodos de cierta clase sin tener que inicializar dicha clase, ya que con su contenedor de *beans* solo tendríamos que recoger un *bean* en concreto e invocar al método que nos haría falta para nuestra aplicación.

2.4 Capa de presentación (JSF)

JavaServerFaces (JSF) es un *framework* basado en el patrón MVC (Modelo Vista Controlador) para aplicaciones Java basadas en Web que simplifica el desarrollo de interfaces de usuario en aplicaciones Java J2EE. Este *framework* ha sido desarrollado por *Sun Microsystem* para su uso en la tecnología J2EE, aunque se puede implementar con otro *framework* como *Spring, Struts etc.,*

Su uso se realiza en la capa de presentación, ya que nos facilita una gran cantidad de componentes para su uso en la Web, aparte posee un controlador que nos dirige de una pagina otras si que tengamos que implantar complejos sistemas de control para el direccionamiento y control de las paginas Web.

JavaServerFaces nos provee con una gran cantidad de componente para implementar en paginas JSP mediante el uso de etiquetas, esto es una gran ventajas ya que unos de los problemas que tenia JSP era que se podía incrustar código Java dentro de una pagina

Web, quedando un código mezclado de HTML y JavaScript, difícil de entender en muchos casos.

JSF ha sido completamente liberado por *Sun Microsystem*, de tal manera que se esta desarrollando numerosos proyecto de mejoras basado en este *framework*, uno de ellos es *Apache MyFaces* que se basa en *JSF*, y nos provee de numerosas mejoras y componentes. A su vez sobre este proyecto se están desarrollando otros proyectos como son *Apache Tomahawk, Apache Trinidad, RichFaces, IceFaces, etc.…* [9][10][11] esto proyecto poseen nuevos componentes que le dan una interfaz mas intuitiva y interactiva con el usuario, y a la vez permite que el programador tenga una gran cantidad de componentes para utilizar sin tener que implementar uno suyo propio.

2.4.1 Características

Entre las muchas características que nos provee JSF, podemos destacar estas como de las más importantes a tener en cuenta [17] [18]:

- Tecnología ejecutada del lado del servidor y no del cliente.
- La interfaz de usuario es tratada como un conjunto de componentes UI.
- JSF es un *framework* muy amigable, ya que provee de una gran cantidad de componentes *Drag & Drop* para su utilización en la construcción de la página Web.
- Se puede utilizar con tecnología JSP o independientemente de esta tecnología.
- Es altamente escalable permitiendo incrementar la funcionalidad del programa.
- Control de navegación mediante un fichero denominado *Faces-Config.xml* en el cual se implementa la navegación de una pagina a otra en función de unos eventos producido en las páginas.

- Separación del código Java de los componentes UI que se implementa en la Web, de tal manera que queda un código limpio tanto en el lado de la pagina Web que implementa el desarrollador como en la parte de implementación de Java, evitando así tener que meter código JSP dentro de código HTML.

- Provee de una gran cantidad de componentes para los desarrolladores, que puede implementar en sus páginas Web.

- Da unas reglas para desarrollar nuevos componentes que pueden ser utilizados por los demás desarrolladores desarrolle con JSF.

- Validar los datos de los componentes UI.
- Uso de etiqueta *tag* para el control de componentes UI permitiendo un código mas limpio.

2.4.2 Arquitectura

La arquitectura [17] [18] de JSF se basa en MVC (modelo- vista-controlador) separando la vista de controlador y este a su vez del modelo, de tal manera que esta todo conectados entre si pero separados en distintas partes.

En la Figura 2.9 podemos ver dicha arquitectura, y las distintas tres partes del patrón de diseño Modelo-Vista-Controlador, explicaremos las tres partes detenidamente:

- **Vista**

  Consta de componentes UI (*RadioButton, inputTextArea, etc*) que se implementan mediante una **etiqueta** (o *tag*), lo cual deja un código limpio, sin mezcla de código Java con HTML como pasa con las páginas JSP. Estas etiquetas poseen una serie de propiedades que permite modificar los componentes UI que estamos utilizando. Esto componentes UI se pueden conectar con un *BackingBean* que se definiría en el fichero *Faces-config* de nuestra aplicación, dicho *BackingBean* posee atributos, y sus respectivos métodos *setter* y *getter* que nos permite consultar o modificar el atributo desde el componente UI, que seria el componente que vería el usuario en la pagina.

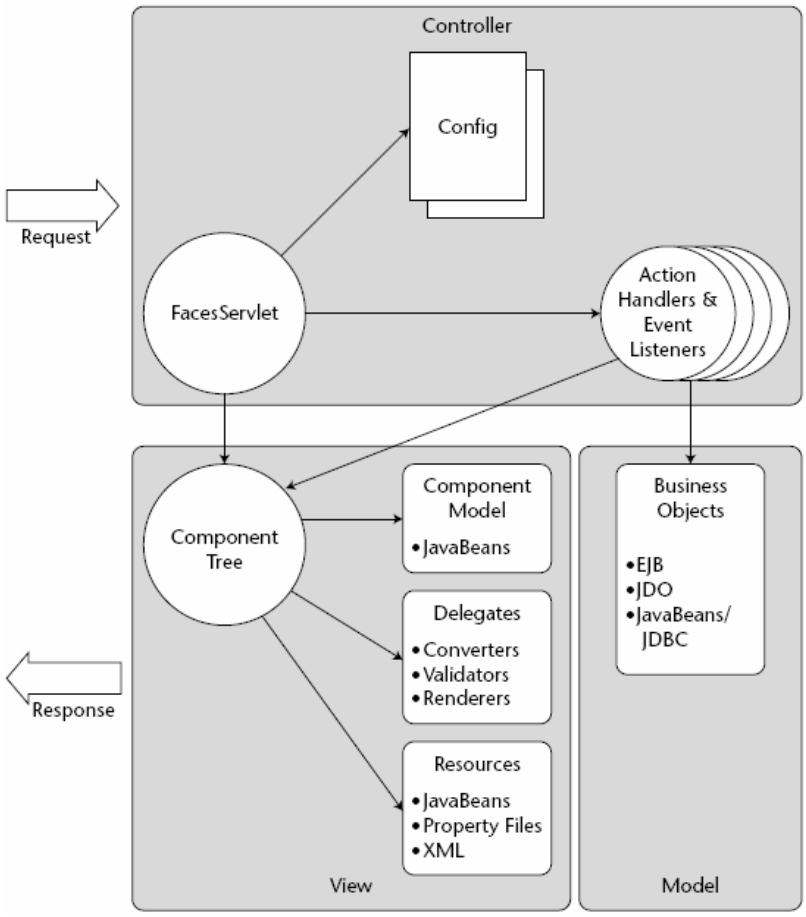


Figura 2.9 Arquitectura JSF

- **Controlador**

  Tiene varias funciones a realizar, las cuales son gestionar toda la navegación Web de nuestra aplicación, controlar todos los eventos que se produzcan y poder gestionarlos, procesar las validaciones de los datos que introduzcamos en los componentes, ejecutar los *beans* cuando sean llamado por un componente, y actualizar los datos de dichos componentes cuando sea necesario.

- **Modelo**

  El modelo seria en nuestro caso nuestra lógica de negocio, para ello *Spring* nos permite conectar los *beans* que tenemos definidos para que puedan ser llamados desde JSF, y a su vez implementarse en los *Backingbeans* de JSF permitiendo conectar nuestra capa de presentación con la capa de negocio.

2.4.3 Implementación

En la implementación de nuestra aplicación hemos utilizado distintos componentes de JSF, lo cual nos ha permitido el desarrollo de una interfaz muy interactiva y bastante amigable en la Figura 2.10 podemos ver código de la aplicación, este código es bastante limpio ya que solo utilizamos etiquetas JSF que implementa dicho componentes UI

Algunas de las etiquetas que hemos utilizado llaman a un *BackingBean*, que implementa métodos Java que vamos a utilizar o alguna propiedad de la clase. Para llamar a un *Backing Bean*, lo haríamos llamándolo con *"#{nombreBackingBean.metodo}"*, en alguna de las propiedades que nos dan las etiquetas y que nos permita llamar a ese *Backing Bean*, a parte para poder llamarlo tendríamos que definirlo en nuestro *Face-config*, para ello lo haríamos como indicamos a continuación [17].

```
<rich:panel id="loginregistrado" rendered="#{usuario.registrado}">
        <f:facet name="header">
                <h:outputText value="LOGIN"></h:outputText>
        </f:facet>
        <h:outputText value="Usuario " />
        <h:panelGroup >
                <h:outputText value="#{usuario.username}" />
                <h:outputText value=" en sesion " />
                <f:verbatim>
                <br>
                </f:verbatim>
                <h:commandLink value="cerrar sesion"
action="#{usuario.logout}"></h:commandLink>
                <h:commandLink value="Modificar Datos" action="#{cliente.Mostrar}"
style="position:relative; left:20px"></h:commandLink>
                <h:commandLink value="Administracion" action="admin"
rendered="#{usuario.administrador}" style="position:relative;
left:40px"></h:commandLink>
        </h:panelGroup>
</rich:panel>
```

Figura 2.10 Pagina JSF con tag

- Definición Jsf tag

```
        <h:commandLink id="loginboton" value="Enviar"
action="#{usuario.ComprobarUsuario}"
              style="color:black;left: 80px; width: 80px; position: relative; height:
24px"
        </h:commandLink>
```

- Implementación en el Faces-config de usuario, las propiedades que define este bean seria:
    1. <managed-bean-name> nombre de *bean* en la pagina JSF
    2. <manager-bean-class> ruta de la clase que implementa el *bean*

3. <manager-bean-scope> indica si el *bean* se mantiene activo durante la sesión, mientras dure la aplicación, o cuando se haga un *request*, en este caso se haría durante la sesión.

4. <manager-property>indica un atributo y sus valores, aquí podríamos definir un *bean* de *Spring* que utilizáramos en JSF, para ello habría que definir en el *Faces-config* que vamos a utilizar los *bean* de *Spring*.

```
<managed-bean>
		<managed-bean-name>usuario</managed-bean-name>
		<managed-bean-class>
			com.plantilla.presentacion.beans.UsuarioBean
		</managed-bean-class>
		<managed-bean-scope>session</managed-bean-scope>
		<managed-property>
			<property-name>pedidoservicio</property-name>
			<value>#{pedidoServicioImpl}</value>
		</managed-property>
</managed-bean>
```

- Clase usurioBean con el método llamado anteriormente

```
public class UsuarioBean extends BasePage {

  public String ComprobarUsuario() throws Exception {
		String status = "failure";
		ClienteServicio dao = (ClienteServicio)
				getSpringBean("clienteServicioImpl");
		this.cliente = dao.buscarUsuario(this.getUsername(),
this.getPassword());

		if (cliente != null) {
			if (cliente.getUsername().compareTo("administrador") ==0 )
{
				this.setIdcliente(this.cliente.getIdcliente());
				this.setDireccion(this.cliente.getDireccion());

	this.setCodigoPostal(this.cliente.getCodigoPostal());
				this.setApellido(this.cliente.getApellido());
				this.setNombre(this.cliente.getNombre());
				this.setTelefono(this.cliente.getTelefono());
				this.setCliente(this.cliente);
				this.setAdministrador(true);
				this.exist = false;
				this.noregistrado = false;
				this.registrado = true;
				status = "success";


			} else {
				this.setIdcliente(this.cliente.getIdcliente());
				this.setDireccion(this.cliente.getDireccion());

	this.setCodigoPostal(this.cliente.getCodigoPostal());
				this.setApellido(this.cliente.getApellido());
				this.setNombre(this.cliente.getNombre());
				this.setTelefono(this.cliente.getTelefono());
				this.setCliente(this.cliente);
				this.exist = false;
				this.noregistrado = false;
				this.registrado = true;
				status = "success";
		}
		} else {
```

```
                        this.exist = true;

                }
                return status;
            }
        }
```

Por lo que se refiere a la gestión de la navegación Web se definiría en el *Face-config*, las propiedades que define la navegación serian:

- <front-view-id> indica la página de inicio
- <navigation-case> definimos las reglas de navegación
- <from-outcome> aquí indicamos una cadena de texto que nos obligaría a ir a la página definida en <to-view-id>

```
<navigation-rule>
        <from-view-id>/pedidorealizado.jsp</from-view-id>
        <navigation-case>
                <from-outcome>seguir</from-outcome>
                <to-view-id>/tienda.jsp</to-view-id>
        </navigation-case>
</navigation-rule>
```

Para implementar y configurar JSF en nuestra web tendríamos que configurar ciertas propiedades en el fichero Web.xml, en nuestra aplicación lo hemos definido como indica la Figura [2.11], en ella definimos tres parámetros XML importantes, los cuales explicamos a continuación:

- servlet: nombre del servlet que controla la navegación y los eventos que se produzca en los componentes y respuesta ante dichos eventos.
- servlet-mapping: indica el nombre del sevlet que realiza la función de mapeado de la web, y las extensiones permitida para las paginas, en este caso solo permitiría paginas terminadas en .faces.

- <welcome-file-list> indica el fichero de inicio de la aplicación en este caso seria *index.jsp*.

```
    <servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
```

```
        <servlet-mapping>
                <servlet-name>Faces Servlet</servlet-name>
                <url-pattern>*.faces</url-pattern>
        </servlet-mapping>

        <welcome-file-list>
                <welcome-file>index.jsp</welcome-file>
        </welcome-file-list>
```

Figura 2.11 Datos a incluir para configura JSF en el web.xml

Anteriormente comentamos que podíamos utilizar los *Bean* de *Spring* en JSF, para ello hemos de definir en el *Faces-Config* los siguientes parámetros [Fig 2.12].

```
<application>
    <variable-resolver>
        org.springframework.web.jsf.DelegatingVariableResolver
     </variable-resolver>
</application>
```

Figura 2.12 Conexión Spring-JSF

2.4.4 Conclusión

JSF es utilizado en nuestra aplicación en la capa de presentación, hay varios *framework* que podíamos haber utilizado para esta capa como Spring, Struts, Tapestry, etc…, pero hemos preferido JSF porque es un framework bastante robusto, ofreciéndonos un gran cantidad de componentes que podemos utilizar en nuestra aplicación para el desarrollo de las paginas Web, y sobre todo por que hay bastantes proyectos de desarrollo(Richfaces, ICEfaces ,Apache Tomahawk), que traen numerosos componentes que podemos utilizar sin tener que implementar.

JSF es una buena herramienta de desarrollo para la parte Web, que nos permite ahorrar bastante tiempo a la hora de crear paginas Web, y sobre todo deja un código bastante limpio, otras de las ventajas es la inclusión de Ajax en algunos de sus componentes permitiéndonos realizar una interfaz mas interactiva con el usuario de tal manera que parezca una aplicación de escritorio.

2.5 *RichFaces*

RichFaces es una extensión sobre JSF desarrollado por Exadel para el desarrollo de aplicaciones RIA (*Rich Internet Application*) que nos da ciertas mejoras en la interfaz de la aplicación Web, entre las mejoras que trae, *RichFaces* permite:

- Incluir *Ajax* en todos los componentes *JSF*, ya que *RichFaces* esta completamente integrado en el ciclo de vida de JSF.

- Crear complejas vistas , dándole una interfaz de escritorio

- Escribir tus propias componentes *Rich* con soporte para *Ajax*, creándola desde cero o aprovechando las que hay, y mejorándolas.

No voy a describir todas las propiedades que tiene, ya que posee bastante y llevaría bastante tiempo explicarlas todas, así que recomiendo el manual de desarrollo de Richfaces que se encuentra disponible en la web en formato PDF, por lo que se refiere a su arquitectura, al estar basado en JSF, ya conocemos su arquitectura. Por lo que se refiere a la inclusión de *Ajax* en sus componentes nos permite manejar eventos que se produzcan en el escritorio y hacer más interactivo la aplicación en el lado del cliente.

2.5.1 Requerimientos Técnicos

Richfaces ha sido desarrollado con una arquitectura para ser compatible con la mayoría de los navegadores Web actuales.
Las herramientas software compatible con esta aplicación, se describen a continuación:

- Java
- JSF
- Contenedor Web
- Navegador
- *RichFaces*
- JDK 1.4
- Sun JSF 1.1 RI - 1.2
- MyFaces 1.1.1 - 1.2

- Facelets JSF 1.1.1 - 1.2
- Seam 1.2. - 2.0
- Internet Explorer 6.0 - 7.0
- Firefox 1.5 - 2.0
- Netscape 7.0
- Safari 3.0
- Apache Tomcat 4.1 - 6.0
- IBM WebSphere 5.1 - 6.0
- BEA WebLogic 8.1 - 9.0
- Oracle AS/OC4J 10.1.3
- Resin 3.0
- Jetty 5.1.X
- Sun Application Server 8 (J2EE 1.4)
- Glassfish (J2EE 5)

2.5.2 Conclusión

Richfaces es una herramienta bastante potente a la hora de desarrollar aplicaciones RIA, y con la inclusión de *Ajax* en sus componentes nos permite realizar aplicaciones de Internet, parecidas a la de escritorio.
RichFace nos es la única herramienta para el desarrollo de aplicaciones RIA, hay muchas más aplicaciones RIA basada en JSF, como por ejemplo *IceFaces* que también la podemos incluir en nuestra aplicaciones.

2.6.1 MyEclipse

MyEclipse es un software de pago basado sobre Eclipse, que ha tenido un gran éxito en la comunidad de los desarrolladores de Java. Las ventajas que nos da es la gran cantidad de *plugin* y herramientas que nos permite ahorrar un gran cantidad de tiempo a la hora de desarrollar nuestra aplicación Web, ya que traen gran cantidad de extensiones para *JSF, AJAX, Hibernate, etc.…*

El hecho de manejar las herramientas anteriores es la decisión por la cual he decidido utilizar este IDE, ya que es de fácil manejo, y no tiene una complejidad muy grande en comparación con *Eclipse* a la hora de manejarlo.

Otro de los elementos clave de su éxito es su accesibilidad; tiene un precio que lo hace accesible a cualquier desarrollador; dándole a éste la posibilidad de pagar la licencia de uso y no depender de la compañía en la que trabaja para que pague un par de miles de dólares por otro IDE que al final de cuantas no ofrece una ventaja que justifique el precio. El precio de *MyEclipse* hasta ahora es de $49.95/por año.

Otras de las ventajas que nos da *MyEclipse* es la gran cantidad de tutoriales que hay en su pagina Web, para ser utilizado con esta herramienta a la hora de desarrollar nuestras aplicaciones o ciertas partes de la aplicación, por ello considero que es una buena herramienta para meterse en la creación de aplicaciones J2EE y sin una complejidad excesiva en su manejo.

Capítulo 3:
Aplicación de la metodología Métrica 3

Este capítulo tratará de analizar, diseñar y construir el sistema de información resultante del proyecto realizado. Esta metodología consta de una serie de procesos, de entre los cuales, los que vamos a realizar son:

- Estudio de Viabilidad del Sistema (EVS). Pretende analizar un conjunto de necesidades dando una solución a corto plazo.

- Análisis del Sistema de Información (ASI). Especificación detallada del sistema de información que satisfaga las necesidades propuestas.

- Diseño del Sistema de Información (DSI). Se define la arquitectura del sistema y el entorno tecnológico que le dará soporte.

- Construcción del Sistema de Información (CSI). Se genera el código de los componentes

3.1 ESTUDIO DE VIABILIDAD DEL SISTEMA (EVS)

En esta actividad se estudia el alcance de la necesidad planteada por el cliente o usuario, realizando una descripción general de la misma. Se determinan los objetivos, se inicia el estudio de los requisitos y se identifican las unidades organizativas afectadas estableciendo su estructura. Se analizan las posibles restricciones, tanto generales como específicas, que puedan condicionar el estudio y la planificación de las alternativas de solución que se propongan.

3.1.1 ACTIVIDAD EVS 1: ESTABLECIMIENTO DEL ALCANCE DEL SISTEMA

Tarea EVS 1.1: Estudio de la solicitud

Se realiza una descripción general de la necesidad planteada por el usuario, y se estudian las posibles restricciones de carácter económico, técnico, operativo y legal que puedan afectar al sistema. Antes de iniciar el estudio de los requisitos del sistema se establecen los objetivos generales del Estudio de Viabilidad, teniendo en cuenta las restricciones identificadas anteriormente.

Descripción general del sistema

Se pretende desarrollar un sistema de información que gestione a través de la web todos los pedidos que los clientes realicen a una tienda de productos hortofrutícolas, esta página gestionara los pedidos y los almacenara para su procesado y envió al cliente.

A continuación describo las principales funciones de esta aplicación.

1. Tienda
    - Visualización de los productos a comprar: Se mostrará una tabla de datos paginada que mostrara imágenes de productos.
    - Información de cada producto: Cada producto mostrado en la celda de la tabla de datos, mostrará una pequeña información del producto como nombre, variedad, cantidad, etc.
    - Cantidad a comprar: La aplicación posee un carrito de la compra para cada producto, así como la cantidad en Kg. a comprar.
    - Información detallada del producto: Cuando se pulse sobre el icono de algunos de los productos mostrado en la tabla, esto no dirigirá a una tabla donde nos mostrara una información más detallada del producto (nombre, variedad, procedencia, información nutricional de producto etc.).
    - Sellos de oferta y ecológico: Cada producto puede poseer un icono de oferta y otro de ecológico.

- Listados de productos: Podemos ver listado de frutas, verduras, frutos secos, otros productos y ofertas generales de frutas, verduras, frutos secos y otros productos.

- Opción de búsqueda: El usuario puede buscar el nombre de un producto y en la pantalla se mostrará todos los productos relacionados.

- Visualización de los datos de compra: La página mostrará todos los productos comprados por el cliente, donde podrá modificar la cantidad e incluso borrarlos de la lista, los datos mostrados serán foto del producto, variedad, cantidad, precio, eliminación del producto.

- Total de compra: Se ira mostrando el total de la compra que lleva el usuario.

- Datos de entrada: El sistema mostrara un nombre de usuario y una contraseña para confirmar que el usuario es el que realiza la compra, no pidiéndose hasta que realice la compra.

- Opciones de registro: Si el usuario no esta registrado, se podrá registrar en la página mediante la opción de registrarse.

- Modificar sus datos: El cliente puede modificar sus datos personales.

- Cerrar sesión: El usuario puede cerrar la sesión.

- Confirmar pedido: El usuario una vez registrado, puede confirmar y realizar su pedido.

2. Confirmación de pedido

   - El usuario una vez que ha confirmado el pedido, puede ver sus datos de pedido, junto con los productos, no pudiendo modificar la cantidad de dichos productos.

- Modificar datos de envío y llegada. El usuario puede modificar los datos de envío, hora de llegada del producto y las observaciones que considere pertinente.

3. Administración

- Puede realizar todas las opciones del cliente.

- Datos de productos

    1. Alta de producto. Puede dar de alta un producto, insertando los datos del producto.

    2. Modificación del producto. Indicaría el nombre del producto a modificar, apareciéndole todos los datos del producto a modificar.

    3. Salir. Es la opción de salir de los datos del producto.

- Datos del pedido

    1. Listado de pedidos sin procesar. Da un listado de pedidos sin procesar, dando información del pedido realizado por el cliente, pudiendo modificarlo para indicar que esta procesado, o para cambiar algunos de los datos del pedido.

    2. Listado de pedido procesados. Da un listado de pedidos procesados, pudiéndose modificar para pasarlo de pedido procesado, ha pedido no procesado.

    3. Salir. Saldría de los datos de pedido.

- Salir. Da la opción de salir del menú del Administrador.

Catálogo de objetivos.

El objetivo del EVS es obtener información lo más detallada posible del sistema de información que se va a desarrollar. Para ello, se obtendrá un conjunto de funcionalidades mínimas para desarrollar una primera solución, contando con las diferentes restricciones, como por ejemplo, técnicas y económicas, entre otras.

Catálogo de requisitos.

Este catálogo se ve en la Tabla 3.1, donde está numerado cada punto que se debe realizar en la solución.

Tarea EVS 1.2: Identificación del alcance del sistema.

Descripción general del sistema.

En este apartado haremos una distinción del sistema; se dividirá en dos partes:

a) Contexto del sistema: Todo lo descrito en el catálogo de requisitos, y sólo eso, entrará en el contexto del sistema, aunque si el cliente lo solicita, se hará dichas partes que se encuentren fuera de nuestro sistema.

Nº	Requisitos	Prioridad	Tipo	Descripción
1	Gestión de clientes	Alta	Funcional	Gestión de datos de los clientes de la tienda, se podrá modificar los datos de envió.
2	Gestión de pedidos	Alta	Funcional	Gestión de los pedidos que realizan los cliente, pudiéndose modificar los datos, y su estado de procesamiento.
3	Gestión de productos	Alta	Funcional	Gestión de los productos que hay en la aplicación Web, pudiéndose modificar, dar alta y buscar.
4	Listado de pedidos	Alta	Funcional	Listado de todos los pedidos procesados y no procesados que se han realizado.
5	Imágenes de producto	Alta	Funcional	Todas la imágenes que hay en la web, tanto del producto minimizado, como más detallado.

Tabla 3.1 Catalogo de requisitos

b) Estructura organizativa: La estructura organizativa del sistema viene dada por el esquema mostrado en la Figura 3.1. Al ser un programa de carácter personal, el uso más frecuente será el de administrador, pero para una posible ampliación de funciones haremos las distinciones pertinentes.

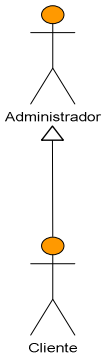


Figura 3.1 Diagramas organizativo.

Catálogo de requisitos.

En este apartado lo que se hará será ampliar el catálogo de requisitos obtenido en el apartado
EVS 1.1 con nuevos requisitos que han ido apareciendo. No se hará las dependencias con otros proyectos ya que no existen. Los requisitos ampliados pueden verse en la Tabla 3.2.
Catálogo de usuarios.
La Tabla 3.3 muestra el catálogo de usuarios obtenido para nuestro sistema. No obstante, pueden existir varios usuarios de cada tipo.

Tarea EVS 1.3: Especificación del alcance del EVS.

Catálogo de objetivos. Igual que la tarea EVS 1.1.
Catálogo de usuarios.

Igual que la tarea EVS 1.2.

Dependencia entre tareas.
En la Tabla 3.4 se deja claro que existen dependencias entre tareas. Por ejemplo, toda tarea depende de la captura de requisitos, ya que sin ésta, nada realizado del proyecto tendría sentido. Lo mismo ocurre con los diagramas y su documentación en Métrica 3, en la que cada diagrama se realiza en alguno (o algunos) de los procesos de esta metodología.

Nº	Función	Prioridad	Tipo	Descripción
6	Sencillez	Media	No funcional	El producto final deberá ser lo sencillo posible de manejar, para una mayor comprensión y rapidez
7	Flexible al ampliar	Alta	No funcional	El producto debe ser fácilmente ampliable
8	Mantenimiento	Alta	No funcional	El sistema debe ser de fácil mantenimiento una vez implantado
9	Robustez	Media/ Alta	No funcional	El sistema debe ser lo mas robusto posible en situaciones anómalas
10	Seguridad	Alta	Funcional	Tendrá control total del sistema.
11	Privilegios del administrador	Alta	Funcional	Tendra control total del sistema
12	Privilegios del Cliente	Media	Funcional	Acceso mas limitado, sólo tendrá acceso a ciertos datos, y a la compra de productos

Tabla 3.2. Catálogo de requisitos ampliado.

Usuario	Descripcion
Administrador	Control de todo el sistema, realiza la mismas funciones que los clientes y aparte controla toda la gestión
Cliente	Control limitado solo a la modificación de sus datos, y al pedido ante de realizarlo.

Tabla 3.3. Catálogo de usuarios.

3.1.2. ACTIVIDAD EVS 2: ESTUDIO DE LA SITUACIÓN ACTUAL

El estudio de la situación actual indica que hay bastantes sistemas existentes que trabaja con una idea parecida a nuestro sistema de información, pero aplicado a otro tipo de productos, en concreto con nuestro tipo de producto hay muy pocos, y con nuestra tecnología aun menos debido a la complejidad en un principio para desarrollar este tipo de sistemas de información, ya que hay otros hechos que se pueden reutilizar, y con un coste inferior. Creemos que nuestra aplicación va a permitir una mejora a la hora de la reutilización del código en proyectos futuros y con ello la reducción del coste de tiempo a la hora de mejora el sistema de información.

3.1.3. ACTIVIDAD EVS 3: DEFINICIÓN DE REQUISITOS DEL SISTEMA

En esta actividad se especifican los requisitos generales, mediante sesiones de trabajo. Una vez finalizadas, se detallan los requisitos y sus prioridades.

Tarea EVS 3.1: Identificación de las directrices y técnicas de gestión.

Catálogo de normas.

Información sobre estándares y procedimientos para proponer una solución:

- Políticas técnicas:
- Gestión de proyectos.
- Seguimiento periódico, revisión por el solicitante, aprobación final al acabar (establecer fecha).
- Desarrollo de sistemas.
    1. Uso de métrica 3.
    2. Base de datos (PostgreSql)
    3. Diagramas en StarUML, MagicDraw
    4. Programación en Java J2EE (JSF, Spring e Hibernate)

- Arquitectura de sistemas. La arquitectura del sistema de información será centralizada, donde habrá un servidor web central, desde donde se podrá conectar cualquier usuario a través de web para poder realizar los pedidos que necesiten.

- Política de seguridad: Validación de usuarios por su clave. El sistema se centra principalmente en la validación de datos, ya que será la piedra angular de una buena

seguridad y evitara errores en nuestra base de datos. Respecto a la disponibilidad del sistema, siempre estará disponible al usuario.

- Directrices de planificación: En principio el proyecto seguirá un modelo de proceso lineal.

Tarea EVS 3.2: Identificación de requisitos.

Las sesiones de trabajo se dividen en tres zonas bien diferenciadas:

- Zona de mantenimiento: aquí se harán todas las revisiones, modificaciones e introducciones de los datos del sistema. Solo el administrador podrá acceder a esta zona.

- Zona de tienda: aquí podrá acceder cualquier tipo de usuario, donde podrá realizar las compras de productos para la realización de sus pedidos, también podrá consultar sus datos y podrá modificarlo así como la dirección de envio y las observaciones que considere pertinente

Se va a añadir algunos requisitos con respecto a los ya expuestos anteriormente, como podemos observar en la Tabla 3.5.

Tarea EVS 3.3: Catalogación de requisitos.

Nº	Función	Prioridad	Tipo	Descripción
12	Sistema Operativo Windows o Linux o cualquiera que utiliza Apache y Java	Alta	No funcional	Sistema Operativo a utilizar para trabajar con el sistema
13	Trabajar a través de la red	Alta	No funcional	El sistema al ser a través de la Web puede funcionar tanto en Internet como en una red local.

Tabla 3.5. Catálogo de requisitos.

3.1.4. ACTIVIDAD EVS 4: ESTUDIO DE ALTERNATIVAS DE SOLUCIÓN

En esta actividad se proponen las distintas soluciones que pueden resolver los requisitos especificados anteriormente.

Tarea EVS 4.1: Preselección de alternativas de solución.
Descomposición inicial del sistema en subsistemas.

El sistema se ha dividido en cuatro subsistemas, tal y como se observa en la Figura 3.2 donde cada uno tendrá tanto funciones propias como funciones relacionadas entre sí.
Los diferentes subsistemas son:

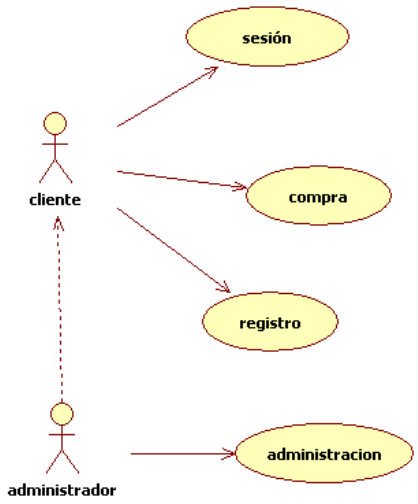


Figura 3.2. Descomposición del sistema en subsistemas.

- Administración: estará recogida la gestión de productos y de pedidos, información referente a los productos y sus modificaciones al igual que los pedidos.

- Sesión: aquí vendrá recogido el acceso al sistema

- Compra: el administrador es un cliente mas dentro de la aplicación, solo que tiene permiso para realizar operaciones administrativas, por lo que puede realizar compras.

- Registro. El usuario antes de poder realizar la compra, deberá estar registrado en la aplicación.

Con esta división se pretende facilitar y simplificar la labor tanto del análisis como de la realización de todo el sistema, aislando las funciones según su contenido.

Alternativas de solución a estudiar

Sólo se va a considerar una única solución, siendo un software a medida según las especificaciones del solicitante; por tanto, se rechaza cualquier posible solución del tipo de adquisición de software, ya sea estándar del mercado, desarrollos a medida o solución mixta. En conclusión sobre alternativas válidas, dispondremos únicamente de la solución desarrollada en un entorno cerrado.

Tarea EVS 4.2 Descripción de alternativas de solución.

Catálogo de requisitos

El catálogo queda igual en su última modificación, en el apartado EVS 3.3. Alternativas de solución a estudiar.

- Catálogo de requisitos (cobertura): La cobertura será total, ya que la entrevista fue personal y, por tanto, se van a cubrir todos los requisitos planteados por el solicitante.

- Subsistemas: el mismo que en el apartado EVS 4.1.

- Entorno tecnológico y de comunicaciones: se deberá tener un ordenador personal que contenga un navegador web para el cliente, para el servidor será necesario un ordenador personal que soporte *Apache TomCat* y *Java*.

- Modelo de negocio: Ver apéndice A. Diagrama de casos de uso.

3.1.5 ACTIVIDAD EVS 5: VALORACIÓN DE LAS ALTERNATIVAS

Tarea EVS 5.1: Estudio de la inversión.

- **Impacto en la organización**:

La implantación del sistema supondrá:

- Organización de la información mejorada.
- Más facilidad de acceso a la información.
- Condiciones de trabajo mejoradas.
- Satisfacción del cliente.

Tarea EVS 5.2: Estudio de los riesgos.

- **Valoración de riesgos**:
  - Desconocimiento del entorno: Siempre puede haber errores en la captura de requisitos, ya que puede haber casos en los que el evento difiera un poco de lo visto anteriormente.

Tarea EVS 5.3: Planificación de alternativas.

- **Plan de trabajo**.

  Igual que la tarea EVS 1.3.

3.2 ANÁLISIS DEL SISTEMA DE INFORMACIÓN (ASI)

En este proceso se realiza una especificación detallada del sistema de información que satisfaga las necesidades de los usuarios y sirva de base para su posterior diseño.

Métrica 3 es una metodología pensada tanto para desarrollo estructurado como orientado a objetos, existen actividades que no conciernen al análisis de este sistema de información, por lo que no se realizan.

A continuación, en las diferentes secciones, se especifican las actividades a realizar y sus tareas correspondientes.

3.2.1. ACTIVIDAD ASI 1: DEFINICIÓN DEL SISTEMA

Esta actividad tiene como objetivo detallar una descripción del sistema, delimitando su alcance, estableciendo las interfaces con otros sistemas e identificando a los usuarios representativos. Puede suceder que esta actividad se haya realizado en parte en el anterior proceso (EVS).

Tarea ASI 1.1: Determinación del alcance del sistema.

Catálogo de requisitos.

Igual que la tarea EVS 1.2.

Glosario.

El glosario servirá para conseguir una mayor precisión en la especificación del sistema de información.

- Administrador. Usuario que administra el sistemas, además de ser un cliente mas de la aplicación
- Cliente. Usuario que se conecta al sistema para realizar la compra de productos o para modificar sus datos.
- Manual de Usuario. Manual de manejo del programa.
- Clave de usuario. Clave de acceso al sistema
- Usuario. Nombre del cliente con el cual se le reconoce en el sistema ,no tiene porque coincidir con su nombre
- Cerrar sesión. Desconectarse de la sesión iniciada.
- Registrarse. Indicar los datos personales y datos de conexión para poder realizar las compras en la tienda.
- Iniciar sesión. Conectarse al sistema con el nombre de usuario y su clave.
- Confirmar pedido. Realizar el pedido para su confirmación una vez comprado todo lo necesario

Tarea ASI 1.2: Identificación del entorno tecnológico.

Catálogo de Requisitos.

Igual que la tarea ASI 1.1.

Descripción general del entorno tecnológico del sistema.

El cliente u ordenadores que se conecten al servidor deberán contar al menos con un ordenador personal, no importa el sistema operativo, solo que tenga un navegador Web. Para el lado del servidor haría falta un ordenador que soporte Java y el servidor *Apache Tomcat*.

Tarea ASI 1.3: Especificación de estándares y normas.

Catálogo de normas.

Se revisan las directrices expuestas en la tarea EVS 3.1.

- Políticas técnicas.

- Gestión de proyectos. Se estima que la fecha de entrega del proyecto sea a partir de finales de Febrero del 2009.

- Desarrollo de sistemas.

- La metodología empleada para el desarrollo del sistema de información ha sido Métrica versión 3. Los motivos para haber usado esta metodología son los estudios que la definen como una buena metodología a seguir y la integración al desarrollo orientado a objetos con respecto a su anterior versión. Los objetivos de esta metodología son:

- Proporcionar o definir sistemas de información para conseguir los objetivos propuestos por la organización.

- Dotar de un software que satisfaga las necesidades de la organización dando mayor importancia al análisis de requisitos.

- Mejorar la productividad de los departamentos SI / TIC, permitiendo una mayor capacidad de adaptación a los cambios y teniendo en cuenta la reutilización en la medida de lo posible.

- Facilitar la operación, mantenimiento y uso del producto software obtenido.

- Para el desarrollo de la base de datos relacional se usara Postgresql.

- El entorno de programación será MyEclipse. Es un entorno de desarrollo rápido de software Java. Su principal uso es el de desarrollo de aplicaciones visuales cliente-servidor, multicapas y web.

- Arquitectura de sistemas. La arquitectura del sistema será centralizada.

- Política de seguridad. Para acceder al sistema se deberá usar una clave de acceso, para prohibir la utilización de zonas del sistema a usuarios no autorizados.

- La seguridad del sistema vendrá dada principalmente por la validación de datos. Se evitará almacenar información errónea, así como inútil o incompleta.

Tarea ASI 1.4: Identificación de los usuarios participantes y finales.
Catálogo de usuarios.

Los usuarios quedan de la misma forma que la tarea EVS 1.2.
Usuario administrador y cliente: como usuario administrador identifico al administrador del sistema, el cual ha sido el interlocutor en la captura de requisitos. Como cliente serán el propio administrador y los clientes. Lo podemos ver en la Figura 3.3 donde se representa la jerarquía de privilegios donde el administrador es quien más privilegios posee y el cliente el que menos.

Figura 3.3. Usuarios del sistema

3.2.2 ACTIVIDAD ASI 2: ESTABLECIMIENTOS DE REQUISITOS

Tarea ASI 2.1: Obtención de requisitos.

Catálogo de requisitos.

Igual que la tarea ASI 1.1.

Modelo de casos de uso.

Ver Apéndice A. Diagramas de casos de uso.

Tarea ASI 2.2: Especificación de casos de uso.

Catálogo de requisitos.

Igual que la tarea ASI 2.1.

Modelo de casos de uso.

Ver Apéndice A. Diagramas de casos de uso.

Especificación de casos de uso.

1. Cliente
    1.1 Iniciar sesión
        i. Escenario normal
            1. El usuario quiere iniciar sesión
            2. El usuario introduce los datos necesarios.
            3. El usuario acepta los datos introducidos.
            4. Se muestra el menú principal del sistema.

        ii. Escenario de excepción
            1. El usuario quiere iniciar sesión.
            2. El usuario introduce los datos.
            3. El usuario acepta los datos introducidos.
            4. Se muestra un mensaje de error por datos incorrectos.

    1.2 Cerrar sesión
        i. Escenario normal
            1. El usuario quiere cerrar una sesión.
            2. El usuario solicita cerrar su sesión.
            3. Se cierra la sesión y se deja la página del principio.

1.3 Comprar producto

i. Escenario normal

1. El usuario indica la cantidad que quiere comprar del producto dentro de la tabla.
2. El usuario marca en el carrito de la compra.
3. El producto se añade a su lista de compra.

1.4 Buscar producto

i. Escenario normal

1. El usuario indica el producto a buscar dentro de las opciones de buscar.
2. Aparece en el panel central de la aplicación de todos los productos buscados con ese nombre.
3. Se puede seleccionar cualquiera de los productos visualizados para comprar.

1.5 Eliminar producto

i. Escenario normal

1. El usuario indica el producto que desea eliminar de la lista de compra. Marcando sobre un icono de eliminar.
2. El producto es eliminado de la lista de productos.

1.6 Disminuir o aumentar cantidad de un producto.

i. Escenario normal

1. El usuario marca sobre los iconos de incrementar o disminuir la cantidad perteneciente a un producto.
2. La cantidad se ve aumentada o disminuida entre los valores de 0 a 100.

1.7 Registrarse

i. Escenario normal

1. El usuario marca sobre el icono registrarse.
2. Aparece una nueva página de registro.
3. El usuario rellena los datos de registro, y le da a enviar datos.

4. Lo dato enviado son almacenados y se envía a una nueva pagina donde se le indica que esta registrado.

ii. Escenario excepción.

1. El usuario marca sobre el icono registrarse.
2. Aparece una nueva página de registro.
3. El usuario rellena los datos de registro, se envía los datos.
4. Se muestra un mensaje de error por datos incorrectos.

1.8 Confirmar pedido

i. Escenario normal

1. El usuario marca sobre el texto confirmar pedido.
2. Aparece una nueva página donde ve su pedido, pudiendo modificar hora de pedido, y las observaciones que considere pertinente.
3. Enviar pedido. El pedido es almacenado en la base de datos para su posterior procesado.

ii. Escenario excepción.

1. El usuario marca sobre el texto confirmar pedido.
2. El sistema genera una nueva pagina indicando que nos esta registrado o que no ha insertado sus datos de usuario y clave.
3. Vuelve otra vez a la página central.

1.9 Modificar datos personales

i. Escenario normal

1. El usuario se registra insertando sus datos de usuario y clave de usuario.
2. El usuario marca sobre la etiqueta modificar.
3. Aparece una nueva pagina con sus datos personales
4. Modifica sus datos y le da a confirmar.
5. Los datos son almacenados en la base de datos.

ii. Escenario de excepción

1. El usuario se registra insertando sus datos de usuario y clave de usuario.

2. El usuario marca sobre la etiqueta modificar.
3. Aparece una nueva pagina con sus datos personales
4. Se genera un mensaje de error cuando se inserta los datos.

1.10 Seleccionar productos

i. Escenario normal

1. El usuario marca sobre una de las posibles opciones para visualizar productos (frutas, verduras, ofertas).
2. Los productos son visualizados en la Web para su posible selección.
3. Si los productos superan la cantidad de productos que se puede mostrar en la página, entonces se paginaría para poder mostrarlos todos.

2. Administración

2.1 Alta de producto

i. Escenario normal

1. El administrador accede a la página de alta de producto.
2. Rellena los datos del producto.
3. Se envía los datos para sus almacenaje y procesado.

ii. Escenario de excepción

1. El administrador accede a la página de alta de producto.
2. Rellena los datos del producto.
3. Se envía un mensaje de error por falta de datos o datos incorrectos.

2.2 Modificación de producto

i. Escenario normal

1. El administrador accede a la página de modificación de producto.
2. Indica el nombre del producto ha modificar.
3. Se visualiza todos los datos de los productos con ese nombre.
4. Se modifica los datos del producto y se realiza su confirmación

ii. Escenario de excepción

1. El administrador accede a la página de modificación de producto.
2. Indica el nombre del producto ha modificar.
3. Se visualiza todos los datos de los productos con ese nombre.
4. Se envía los datos
5. Se produce un mensaje de error, debido a la falta de datos o datos incorrectos.

2.3 Modificación de pedido

i. Escenario normal

1. El administrador accede a la página de consulta de pedido.
2. Lista pedido procesados o sin procesar.
3. Consulta los pedidos o los modifica.

ii. Escenario de excepción

1. El administrador accede a la página de consulta de pedido.
2. Lista pedido procesados o sin procesar.
3. Consulta los pedidos o los modifica.
4. Se produce un mensaje de error, debido a la falta de datos o datos incorrectos.

Tarea ASI 2.3: Análisis de requisitos.

Catálogo de Requisitos.

Igual que la tarea ASI 2.2.

Modelo de casos de uso.

Ver Apéndice A. Diagramas de casos de uso.

Especificación de casos de uso.

Igual que la tarea ASI 2.2.

Tarea ASI 2.4: Validación de requisitos.

Todos los requisitos especificados y los casos de uso se han confirmado que son válidos, completos y consistentes.

3.2.3. ACTIVIDAD ASI 3: IDENTIFICACIÓN DE SUBSISTEMAS DE ANÁLISIS

En esta actividad se lleva a cabo la división del sistema en subsistemas. Esta actividad se desarrolla en paralelo con otras actividades, por lo que se realiza una realimentación y ajuste continuo con respecto a la definición de subsistemas, interfaces y dependencias.

Tarea ASI 3.1: Determinación de subsistemas de análisis.

Descripción de subsistemas de análisis.

Igual que la tarea EVS 4.1.

Descripción de interfaces entre subsistemas.

Con la organización expuesta en la Tabla 3.6, todos los requisitos están resueltos, desglosándose en casos de uso. También se ha asegurado de que no se queda nada sin realizar y de que no exista redundancia.

Diagrama de paquetes.

Las dependencias entre subsistemas se muestran en la Figura 3.5. En la Tabla 3.7 se muestran las mismas dependencias pero de una forma más clara.

Tarea ASI 3.2: Integración de subsistemas de análisis.

Descripción de subsistemas de análisis.

Igual que la tarea EVS 4.1.

Descripción de interfaces entre subsistemas.

Igual que la tarea EVS 3.1.

3.2.4 ACTIVIDAD ASI 4: ANÁLISIS DE CASOS DE USO

Tarea ASI 4.1: Identificación de clases asociadas a cada caso de uso.

Modelo de clases de análisis.

En esta tarea se extraen los objetos que se implementarán mediante clases, ya que se usa programación orientada a objetos. Estos objetos son extraídos mediante la realización de casos de uso y la relación existente entre ellos.

Ver Apéndice A. Diagramas de casos de uso.

Ver Apéndice A. Diagramas de clases del dominio del problema.

Subsistema	Requisitos(Caso de uso)
Inicio	Iniciar sesión Cerrar sesión Salir del programas
Tienda	Compra producto Modificar cantidad de producto Eliminar producto Confirmar pedido Registrarse Iniciar sesión Ver producto (frutas, verduras, etc..)
Producto	Alta producto Modificación producto Consulta producto
Compra	Lista de pedidos procesado Lista de pedidos sin procesar
Información	Temporada de productos Acerca de Contacto
Cliente	Registrar cliente Iniciar sesión cliente Modificar cliente

Tabla 3.6 Descripción de interfaces entre subsistemas

Tarea ASI 4.2: Descripción de la interacción de objetos.

Análisis de la realización de los casos de uso.

Una vez se obtienen los objetos, éstos cooperan entre ellos para llevar a buen término un caso de uso. Para la representación de dicha cooperación usamos la técnica de interacción de objetos.

Ver Apéndice A. Diagramas de interacción de objetos.

3.2.5 ACTIVIDAD ASI 5: ANÁLISIS DE CLASES

Esta actividad consiste en describir las clases que han sido identificadas anteriormente, indicando las responsabilidades, atributos y relaciones entre ellas.

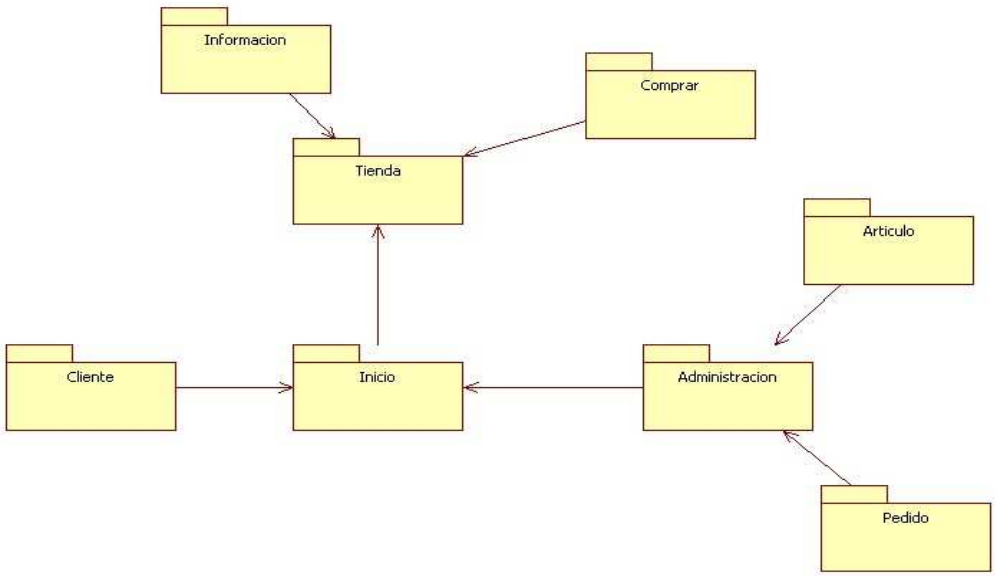


Figura 3.5 Diagrama de dependencias de paquetes

Tarea ASI 5.1: Identificación de responsabilidades y atributos.

La identificación de responsabilidades define la funcionalidad de cada clase, para así identificar las operaciones que les va a pertenecer, junto con sus atributos correspondientes. Los atributos son las propiedades que tendrán cada clase y se identifican en base a las responsabilidades de la clase. Los tipos de dichos atributos deben de conocerse en el dominio.

Modelo de clases de análisis.

Ver Apéndice A. Diagrama de clases del dominio del problema.

Comportamiento de clase de análisis.

Ver Apéndice A. Diagrama de clases del dominio del problema.

Tarea ASI 5.2: Identificación de asociaciones y agregaciones.

En esta tarea se identifican las relaciones existentes entre las clases, como los mensajes que se envían entre ellas.

Modelo de clases de análisis.

Ver Apéndice A. Diagrama de clases del dominio del problema.

Subsistema	Depende de
Información	Tienda
Comprar	Tienda
Inicio	Tienda
Cliente	Tienda Inicio
Administración	Tienda Inicio
Articulo	Administración Inicio Tienda
Pedido	Administración Inicio Tienda

Tabla 3.7 Dependencias entre subsistemas.

3.2.6. ACTIVIDAD ASI 8: DEFINICIÓN DE INTERFACES DE USUARIO

En esta actividad se especifican las interfaces entre el sistema y el usuario, como son el formato, diálogos e informes. Se realiza un análisis de los procesos del sistema de información, en los que se requiere una interacción con el usuario, para así crear una interfaz que satisfaga los requisitos especificados.

Tarea ASI 8.1: Especificación de principios generales de interfaz.

Principios generales de la interfaz.

Se va a desglosar una serie de principios y directrices, tanto para la interfaz como para los documentos de impresión. Se deberán de cumplir en la medida de lo posible para que el diseño de la interfaz resulte adecuado al cliente solicitado.

- Interfaz intuitiva y amigable. Las opciones disponibles deben estar visibles en todo momento y de fácil acceso, ya que las personas que pueden llegar a utilizar el sistema no tienen por qué ser usuarios expertos informáticos, todo se centrara en una pagina central, en la cual se podrá realizar todos los procesos necesario para poder realizar la compra de productos.

- Los formularios principales siempre aparecerán en su forma maximizada, para tener una visión total del sistema, aunque la configuración no tiene porque aparecer igual en todos los navegadores ya que pueden diferir unos de otros. Si el valor de la página es un tamaño mas reducido los navegadores incorporan un desplazador para poder desplazarse y visualizar la pagina. En el caso de que se presente un formulario que no requiera el total de la pantalla, se presentará en el centro de la pantalla para que se tenga fácil acceso.

- Los mensajes de error y confirmación aparecerán siempre de la misma forma, al lado del campo donde se ha producido el error.

La interfaz utilizada se basa en código *Html* y *JavaScript* que es generado por el servidor, con lo cual es un código genérico que puede correr en cualquier cliente que tenga un Navegador Web, todo los proceso que indique el cliente se enviara al servidor donde se procesaran y devolverán un resultado como código *Html*.
La página tienda se ha dividido la página principal en tres zonas delimitadas.

- Zona izquierda: En esta zona se indica los productos que se quieren visualizar en la página central, pudiendo seleccionar entre las distintas opciones e incluso buscar un producto en concreto.

- Zona central: Aquí se visualiza todos los productos que son indicados en la zona izquierda, pudiendo comprar dicho producto, incrementar la cantidad a comprar o seleccionar un producto para obtener una información más detallada.

- Zona derecha: Aquí se muestra los datos para registrarse o si esta registrado poder modificar sus datos, confirmar el pedido o realizar acciones administrativas si es administrador.

Tarea ASI 8.3: Especificación de formatos individuales de la interfaz de pantalla.

Especificación de interfaz de usuario:

Los formatos individuales y catálogo de controles y elementos de diseño de la interfaz de pantalla se pueden ver en el manual de usuario en el capítulo 4.

Tarea ASI 8.4: Especificación del comportamiento dinámico de la interfaz.

Esta tarea permite contemplar todos los casos y acciones posibles. Para ello, recurrimos a los diagramas de transición de estados para representar todo el comportamiento de forma clara e inequívoca de las ventanas del sistema.

Ver apéndice A. Diagrama de transición de estados.

3.2.7 ACTIVIDAD ASI 9: ANÁLISIS DE CONSISTENCIA Y ESPECIFICACIÓN DE REQUISITOS.

Tarea ASI 9.1: Verificación de los modelos.

En esta tarea se comprueba la calidad formal de los diferentes modelos seguidos durante el análisis. A continuación verificamos cada uno de los modelos seguidos.

- Interfaz de usuario.

  Mediante la funcionalidad obtenida y la asignación de perfiles para el acceso al sistema, los requisitos establecidos han quedado satisfechos. Para este modelo se ha reflejado la relación entre el modelo de casos de uso, el modelo de clases del dominio del problema y los diagramas de interacción de objetos para verificar la consistencia y correspondencia entre ellos.

  Para la especificación de la interacción entre el sistema y el usuario final se ha usado los formularios y diálogos.

- Modelo y especificación de casos de uso.

  La división del sistema en subsistema resulta correcta por las siguientes razones: semejanza de requisitos, no heterogeneidad y prioridad.

  Para cada caso de uso se ha tenido en cuenta varios escenarios, el normal, que es el comportamiento lógico del sistema para la interacción de éste con el usuario, y uno o varios escenarios de excepción según la situación lo requiera. Estos escenarios de excepción indican el comportamiento anómalo de la interacción del sistema y el usuario final.

- Modelo y comportamiento de clases de análisis.

El modelo se ha dividido en tres partes bien diferenciadas:

- Dominio del problema. El modelo de negocio del sistema.
- Gestión de datos. El paso a tablas de la base de datos con respecto al dominio del problema.
- Interfaz de usuario. Diagrama de composición de la interfaz y las relaciones existentes entre las ventanas que lo componen.

Se ha comprobado que se cumplen todos los requisitos funcionales y la relación entre los subsistemas, por lo que la calidad formal exigida en esta tarea se ve satisfecha.

Tarea ASI 9.2: Análisis de la consistencia entre modelos.

Para esta tarea nos aseguramos la consistencia entre modelos. Para ello, se irá desglosando los modelos y se comprueba de que no hay ambigüedad ni información duplicada.

- Modelo de clases

  La comprobación de la consistencia entre clases se ha hecho de la siguiente forma. Nos hemos asegurado de que todas las clases que se envían mensajes, llegan a un objetivo real, comprobando que la clase receptora es la adecuada.

  Para el caso de la solicitud de datos por parte del usuario, se comprueba que verdaderamente se puede enviar dicha petición.

- Análisis de la realización de los casos de uso

  Para esta relación nos fijamos en el diagrama de interacción de objetos y nos aseguramos de que la navegación en la interfaz corresponde con los mensajes de dichos diagramas.

- Análisis de los diagramas de transición de estados

  Para la realización de los diagramas de transición de estados, nos hemos basado en una extensión UML denominado WAE-UML (Web Application Extensión for UML, Extensión de Aplicación Web para UML).

- Análisis de los diagramas de interacción.

  Para la realización de estos diagramas nos hemos basado en la relación del interfaz del cliente con las clases que se ejecutan y se relacionan con la interfaz.

Tarea ASI 9.3: Validación del modelo.

La validación del modelo ha sido aceptada al haberlo comprobado en la tarea anterior. Solo faltaría la comprobación por parte de los usuarios expertos. También se ha seguido un análisis de forma que se pueda ampliar y/o hacer cambios en el futuro si así la situación lo requiere.

Tarea ASI 9.4: Elaboración de la especificación de requisitos software (ERS).

Esta tarea recoge la información necesaria para la aprobación final de las actividades realizadas en el análisis del sistema de información.

- Introducción. Igual que la tarea EVS 1.1 (Descripción general del sistema).
- Ámbito y alcance. Igual que la tarea EVS 1.2 (Contexto del sistema. Estructura organizativa).
- Participantes. Igual que en la tarea ASI 1.4 (Catálogo de usuarios)
- Requisitos del sistema de información. Igual que la tarea ASI 1.1 (Catálogo de requisitos).
- Visión general del sistema de información. Igual que la tarea ASI 8.2 (Catálogo de perfiles de usuario).
- Referencias de los productos a entregar. Los productos a entregar son los modelos que se han ido realizando a lo largo de todo el análisis.
    - Diagramas de casos de uso.
    - Diagramas de clases.
    - Diagramas de estados.
    - Diagrama de interacción
    - Estudio de viabilidad del sistema
    - Análisis del sistema de información

3.3 DISEÑO DEL SISTEMA DE INFORMACIÓN (DSI)

En este proceso se define la arquitectura del sistema, así como su entorno tecnológico. También se especifica de forma detallada los componentes del sistema de información. A continuación se irán haciendo las distintas actividades para conseguir los objetivos de dicho proceso.

3.3.1. ACTIVIDAD DSI 1: DEFINICIÓN DE LA ARQUITECTURA DEL SISTEMA

En esta actividad se define la arquitectura general del sistema de información, junto con la especificación de las particiones físicas del mismo, la descomposición lógica en subsistemas y la ubicación de dichos subsistemas en cada partición. También se especifica la infraestructura tecnológica que dará soporte al sistema de información.

Tarea DSI 1.1: Definición de niveles de arquitectura.

Para definir los niveles de arquitectura se definen las principales particiones del sistema, las cuales se representan en diferentes niveles o capas de la arquitectura:

- Capa de presentación: interfaz de usuario, seria la aplicación que ve el usuario, donde se manipulan los datos y se gestiona la comunicación entre el usuario y el programa, para esta capa utilizamos *JSF* (Java Server Faces).

- Capa de lógica: es la parte central donde se desarrolla en núcleo del programa, hace de comunicación entre la capa de presentación y la de persistencia, para esta capa utilizamos *Spring*.

- Capa de persistencia: su función es realizar la persistencia de nuestro programa, comunicándose con la base de datos, para almacenar o recuperar información. utilizamos *Hibernate* para desarrollar esta capa.

Esta tres capas serian las que componen nuestra aplicación, a la hora de implementarla, por el lado del servidor necesitaríamos un servidor web (Apache)

que contenga un contenedor servlet (TomCat) y una base de datos donde nos conectaríamos para almacenar y recuperar los datos (Figura 3.9 y 3.10).

Por el lado del cliente solo nos bastaría con un navegador web que se conecte a la aplicación.

Tarea DSI 1.2: Identificación de requisitos de diseño y construcción.

En la siguiente lista se muestran las características que pueden condicionar el diseño y construcción del sistema de información.

- Un ordenador cliente con un navegador web.
- Un servidor web (Apache) con contenedor Servlet (*Tomcat*).
- Una Base de Datos relacional (*Oracle, MySql, PostGresql*)

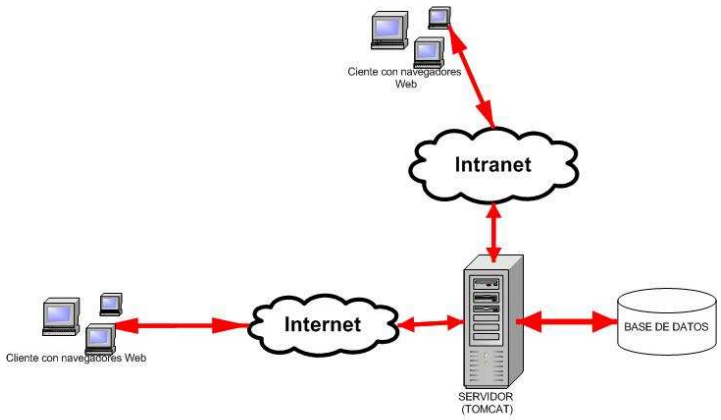


Figura 3.9 Diagrama de conexión

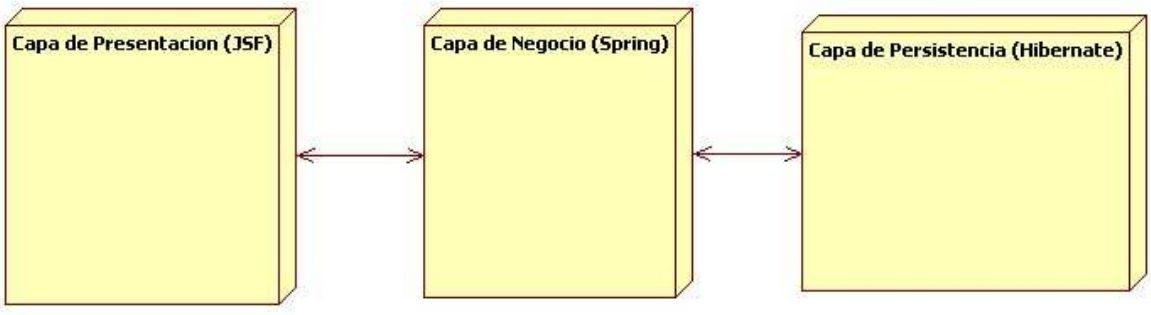


Figura 3.10 Diagrama de capas

Catálogo de Requisitos.

A los requisitos expuestos en la tarea ASI 1.1 se le añaden los requisitos identificados en esta tarea mostrados en la Tabla 3.11.

Nº	Funcion	Prioridad	Tipo	Descripción
13	Ordenador Cliente	Alta	No funcional	Ordenador personal con navegador Web.
14	Servidor Web	Alta	No funcional	El servidor al ser a través de la web puede funcionar tanto en Internet como en una red local
15	Base de datos	Alta	No funcional	Base de datos relacional

Tabla 3.11. El catálogo de requisitos

Tarea DSI 1.3: Especificación de excepciones.

Esta tarea se encarga de catalogar y describir situaciones anómalas en el sistema de información a partir de los niveles de arquitectura expuestos anteriormente. Las excepciones las clasificaremos en dos partes, excepciones de sistema y excepciones de usuario.

- Excepciones del sistema. Son causas anómalas ajenas al usuario que está manipulando la información.

- Error de conexión. Se produce cuando se intenta acceder a la base de datos y ésta no está disponible. Esta excepción afecta a todo el sistema. Respuesta del sistema: “Error de conexión. Intente la operación en otro momento.”.

- Error de conexión a la aplicación. Se produce cuando un cliente se quiere conectar a través de la red y la aplicación no esta activa, con lo que el navegador da error de conexión.

- Excepciones de usuario. Son excepciones producidas por manipulación incorrecta del usuario con el sistema de información.

- Excepción de validación de datos. Se produce cuando el usuario intenta introducir datos erróneos en el sistema. Puede afectar a cualquier zona donde se puedan introducir datos.

Tarea DSI 1.4: Especificación de estándares y normas de diseño y construcción.

En esta tarea se actualiza el catálogo de normas establecido en la tarea ASI 1.3. Se actualizará teniendo en cuenta estándares técnicos y de nomenclatura, normas y recomendaciones que puedan afectar al diseño o construcción del sistema de información.

Se añadirá al catálogo de normas, los estándares y normativas de la instalación, respecto al aspecto externo. La instalación se hará basándose en los niveles de arquitectura expuestos en la tarea DSI 1.1.

Tarea DSI 1.6: Especificación del entorno tecnológico.

En esta tarea se describen los diferentes elementos de la infraestructura que da soporte al sistema de información. Estos elementos se van a agrupar en:

- Hardware
    - Cliente- A nivel de cliente se necesitara un ordenador con 128 MB.
    - Servidor- este ordenador debe poseer 1gb de RAM mínimo y un disco duro bastante grande para almacenar la información de los pedidos.
- Software
    - Cliente- cualquier Sistema Operativo que posea un navegador Web.
    - Servidor- cualquier Sistema Operativo que tenga servidor Web (Apache) y contenedor Servlet (Tomcat).
- Comunicaciones.

  Al ser una aplicación cliente-servidor que funciona a través de la Web, necesitaríamos una línea media ya que las paginas tiene un de peso medio de (315KB), en una red local no daría muchos problemas al tener un ancho de banda bastante grande para esta aplicación, en cambio para Internet necesitaremos un

ancho de banda bastante aceptable para soportar la carga de los usuarios que se conecten a nuestra aplicación a través de Internet.

Tarea DSI 1.7: Especificación de requisitos de operación y seguridad.

Para esta tarea, se especifican políticas de seguridad teniendo en cuenta la arquitectura descrita durante esta tarea (DSI 1). Con estas especificaciones se conseguirá garantizar protección en el sistema a alteraciones, consultas indebidas de datos y minimizar el riesgo de pérdida de los mismos en la medida de lo posible.

- Acceso al sistema. Los usuarios pueden acceder al sistema para realizar las compras de los productos, pero no podrán confirmar dicha compra hasta no haberse registrado como usuario de la aplicación.
- Administración del sistema. Solo el usuario Administrador podrá acceder a ciertas partes del sistema, mediante la inserción del nombre administrador y de la clave correspondiente, una vez dentro del sistema se podrá modificar ciertos datos de la aplicación.

- Copia de seguridad. Como la copia se almacena en una base de datos, tendremos que guardar los datos desde la base de datos cada cierto tiempo, para no perder la información.

3.3.2 ACTIVIDAD DSI 3: DISEÑO DE CASOS DE USO REALES

El objetivo de esta actividad es especificar el comportamiento del sistema para cada caso de uso. Para ello, se utilizan los objetos y subsistemas que interactúan en un caso de uso, determinando las operaciones de los distintos subsistemas de diseño.

Tarea DSI 3.1: Identificación de clases asociadas a un caso de uso.

En esta tarea se relaciona los casos de uso con las clases especificadas en el dominio del problema que hacen llegar a buen término un caso de uso. Esta relación se puede observar en la Tabla 3.12.

Requisito(Caso de uso)	Clases
Inicio sesión Cerrar sesión	UsuarioBean
Comprar Confirmar pedido	ClienteBean UsuarioBean
Registrarse	ClienteBean
Visualizar producto	ProductoListBean ProductoBean
Administración de pedidos Administración de productos	Pedidomodificación ProductoBean
Listado de pedido	PedidoUsuarioBean
Visualizar paneles administrador	admin

Tabla 3.12 Clase asociada a un caso de uso

Tarea DSI 3.2: Diseño de la realización de los casos de uso.

En esta tarea se especifica cómo se relacionan entre sí las clases para llevar a buen término un caso de uso.
Ver Apéndice A. Diagramas de interacción de objetos.

Tarea DSI 3.3: Revisión de la interfaz de usuario.

En esta tarea se revisa los elementos utilizados en la interfaz de usuario. Todos los elementos que forman la interfaz gráfica ya fueron analizados y comentados en el ASI y éstos han sido revisados.

3.3.3 ACTIVIDAD DSI 4: DISEÑO DE CLASES

En esta actividad se transforma el modelo de clases lógico en un modelo de clases de diseño.
Dicho modelo recoge la especificación detallada de cada una de las clases, es decir, sus atributos, operaciones, métodos, y el diseño preciso de las relaciones establecidas entre ellas, bien sean de agregación, asociación o jerarquía.

Tarea DSI 4.1: Identificación de clases adicionales.

Se han revisado las clases y no se han encontrado ninguna adicional.

Tarea DSI 4.2: Diseño de asociaciones y agregaciones.

Las asociaciones y agregaciones existentes se pueden observar gráficamente en diagrama de clases del dominio del problema. En dicho diagrama se observa las diferentes multiplicidades, así como la posible navegación entre clases, donde existen clases que no pueden ver la relación mientras que al contrario sí.

Tarea DSI 4.3: Identificación de atributos de las clases.

Para observar los atributos identificados hay que observar el diagrama de clases realizado, donde se aprecian todos los atributos y el tipo que poseen.
Ver apéndice A. Diagrama de clases del dominio del problema.

Tarea DSI 4.4: Identificación de las operaciones de las clases.

En esta tarea se detallan las operaciones que posee cada clase del modelo realizado. En el diagrama se distingue tantos las operaciones dentro de una clase como las interfaces de operaciones comunes.
Ver apéndice A. Diagrama de clases del dominio del problema.

Tarea DSI 4.5: Diseño de la jerarquía.

No se va a realizar ésta tarea ya que no se ha encontrado ninguna relación de herencia.

Tarea DSI 4.6: Descripción de métodos de las operaciones.

A continuación describiremos mediante el leguaje natural los métodos que se usan para detallar como se realiza cada una de las operaciones:

- En ProductoBean:
    - Salvarproducto: Salva el producto en la base de datos.
    - productoActualizar: Actualiza un producto de la base de datos.

- En ClienteBean
    - ComprobarUsuario: Comprueba los datos de un usuario si esta registrado en la base de datos.
    - Limpiar: Limpia los datos de los formularios donde se registra el usuario.
    - Registrar: Almacena los datos del usuario que se ha registrado en la pagina web.
    - Mostrar: Muestra los datos del usuario registrado en la página web.
    - Update: Actualiza los datos del usuario registrado en la pagina web.

- En UsuarioBean
    - Logout: Termina la sesión del usuario en la pagina web.
    - Confirmarpedido: Realiza la confirmación de la lista de productos que el usuario ha realizado en la página web.
    - Realizarpedido: Confirma el pedido que el usuario ha realizado.
    - AnadirLinea: Añade un producto a la lista de productos comprados.
    - eliminarLinea: Elimina una línea de la lista de productos comprados.
    - roundnum: Redondea el precio de un producto.
    - setTotalproducto: Suma el total de todos los productos y lo añade al total del producto.

- ProductoListBean
    - Limpiar: Limpia los datos de todos los productos a visualizar.
    - VisualizarVerdura: Muestra los datos de verdura a visualizar en la página Web.
    - VisualizarFrutas: Muestra los datos de frutas a visualizar en la página Web.
    - VisualizarOferta: Muestra las ofertas de producto a visualizar en la página Web.
    - VisualizarOfertaFrutas: Muestra las ofertas de frutas a visualizar en la página Web.
    - VisualizarOfertaVerdura: Muestra las ofertas de verduras a visualizar en la página Web.
    - VisualizarBuscar: Muestra los productos que se ha indicado en el campo buscar.
    - ProductoDetalle: Muestra el producto con más información.

- getOferta: Indica si un producto esta en oferta

Tarea DSI 4.7: Especificación de necesidades de migración y carga inicial de datos

Al ser una base de datos relacional en el servidor, podríamos guardar los datos e exportarlo para poder ser utilizado por cualquier otra base de datos relacional e incluso por cualquier otro programa que pueda acceder a esos datos para su gestión.
La carga inicial de datos será todos los productos que almacenamos en nuestra base de datos, así como aquellas modificaciones que podrían realizarse sobre los productos (precio, cantidad, procedencia etc.).

3.3.4 ACTIVIDAD DSI 6: DISEÑO FÍSICO DE DATOS

En esta actividad defino la estructura física de los datos que utilizará el sistema, a partir del modelo de clases realizado anteriormente. Hay que comentar que, en el caso de diseño orientado a objetos, esta transformación del modelo físico se realiza a través de una serie de reglas que a continuación paso a comentarlas:

1. Relaciones de asociación y agregación

   - Muchos a muchos: A partir del modelo de clases las relaciones de este tipo las realizo de la forma siguiente, creo tres tablas, una por cada modelo con su llave primaria subrayada y una tercera tabla que englobará las llaves primarias de cada modelo y será llave primaria de esta tercera tabla.

   - Uno a muchos: En esta relación creo dos tablas, por cada uno de los modelos, y en la relación de muchos creo un atributo referenciado a la otra tabla que será una llave foránea.

   - Uno a uno: En esta relación creo dos tablas, por cada uno de los modelos, y en una de las relaciones creo un atributo referenciado a la otra tabla que será a la vez llave primaria y llave foránea a la otra tabla.

2. Relación de herencia

- Opción 1: Creamos una tabla por cada una de nuestras clases. En nuestra tabla de donde vamos a heredar creamos un atributo identificador, además de sus correspondientes atributos adicionales. A partir de aquí nos creamos las demás tablas que heredan de ésta y nos creamos un atributo identificador igual que la tabla a la que vamos a heredar, además de sus atributos adicionales.

- Opción 2: Creamos todas las tablas que heredan de nuestra tabla a heredar y dentro de cada tabla introducimos todos los campos de nuestra tabla a heredar. En esta opción tendremos una tabla menos que en la primera opción.
- Opción 3: Nos creamos una única tabla con todos los atributos de nuestras clases que heredan y sobre las que heredan, y añadimos un atributo donde haremos la distinción de que atributos se heredan en ese momento.

Tarea DSI 6.1: Diseño del modelo físico de datos.

En esta tarea se realiza el diseño del modelo físico de datos. A continuación se detallarán algunos aspectos de este modelo.

- Tabla Cliente. Esta tabla contiene los datos personales del cliente, así como sus datos de conexión, lo cual le permitirá ser reconocido en la aplicación como el usuario que realmente se esta conectando y que sus datos corresponde con dicho usuario.

- Tabla Producto. Contiene los datos de los producto que se van a visualizar en la pagina web de nuestra aplicación.

- Tabla Pedido. Contiene los datos del pedido, así como en nombre del cliente que ha realizado el pedido.

- Tabla Listapedido. Esta tabla contiene las líneas de los productos que ha comprado un cliente con un determinado identificador de pedido, el cual es único con respecto a los demás pedidos.

3.3.5 ACTIVIDAD DSI 7: VERIFICACIÓN Y ACEPTACIÓN DE LA ARQUITECTURA DEL SISTEMA

En este punto se comprueba la calidad de las especificaciones del diseño del sistema de información garantizando la coherencia entre los distintos modelos así como la calidad técnica de cada especificación.

Tarea DSI 7.1: Verificación de las especificaciones de diseño

Veo que está asegurada la calidad formal de los distintos modelos ya que para la elaboración de cada producto se han seguido las especificaciones correspondientes. Para esto he revisado las siguientes tareas:
Catálogo de Requisitos (DSI 1.2), Catálogo de Excepciones (DSI 1.3), Catálogo de Normas(DSI 1.4), Diseño de la Arquitectura del Sistema (DSI 1.5), Entorno Tecnológico del Sistema(DSI 1.6), Diseño Detallado de Subsistemas de Soporte (DSI 2.1), Modelo Físico de Datos Optimizado (DSI 6.3), Esquemas Físicos de Datos (DSI 6.4), Asignación de Esquemas Físicos de Datos a Nodos (DSI 6.4), Diseño de la Realización de los Casos de Uso (DSI 3.4),Diseño de Interfaz de Usuario (DSI 3.3), Modelo de Clases de Diseño (DSI 4.6) y Comportamiento de Clases de Diseño (DSI 4.4).

Tarea DSI 7.2: Análisis de consistencia de las especificaciones de diseño

Durante el desarrollo de los diferentes modelos se ha seguido un diseño coherente de estos comprobando que no presentasen ambigüedades o duplicación de información. Por lo tanto los productos de salida son los adecuados para este punto del proceso de diseño. Las comprobaciones que se han ido realizando son las siguientes:

- Respecto a la arquitectura del sistema / diseño detallado de subsistemas, he ido comprobando que cada clase tiene su propio paquete.

- En el modelo de clases he ido comprobando que cada mensaje entre objeto se corresponde con una operación de clase y que todos los mensajes se envían a las clase correctas.
- Cada diagrama de iteración de objetos tiene una correspondencia con su modelo de clases y respecto a los diagramas de transición de estado comprobamos que cada uno de los eventos que se produce en la página web se corresponde con una operación de clase.

Tarea DSI 7.3: Aceptación de la arquitectura del sistema

He realizado una comprobación completa del sistema, intentando buscar que no haya incoherencias ni ambigüedades. Buscando que todos los elementos del sistema se integren correctamente y exista una correlación directa con el interfaz web del usuario. De forma que quede implementados de forma clara y concisa todos los diagramas y modelos que he realizado en el proyecto.

3.3.6 ACTIVIDAD DSI 8: GENERACIÓN DE ESPECIFICACIONES DE CONSTRUCCIÓN

En esta actividad se detallan las especificaciones para la construcción del sistema de información, que en mi caso se trata de una página Web de productos hortofrutícolas.

Tarea DSI 8.1: Especificación del entorno de construcción

Detallo el entorno de desarrollo en el cuál se trabaja. Se definirán tanto el entorno tecnológico, así como las herramientas de construcción utilizadas:

- Entorno tecnológico

  Utilizo un portátil Pentium IV centrino a 1.8 Ghz con 3 Gbytes de memoria Ram, un disco duro de 60 Gbytes y un sistema operativo Windows Xp profesional.

- Herramientas de construcción

  Las herramientas para el desarrollo de la aplicación son:
  - Base de datos relacional *PosgrestSql.*
  - Lenguaje de programación Java

- Entorno de desarrollo MyEclipse
- Frameworks basado en la tecnología Java (*Spring, JSF e Hibernate*)
- Gestor Ftp Filezilla.

Tarea DSI 8.3: Elaboración de especificaciones de construcción

En este apartado realizo una descripción, más o menos precisa de cada unidad desarrollada en la aplicación. Se describirá que función realiza cada página web junto con algunas características que puedan ser interesantes.

- Pagina Web principal
  - Visualización de productos.

    Es el marco central de nuestra página Web de inicio, donde nos muestra los productos de la página Web con sus precios.

  - Panel de opciones.

    Es el marco izquierdo de nuestra página Web podemos seleccionar las distintas opciones que nos ofrecen, permitiéndonos poder ver lo marcado en el marco central de nuestra aplicación.

  - Panel datos de registro.

    Es un panel situado en el marco derecho de nuestra aplicación, en el podemos registrarnos en nuestra pagina Web, para poder realizar las compras y posterior envió de los productos comprados en la página.

  - Panel Total pedido

    Es un panel situado en el marco derecho de nuestra aplicación, en el vemos los productos que hemos seleccionado, pudiendo eliminar, incrementar o disminuir los kilos de los productos que hay se encuentra.

- Pagina Web Confirmar pedido

  En esta página mostramos los datos del pedido, así como los datos del cliente que realiza el pedido, hay una serie de campos como hora de entrega y

observaciones en la cual el usuario puede inserta los datos que considere pertinente ,así como la hora de entrega del pedido realizado.

- Pagina Web registro de usuario

  En esta pagina mostramos los datos que el usuario nos ha de suministrar para registrarse en nuestra pagina Web, en ella nos indica sus datos personales y el nombre de usuario y la contraseña con la cual se va a registrar en nuestra pagina Web.

  Esta página Web se utiliza también para consultar los datos de usuario registrado en la página, pudiendo hacer las modificaciones que considere necesarias de sus datos.

- Pagina Web Administración

  Esta pagina solo puede ser accedida por el administrador de la pagina Web, en ella se muestra varias opciones que permitirán la modificación de la pagina Web, así con la consulta de los pedidos hecha por los usuarios registrados en la pagina Web.

  Las opciones que nos muestran son:

  - Productos: aquí podemos modificar, o dar de alta un producto.
    - Alta producto: Nos pide una serie de datos (nombre, Variedad, precio, Etc.), una vez relleno estos datos procederíamos al envió de los datos, para poder ser almacenados en nuestra Base de Datos.

    - Modificar producto: no muestra el nombre del producto a modificar, una vez insertado dicho nombre y pulsando sobre la lupa, nos mostraría todo los productos, pudiendo modificar los distintos campos.

    - Salir: saldríamos de esta pagina Web, y volveríamos a la pagina central de la administración.

- Pedido: en esta pagina podemos consultar los pedidos hecho por los usuario así como realizar dichos pedido y almacenarlos en pedidos realizados

    - Pedidos sin realizar: nos muestra aquello pedido que han realizado el usuario indicando la fecha y hora del pedido, pudiendo realizar la opción de consultar para poder ver el pedido hecho por el usuario, o actualizar para modificar de pedido sin realizar a pedido realizado.

    - Pedido realizado: nos muestra todos los pedidos realizado y enviado a nuestro cliente.

- Salir: sale de la página Web del administrador volviendo a la página principal de nuestra aplicación.

Tarea DSI 8.4: Elaboración de especificaciones del modelo físico de datos

A continuación genero las especificaciones necesarias para la definición y creación de los elementos del modelo físico de datos.
El sistema gestor de base de datos en el que se basa la aplicación es PostgreSql, y el lenguaje de definición de datos que utilizaré será el estándar SQL, utilizado en aquellas partes de nuestro proyecto que lo requieran.
Para una mayor información del modelo físico de datos ver el diagrama de gestión de datos.

3.3.7 ACTIVIDAD DSI 9: DISEÑO DE LA MIGRACIÓN Y CARGA INICIAL DE DATOS

Por lo que ya comenté en la tarea DSI 4.7 no voy a realizar la migración y la carga inicial de datos. Cuando el cliente confirme la implantación de este software en su tienda se analizará su base de datos (si es que ya usaba un software anteriormente) y se estudiará la mejor forma para realizar la migración de datos.

3.3.8 ACTIVIDAD DSI 10: ESPECIFICACIÓN TECNICA DEL PLAN DE PRUEBAS

En esta tarea se realiza la especificación de detalle del plan de pruebas del sistema de información para cada uno de los niveles de prueba establecidos en el proceso Análisis del Sistema de Información:

Pruebas unitarias: pequeños módulos auxiliares, que se encargan de verificar la funcionalidad y la estructura de cada componente individual y que comprenden las verificaciones asociadas a cada componente del sistema de información.

Pruebas de integración comprenden verificaciones asociadas a grupos de componentes, generalmente reflejados en la definición de subsistemas de construcción o en el plan de integración del sistema de información. Tienen por objetivo verificar el correcto ensamblaje entre los distintos componentes.

Las ventajas de usar este tipo de pruebas son muchas, entre ellas podemos decir:

- Los errores son más fáciles de localizar.
- Los errores están más acotados.
- Se da más seguridad al programador.

Tarea DSI 10.1: Especificación del entorno de pruebas

En esta tarea defino detalladamente el entorno necesario para la realización de las pruebas del sistema: unitarias y de integración.

Debido a los escasos recursos de los que dispongo, el entorno de pruebas coincide con el entorno de construcción. Por ello para un mayor detalle de esta tarea ir a la tarea DSI 8.1 donde se especifica al detalle el entorno de construcción.

Las pruebas que se han llevado a cabo son simples, es decir, se han introducidos datos tantos correctos como incorrectos para comprobar su funcionamiento.

Tarea DSI 10.2: Especificación técnica de niveles de prueba

En este punto especificaré detalladamente los distintos niveles de pruebas con los que trabajo así como el nivel adecuado de aceptación para la aplicación.

Especificación de las pruebas unitarias:
Deberé controlar:
Campos obligatorios: Se deberá controlar que el usuario introduce cada uno de los campos obligatorios. En caso de que no se cumpla, se procede ha avisar al usuario mediante un mensaje indicándole que campos son obligatorios.

Excepciones: Este tipo de pruebas tienen como objetivo asegurar la integridad de cada unidad de la aplicación. Cada unidad deberá tratar las posibles excepciones que pueda lanzar el sistema. Algunos de ellos son: error en la conversión de los datos, problemas de integridad en base datos, errores de violación de clave primaria etc. Para cada una de las pruebas se requiere que cada excepción sea tratada, informando al usuario de dicha excepción mediante un mensaje por pantalla de lo ocurrido para que éste pueda localizar rápidamente el error y corregirlo.

Funcionamiento: Por último y quizás lo más importante, se deberá asegurar que cada unidad implementada realiza la acción a la cuál estaba destinada hacer. Con ello facilitaremos el trabajo de integración de cada una de las partes. Para probar el funcionamiento se procederá a introducir datos tantos correctos como incorrectos y ver en cada momento cuál es la reacción de nuestra aplicación.

Las pruebas de integración:
Se empezarán a integrar las unidades desde los componentes situados en los niveles más inferiores, es decir aquellas partes o unidades que no requieren de otras. Una vez integrada una unidad se procede a realizar las pruebas unitarias necesarias. Si todo va bien se procede a la siguiente, así hasta integrar la última componente. De esta forma aseguramos que cada componente que se añade funciona correctamente y no añadimos nada más a la aplicación, hasta que lo anterior funcione correctamente. Al final se obtendrá una aplicación que funcionará correctamente.
Para facilitar dicha integración, antes de programar, se establece una nomenclatura que cada uno de los componentes que participan deben de llevar a cabo. Cada componente de cualquier unidad debe estar identificado completamente. El nombre o identificador utilizado debe ser significativo.

Las pruebas realizadas en la integración son:

Funcionamiento de cada unidad: Como se dijo anteriormente cada unidad que se integra debe funcionar correctamente, del mismo modo que lo hacía en las pruebas unitarias, es decir se deben seguir controlando las excepciones, campos obligatorios etc. Debemos de tener en cuenta o mejor dicho probar que la integración de una unidad no afecta al funcionamiento correcto de lo anterior.

Comunicación entre unidades: En aquellos casos en que una unidad importe a otra, debemos verificar el funcionamiento de las llamadas entre dichas unidades. Para ello probamos cada una de las llamadas que se pueden realizar de una unidad hacia otra, y comprobar que los resultados obtenidos sean los esperados. Si durante la realización de dicha prueba se observa alguna alteración en el funcionamiento de alguna de las partes, se procederá a solucionar dicho problema. En caso de buen funcionamiento seguiríamos integrando.

Funcionamiento global: Ver que el sistema funciona correctamente y que cada una de las operaciones, como pueden ser modificaciones, inserciones o eliminados lógicos actualizan la base de datos correctamente.

Tarea DSI 10.3: Revisión de planificación de pruebas

A partir del plan de pruebas de la tarea 10.2 se completa y se especifica la planificación de las pruebas, determinando los distintos perfiles implicados en la preparación y ejecución de las pruebas y en la evaluación de los resultados, así como el tiempo estimado para la realización de cada uno de los niveles de prueba, de acuerdo a la estrategia de integración establecida.

3.3.9 ACTIVIDAD DSI 11: ESTABLECIMIENTO DE REQUISITOS DE IMPLANTACIÓN

En esta actividad se definen los requisitos necesarios que deben adjuntarse con la documentación para que el usuario pueda manipular el nuevo sistema con el menor número de problemas.

Tarea DSI 11.1: Especificación de requisitos de documentación de usuario

Esta tarea recoge toda la información necesaria para la especificación de la documentación a entregar al usuario.

El formato en el que distribuiré el manual de usuario será en formato PDF. Para una mayor comodidad, aunque con un mayor coste, se puede proporcionar dicho manual mediante un libro impreso.

Una posible estructura podría contener en las primeras páginas un pequeño guión donde indica donde localizar cierta información. Este manual contendría información sobre las características de la aplicación como se realiza cada una de las acciones de la aplicación, información sobre posibles funcionalidades que se pueden añadir en el futuro, requisitos necesarios que debe tener el sistema para su implantación etc.

Tarea DSI 11.2: Especificación de requisitos de implantación

Tarea que define los requisitos de implantación de nuestro sistema. La formación de los usuarios requerida son conocimientos básicos.

Los requisitos requeridos en infraestructuras e instalación aconsejados para la implantación del software son los siguientes:

1. Servidor: el servidor que contiene la aplicación Web y la base de datos ha de ser cualquier ordenador actual con gran cantidad de memoria, tanto de memoria RAM, como de disco duro, Incluso se podría montar dos ordenadores, uno conteniendo la base de datos de la aplicación, y otro con la aplicación en si.

2. Cliente: el cliente puede se cualquier ordenador que contenga un navegador actualizado a la ultima versión (Mozilla Firefox, Internet Explorer etc…).

3.4 CONSTRUCCIÓN DEL SISTEMA DE INFORMACIÓN (CSI).

3.4.1 ACTIVIDAD CSI 1: PREPARACIÓN DEL ENTORNO DE GENERACIÓN Y CONSTRUCCIÓN

En esta tarea se asegura la disponibilidad de la infraestructura necesaria para la generación del código de los componentes y procedimientos del sistema de información

y la disponibilidad de todos los medios y facilidades para que se pueda llevar a cabo la construcción del sistema de información. Entre estos medios, cabe destacar la preparación de los puestos de trabajo, equipos físicos y lógicos, gestores de bases de datos, bibliotecas de programas, herramientas de generación de código, bases de datos o ficheros de prueba, entre otros.

Tarea CSI 1.1: Implantación de la base de datos física o ficheros

Se crean las tablas del sistema gestor de base de datos, paso el modelo entidad-relación, realizado con una herramienta UML, a la base de datos relacional PostgreSql. El alias de la base de datos será "HORTOBD".
De cada tabla defino el nombre de los campos, los tipos de datos y su longitud, las asociaciones entre las tablas, las llaves primarias y las restricciones necesarias según la estructura física de nuestra Base de Datos, etc.

Tarea CSI 1.2: Preparación del entorno de construcción

En esta tarea se asegura la disponibilidad de la infraestructura necesaria para la generación del código de los componentes y procedimientos del sistema de información y se prepara el entorno en el que se construirán los componentes del sistema de información. Las herramientas que voy a utilizar en el desarrollo del programa son las ya dichas en la tarea
DSI 8.1. En cuanto al lugar de trabajo, trabajo en mi casa con dos ordenadores propios (características descritas en el DSI 8.1), el cual tiene instalados los programas nombrados antes.

3.4.2 ACTIVIDAD CSI 2: GENERACIÓN DEL CÓDIGO DE LOS COMPONENTES Y PROCEDIMIENTOS

La generación del código de los componentes y procedimientos se hace según las especificaciones de construcción del sistema de información, y conforme al plan de integración del sistema de información.

El objetivo de esta actividad es la codificación de los componentes del sistema de información, a partir de las especificaciones de construcción obtenidas en el proceso Diseño del Sistema de Información (DSI), así como la construcción de los procedimientos de operación y seguridad establecidos para el mismo.

En paralelo a esta actividad, se desarrollan las actividades relacionadas con las pruebas unitarias y de integración del sistema de información. Esto permite una construcción incremental, en el caso de que así se haya especificado en el plan de pruebas y en el plan de integración del sistema de información.

No sólo hay que generar el código, sino que también hay que construir todas las tablas de la base datos, con sus índices y relaciones, según las especificaciones de las actividades de análisis y diseño.

Tarea CSI 2.1: Generación del código de componentes

En esta tarea se genera el código correspondiente a cada uno de los componentes del sistema de información, identificados en la tarea DSI 8.2.

Para generar el código fuente se tienen en cuenta los estándares de nomenclatura, codificación y calidad utilizados por la organización y recogidos en el catálogo de normas.

Para conseguir dicho objetivo, se recoge la información relativa al producto del diseño: Especificaciones de construcción del sistema de información, se prepara el entorno de construcción, se genera el código de cada uno de los componentes del sistema de información y se van realizando, a medida que se vaya finalizando la construcción, las pruebas unitarias de cada uno de ellos y las de integración entre subsistemas.

CSI 2.2: Generación del código de los procedimientos de operación y seguridad

El objetivo de esta tarea es generar los procedimientos de operación y administración del sistema de información, así como los procedimientos de seguridad y control de

acceso, necesarios para ejecutar el sistema una vez que se haya implantado y esté en producción.

3.4.3 ACTIVIDAD CSI 3: EJECUCIÓN DE LAS PRUEBAS UNITARIAS

En esta actividad se realizan las pruebas unitarias de cada uno de los componentes del sistema de información, una vez codificados, con el objeto de comprobar que su estructura es correcta y que se ajustan a la funcionalidad establecida.

Tarea CSI 3.1: Preparación del entorno de las pruebas unitarias

Esta tarea prepara los todos recursos necesarios para la realización de pruebas unitarias de cada una de las componentes del sistema de información.
El entorno para llevar a cabo las pruebas unitarias coincide con el entorno de trabajo.
Para obtener una mayor información consultar la documentación de las pruebas unitarias especificada en el DSI 8.1.

Tarea CSI 3.2: Realización y evaluación de las pruebas unitarias

El objetivo de esta tarea es comprobar el correcto funcionamiento de los componentes sistema de información, codificados en la actividad de Generación del Código de los Componentes y Procedimientos (CSI 2).
Se ha realizado el plan de pruebas especificado en el apartado DSI 10.2, obteniendo una serie de resultados, los cuáles se han analizado y estudiado en detalle. Según los resultados de las pruebas llevadas a cabo, se ha procedido a solucionar los problemas, en caso de existir, o simplemente a continuar con el trabajo.

3.4.4 ACTIVIDAD CSI 4: EJECUCIÓN DE LAS PRUEBAS DE INTEGRACIÓN

El objetivo de esta actividad es verificar si los componentes o subsistemas interactúan correctamente a través de sus interfaces, tanto internas como externas, cubren la funcionalidad establecida y se ajustan a los requisitos especificados.

Tarea CSI 4.1: Preparación del entorno de las pruebas de integración

En esta tarea se disponen los recursos necesarios para realizar las pruebas de integración de componentes y subsistemas que conforman el sistema de información.
Los recursos necesarios para realizar las pruebas de integración de los componentes y subsistemas que conforman el sistema de información será el mismo que el entorno usado para las pruebas unitarias.

Tarea CSI 4.2: Realización de las pruebas de integración

El objetivo de esta tarea es verificar el correcto funcionamiento de las interfaces existentes entre los distintos componentes y subsistemas, conforme a las verificaciones establecidas para el nivel de pruebas de integración.
Se ha llevado a cabo la ejecución del plan de pruebas especificado en el apartado DSI 10.2 para las pruebas de integración.

Tarea CSI 4.3: Evaluación del resultado de las pruebas de integración

El objetivo de esta tarea es analizar los resultados de las pruebas de integración y efectuar a su evaluación.
Se ha llevado a cabo la evaluación de las pruebas unitarias realizadas. A partir de estas, compara los resultados obtenidos con los esperados, identifica (en caso de existir) problemas para poder solucionarlo y vuelve a ejecutar el plan de pruebas, hasta que los resultados esperados y obtenidos sean equivalentes. Una vez que todos los componentes realizan este proceso se procede a la integración.

Tras las evaluaciones anteriores se procede a su integración. Al igual que antes, comienzo a realizar pruebas y voy comparando resultados esperados y obtenidos.
Mientras que estos no sean equivalentes, seguiré ejecutando pruebas de integración.
Puedo, llegado un punto, cambiar los casos de prueba y en caso de que persistan problemas, se procede a la desintegración de dicha parte para realizar de nuevo las pruebas unitarias.
Como resultado de las pruebas de Integración el programa recoge un grado de cumplimiento alto, por lo tanto, no es necesario volver a realizar el plan de pruebas.

3.4.5 CSI 5: EJECUCIÓN DE LAS PRUEBAS DEL SISTEMA

El objetivo de las pruebas del sistema es comprobar la integración del sistema de información globalmente, verificando el funcionamiento correcto de las interfaces entre los distintos subsistemas que lo componen y con el resto de sistemas de información con los que se comunica.
En la realización de estas pruebas es importante comprobar la cobertura de los requisitos, dado que su incumplimiento puede comprometer la aceptación del sistema por el equipo de operación responsable de realizar las pruebas de implantación del sistema, que se llevarán a cabo en el proceso Implantación y Aceptación del Sistema.

CSI 5.1: Preparación del Entorno de las Pruebas del Sistema

En esta tarea se preparan todos los recursos necesarios para realizar las pruebas del sistema, de acuerdo a las características del entorno establecidas en el plan de pruebas.
Para ello se asegura la disponibilidad del entorno y de los datos necesarios para ejecutar estas pruebas, se preparan las bibliotecas o librerías que se estimen oportunas para la realización de las mismas, así como los procedimientos manuales o automáticos asociados.

CSI 5.2: Realización de las Pruebas del Sistema

El objetivo de esta tarea es comprobar la integración de todos los subsistemas y componentes del sistema de información, así como la interacción del mismo con otros sistemas de información con los que se relaciona, de acuerdo a las verificaciones establecidas para el nivel de pruebas del sistema.

CSI 5.3: Evaluación del Resultado de las Pruebas del Sistema

El objetivo de esta tarea es analizar los resultados de las pruebas de sistema y efectuar su evaluación. Dicha evaluación recoge el grado de cumplimiento de las pruebas y consiste en:

- Comparar los resultados obtenidos con los esperados: el único resultado esperado y necesario es que el programa funcione correctamente, las pruebas sobre la

integración entre BD-interfaz, y las unidades de interfaz en las consultas mínimas funciona correctamente.

- Identificar el origen de cada problema detectado para poder remitirlo, determinar la envergadura de las modificaciones y qué acciones deben llevarse a cabo para resolverlo de forma satisfactoria.
- Indicar si el plan de pruebas debe volver a realizarse total o parcialmente, y si será necesario contemplar nuevos casos de prueba no considerados anteriormente: si se le añaden componentes en la nueva versión si será necesario volver a aplicar las mismas pruebas.

3.4.6 ACTIVIDAD CSI 6: ELABORACIÓN DE LOS MANUALES DE USUARIO

El objetivo de esta actividad es elaborar la documentación de usuario, tanto de usuario final como de instalación.

El manual de instalación es dónde se especifica cómo instalar, desinstalar y ejecutar la aplicación. El manual de usuario final debe explicar de forma clara y precisa cada una de las funciones de todas las aplicaciones que conforman el programa, de tal forma que el usuario no precise ayuda de nadie más para realizar sus servicios.

Los requisitos de documentación especifican aspectos relativos a los tipos de documentos a elaborar y estándares a seguir en la generación de los mismos.

Los requisitos de documentación especifican aspectos relativos a los tipos de documentos a elaborar y estándares a seguir en la generación de los mismos.

3.4.7 ACTIVIDAD CSI 7: DEFINICIÓN DE LA FORMACIÓN DE USUARIOS FINALES

En esta actividad se establecen las necesidades de formación del usuario final, con el objetivo de conseguir la explotación eficaz del nuevo sistema.

En este caso no será necesaria ninguna formación de usuarios finales ya que el software se ha realizado de forma que sea lo más intuitivo posible. Desde el primer día de uso podrá usar el programa sin necesidad de ninguna formación anterior, y en caso de que el usuario tuviera alguna duda tendrá a su disposición el manual de usuario o la ayuda incluida.

Capítulo 4:
Manual de Usuario

En este capítulo vamos a explicar la instalación, manejo y configuración de nuestra aplicación. En un principio explicaremos como se instala todo el software necesario para la instalación, después pasaremos a explicar el funcionamiento y como se gestiona toda la aplicación, separaremos la documentación en 3 partes, una primera instalación y configuración explicada anteriormente, una segunda sobre el manual del usuario, donde explicaremos como es la pagina web y el entorno que le rodea, así como todos los proceso que hay que seguir para la compra de los productos , y una tercera y definitiva dedicada al administrador y los procesos que puede realizar para llevar la gestión de la página web.

4.1 Instalación

Para la instalación de la aplicación se instala un servidor web, para este caso, al ser una aplicación desarrollada con Java tendremos que instalar como servidor web *Apache* y como contenedor Servlet *Apache TomCat,* para la gestión de los datos usaremos una base de datos relacional (PostgreSql). Todos estos programas anteriormente indicados podremos descargarlos de las páginas web:
http://www.apache.org
http://www.tomcat.apache.org
http://www.postgresql.org

4.1.1 Instalación. Máquina virtual de Java

Se descarga la Java Virtual Machine de la página web de Sun MicroSystems (http://java.com/es/download/), y se procede a su instalación. La versión sobre la que trabaja el programa será la versión 1.5 o superior.
Esta aplicación permite ejecutar la aplicación en el contenedor de servlet Tomcat.

4.1.2 Instalación de Base de datos PostgreSql

Descargar el gestor de base de datos PostgreSql de la página web, seguir todos los pasos para su instalación indicando el usuario "administrador" y la clave, así como el puerto al cual debemos conectarnos para poder acceder a la base de datos.

Una vez instalado se arranca la aplicación PgAdmin (Figura 4.1), conectando a PostgreSQL Database Server 8.3, donde se mostrará todas las bases de datos del sistema, Tras esto, debe crearse una nueva base de datos a la que la llamaremos "hortobd" usando la opción "bases de datos" como se aprecia en la Figura 4.2. Una vez creada la base de datos, se pulsa sobre la opción restaurar, cargando así el fichero de datos de la aplicación, denominado hortobasededatos.backup, para ello se selecciona la opción herramientas->restaurar de la aplicación pgAdmin. Este fichero contiene los datos, así como las tablas y sus respectivos enlaces entre ellas, dejando la base de datos lista para poder trabajar.

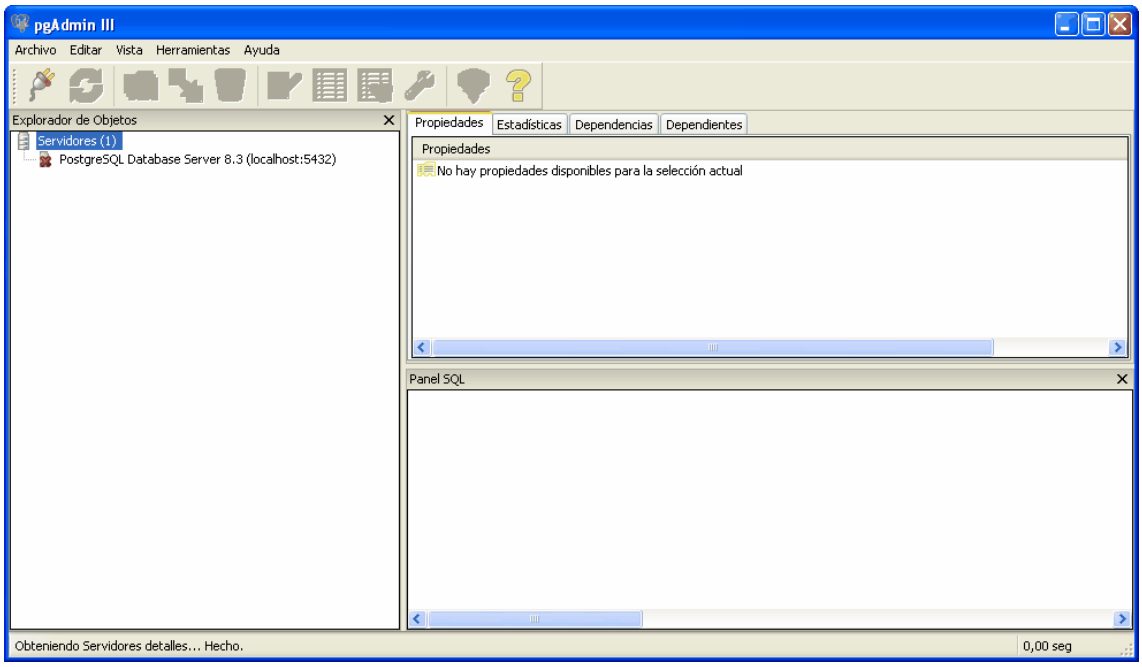


Figura 4.1 Instalación y configuración de la base de datos

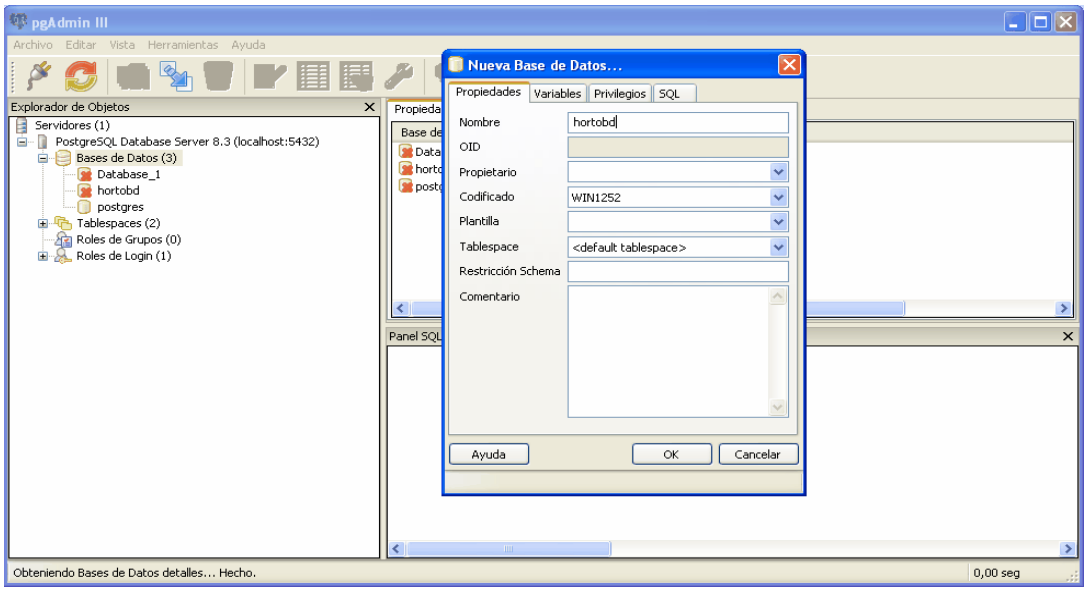


Figura 4.2 Creación de la base de datos

4.1.3 Instalación de contenedor servlet Apache Tomcat

Tras descargar la aplicación Apache Tomcat de la página web oficial Apache.

Haciendo un doble click sobre el icono, se procederá a la instalación, en la cual se pedirá el directorio de instalación del Apache Tomcat (Figura 4.3).

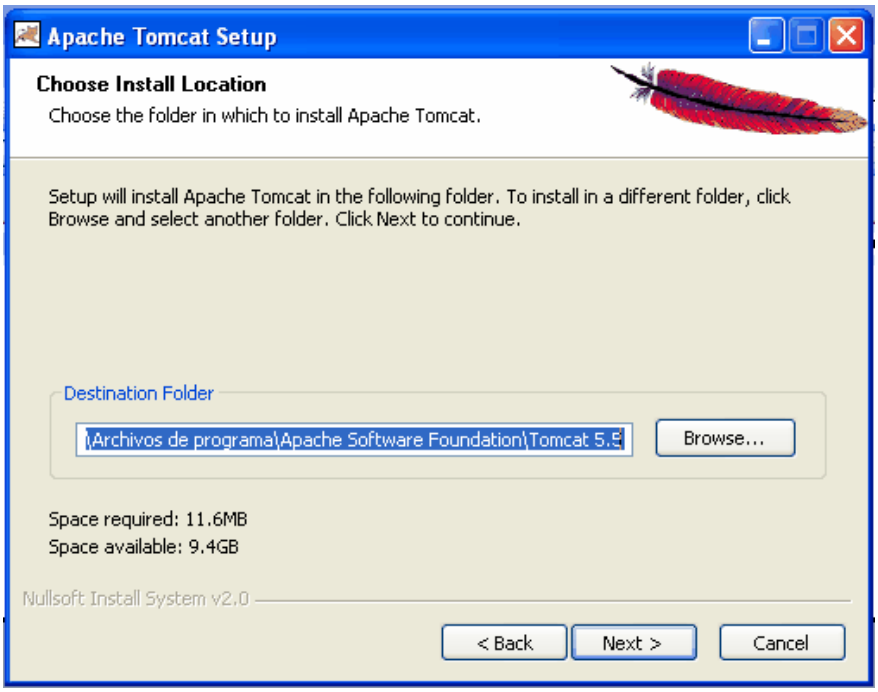


Figura 4.3 Directorio de instalación de Apache TomCat

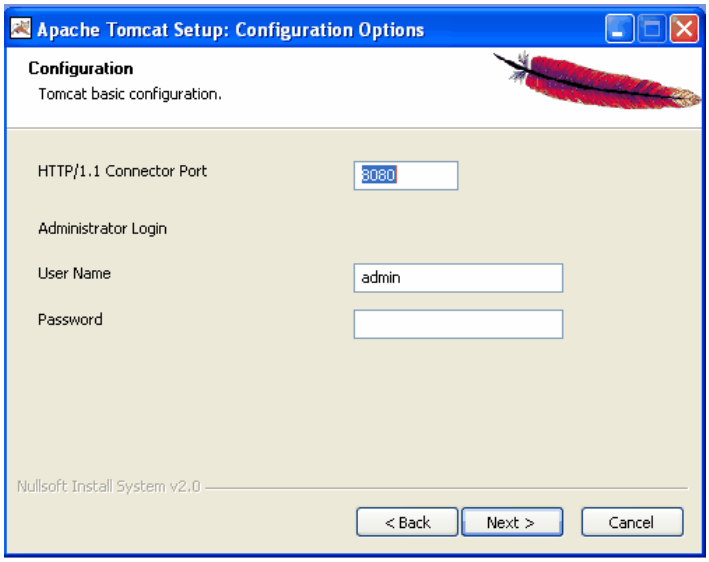


Figura 4.4 Opciones de configuración

Siguiendo con la instalación, se llega a la siguiente ventana (Figura 4.4), en la que se pedirá el puerto al que se deberán conectar para ejecutar nuestra aplicación tanto localmente como a través de Internet. Dejando por defecto esta dirección de puerto e indicando el nombre de usuario y contraseña del administrador de la aplicación.

En el siguiente paso (Figura 4.5), se pide la dirección donde se encuentra la máquina virtual de Java, buscándola en el directorio por defecto, si no la encuentra se indica la dirección donde esta, y una vez seleccionada se continua con la instalación de la aplicación

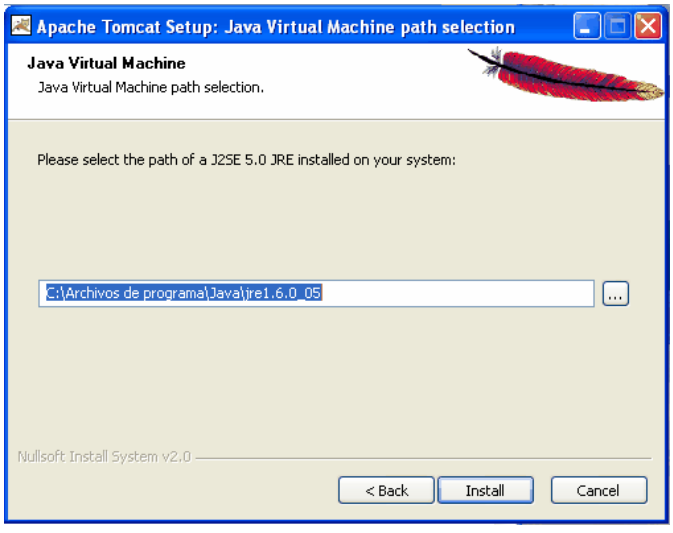


Figura 4.5 Opciones de configuración

Una vez instalada la aplicación para ver que se ha cargado bien Tomcat, se arranca el navegador y se teclea http://localhost:8080/index.jsp el cual se dirigirá a la página de configuración de Tomcat (Figura 4.6), con esto se habrá acabado la instalación de Tomcat. El siguiente paso será la instalación de la aplicación, para ello de cargará y se desplegará en el Tomcat Manager el fichero .war (Figura 4.7).

Una vez desplegada la aplicación se vera en el navegador, para ello se ejecuta la aplicación tecleando http://localhost:8080/tiendafrutas. La razón por la que se teclea esta ruta es para indicarle que la aplicación esta instalada en nuestro servidor local, en el puerto 8080, que es donde esta escuchando el servidor Apache y el contenedor de servlet Apache Tomcat, en cual contiene la aplicación tiendafrutas que es la que se ejecuta (Figura 4.8).

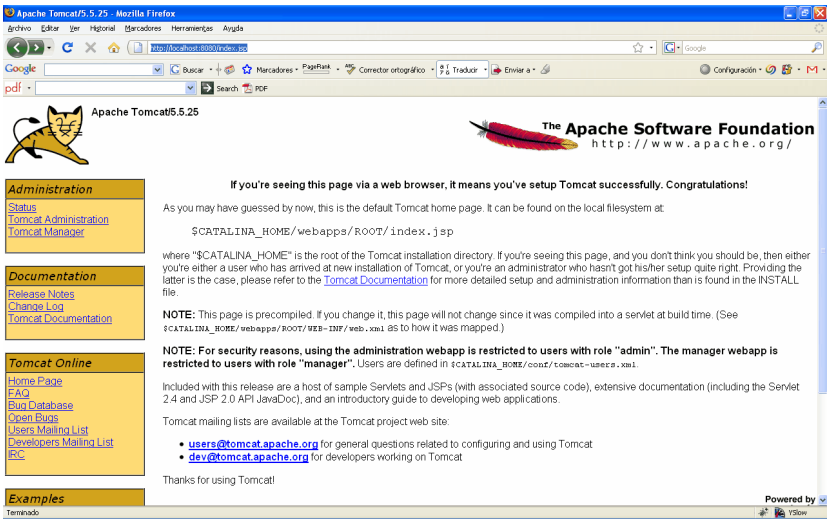


Figura 4.6 Pagina Web de configuración de Tomcat

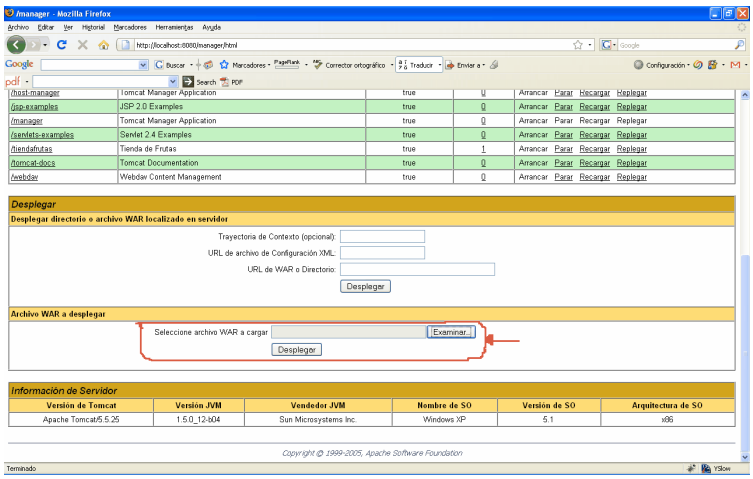


Figura 4.7 Pagina Web de configuración de Tomcat

4.1.4 Instalación de un servidor de Ftp (Filezilla)

Para la gestión de las imágenes de los productos de la página web de la aplicación vamos a a instalar un servidor de Ftp en el servidor y crearemos una cuenta para el administrador para que pueda subir y crear directorio en el directorio donde se encuentra las imágenes, esta imágenes que suba deberán cumplir las siguientes propiedades:

- Iconos.

  Tendrán una resolución de 58x76 pixeles y el nombre de la imagen será imgtitulo.gif. Esta imagen es la que vera el cliente en la pagina principal.

- Imágenes.

  Esta imagen puede ser mas grande teniendo resoluciones desde 58x76 pixeles hasta tamaños mas grande, siempre y cuando este dentro del marco de visualización, y el nombre de la imagen será 01.jpg

Para la gestión de esta imágenes montaremos el servidor de fichero Filezilla, crearemos una cuenta de usuario y una contraseña (Figura 4.8.1) e indicaremos el directorio donde se encuentra las imágenes (Figura 4.8.2), para que el administrador pueda gestionar este directorio creando e insertando imágenes en el directorio que considere necesario, y siempre que cumpla las condiciones anteriormente vista para crear icono e imágenes.

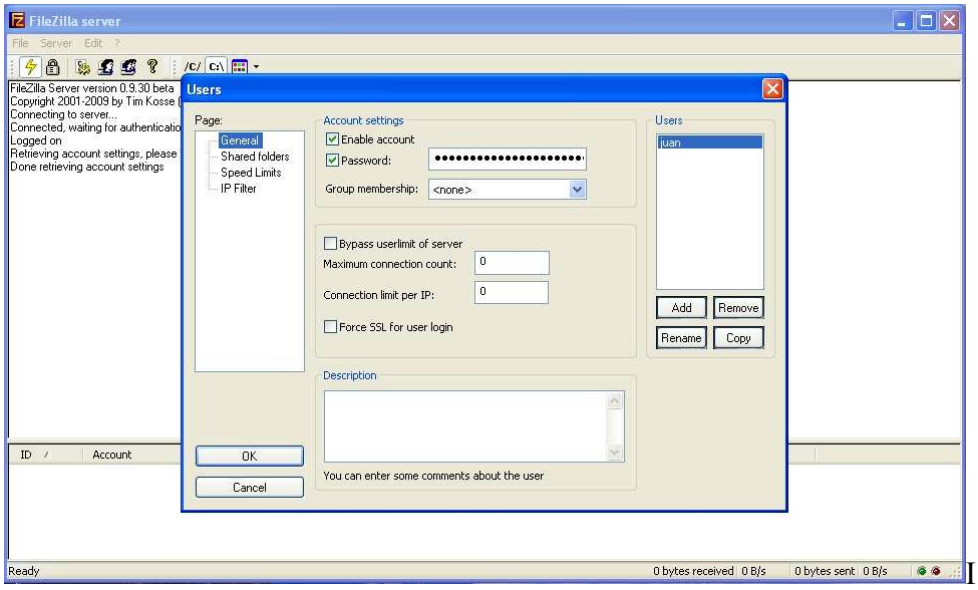


Figura 4.8.1 Creación de una cuenta de usuario

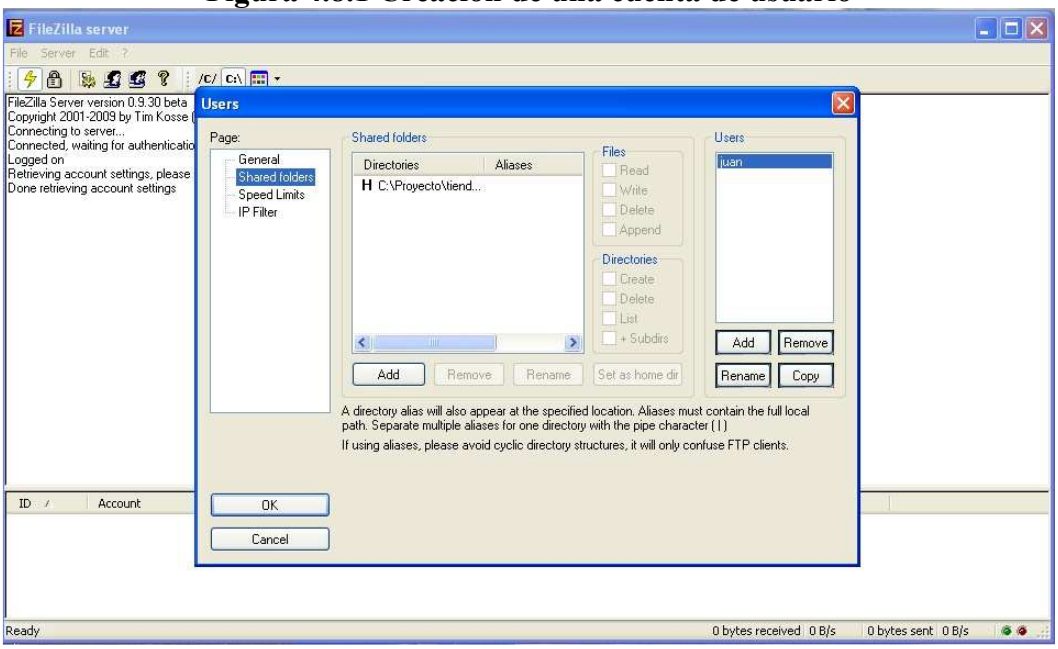


Figura 4.8.2 Creación de la ruta del directorio Ftp

4.2 Manual del programa

El programa consta de una serie de páginas web relacionadas entre sí, las cuales vamos a detallar a continuación

4.2.1 Página web principal

En esta página (Figura 4.9) es donde se realiza todos los procesos que permiten consultar y realizar la compra de los productos disponibles, así como darnos de alta si no estamos registrados. La página consta de tres partes bien definidas:

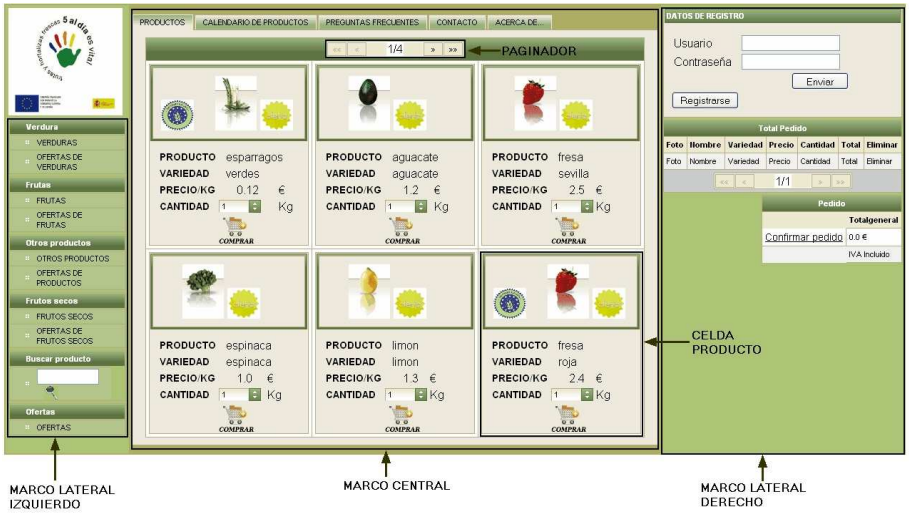


Figura 4.9 Página de inicio de nuestra aplicación

4.2.1.1 Marco lateral izquierdo

Esta parte de la página web principal (Figura 4.9) muestra el logotipo de la aplicación y debajo de éste, los posibles tipos de productos que se pueden visualizar en la página central, así como las ofertas de productos. En este caso se muestra cuatro tipos diferente: verduras, frutas, frutos secos y otros productos. Para cada tipo se pueden ver los productos disponibles de forma ordinaria y las ofertas de cada tipo. Hay una opción de buscar un producto, visualizando en la página central todas las variedades de este producto. Finalmente, debajo del recuadro de búsqueda, hay un apartado exclusivo de ofertas, que sirve para mostrar las ofertas de todos los productos de la página web.

4.2.1.2 Marco central

Esta parte de la página principal muestra todos los productos que podemos comprar. Consta de:

- **Un páginador**

  Muestra la página actual de los productos (Figura 4.9), el número de páginas que hay de dichos productos, y su desplazamiento de un página a otra. El paginador tiene una serie de botones que permite ir hacia delante o hacia atrás en la paginas.

- **Celda producto**

  Muestra la fotografía del producto (Figura 4.9) así como los respectivos logotipos de oferta y de ecológico (si proceden). También muestra los datos del producto (nombre, variedad, precio/kg o precio/unidad, cantidad) y un icono etiquetado como “COMPRAR”, que sirve para poder realizar el pedido de ese producto en la cantidad escogida. Pulsando sobre la imagen del producto se ve una información más detallada (Figura 4.10), donde hay un campo descripción, el cual muestra una información más detallada del producto, asi como sus cualidades nutricionales y todo aquello que el administrador quiera incluir en este campo. Habrá un icono volver, el cual nos retornará a la página principal de la aplicación.

Figura 4.10 Descripción del producto

4.2.1.3 Marco lateral derecho

Este marco (Figura 4.9) muestra una zona de registro de usuario, los datos de compra del usuario y un campo para confirmar el pedido, los cuales detallaremos a continuación.

- **Datos de registro**

  El usuario inserta sus datos de usuario y clave de usuario para registrarse en la página web (Figura 4.11), si no está registrado tiene la opción de registrarse, donde ira a otra página web llamada registro de usuario, en la cual podrá rellenar sus datos personales para su registro. Una vez registrado podrá ver y modificar sus datos personales, así como cerrar sesión o poder realizar y confirmar pedido.

Figura 4.11 Registro de usuario

Figura 4.12 Total Pedido.

- **Total pedido**

  Este marco (Figura 4.12) muestra los productos que se han comprados, así como la cantidad, precio y variedad del producto. También posee un campo eliminar, donde podrá eliminar el producto de la lista de compra. Hay que tener cuidado con este campo, ya que no pide confirmación de eliminación, borrando el producto directamente. El campo cantidad se puede incrementar o reducir variando el precio de compra y el total de pedido.

- **Pedido**

  Muestra el total de pedido y la opción confirmar pedido, redireccionando a otra página web donde realizar la compra del producto.

4.2.2 Página web registro de usuario

En esta página web se realiza la inserción de los datos personales (Nombre, Apellido, Dirección etc.) así como los datos de registro, los cuales permite posteriormente insertarlos en la página web principal para poder registrarse.

Figura 4.13 página web para el registro de un usuario

4.2.3 Página web formulario del pedido

Esta página web **(Figura 4.19)** es la definitiva antes de confirmar el pedido y almacenarlo en la base de datos de nuestra aplicación. Esta dividida en tres zonas.

- **Datos personales**

  Muestra los datos personales del cliente que realiza el pedido, indicando nombre, apellido, dirección, teléfono y código postal. Hay un campo observaciones en el cual el cliente podrá escribir todo aquello que considere pertinente de su pedido, por ejemplo características de los productos a entregar, otro teléfono de contacto u otra dirección de entrega si no esta en esa dirección.

- **Datos de pedido**

  Muestra datos del pedido, número del pedido, hora y fecha que se realiza el pedido, un campo para rellenar denominado hora de entrega, que rellenará el cliente si desea que se entregue a una hora determinada y otro llamado observaciones, donde indicará los datos que considere el usuario.

Figura 4.14 Formulario pedido

- **Pedido**

  Muestra los datos del pedido que ha realizado el usuario en la página anterior, indicando el total del pedido. El usuario no puede modificar ninguno de los datos desde esta página, tendrá que ir a la página de inicio para poder modificar dichos datos.

- **Totalgeneral**

  Indica el precio total del pedido.

- **Enviar**

  Procesa el pedido, enviándolo a la página web para almacenarlo y procesarlo y dándonos la opción de cerrar sesión o de continuar comprando.

- **Volver**

  Regresa a la página anterior

4.3 Manual de usuario de administración de nuestro portal web

La parte administración nos permite gestionar los pedidos de los usuarios, así como dar de alta los productos y su modificación. En ningún caso esta aplicación va a gestionar la facturación, lo cual no se descarta que en posibles versiones posteriores se implemente esta opción en la administración del programa, pero esta versión solo se va a dedicar a la gestión de pedidos para su posterior procesado y envío al cliente.

El pago del pedido se realizará una vez llegue el producto al cliente, para que el cliente pueda evaluar la mercancía antes de abonarla. No se descarta que en un futuro se pueda hacer a través de la web, con tarjeta de crédito o mediante una cuenta donde domiciliar el pago del pedido.
Nuestra Web de administración la hemos divididos en varios páginas Web que detallamos en los siguientes apartados:

Figura 4.16 Administración

4.3.1 Página Web Administración

A esta página web solo puede acceder el administrador. En ella se pueden gestionar los pedidos de los clientes, así como modificar y dar de alta los productos que posteriormente se verán en la página web. Esta dividida en tres zonas como se muestra (Figura 4.20):

4.3.2 Administración de pedidos

Esta página web permite dar de alta un producto, modificar dicho producto o salir de esta página web, usando las lengüetas disponibles en la parte superior (Figura 4.16).

4.3.2.1 Modificar producto

Consta de una opción de búsqueda en la parte superior, donde insertaremos el nombre del producto a modificar. Una vez insertado mostrará todas las variedades de dicho producto tal y como se aprecia (Figura 4.17).

Todas las opciones pueden ser modificadas, lo cual hará que posteriormente se modifique en la página web de la aplicación y los clientes puedan ver dichas modificaciones.

Al pulsar sobre el icono modificar, veremos una página donde indicará los datos del producto a modificar y tendremos un botón enviar el cual almacenará en el servidor las modificaciones que hayamos realizado sobre el producto (Figura 4.19).

4.3.2.2 Dar de alta un producto

En esta página (Figura 4.19) se da de alta un producto que se visualizará en la página web principal, donde el cliente podrá visualizar e incluso comprar dicho producto.

Figura 4.17 Modificar producto

Figura 4.18 Modificar producto

4.3.3 Gestión de pedidos

Esta página web (Figura 4.20) nos muestra toda la gestión que se realiza sobre los pedidos de los clientes. Los pedidos los clasificamos en dos clases:

4.3.3.1 Pedidos sin realizar

Este marco (Figura 4.21) muestra aquellos pedidos que han llegado de los clientes y que todavía no ha sido procesado ni enviado. Aquí se puede ver la hora de llegada, así como el cliente que lo ha realizado. Se puede ver mas datos pulsando sobre al botón consultar, obteniendo una información mas detallada del pedido. También se puede modificar sus estado en el proceso pasándolo a pedido procesado, para ello solo hay que cambiar Pedido Realizado a la opción “sí” y pulsando actualizar.

Figura 4.19 Alta de producto.

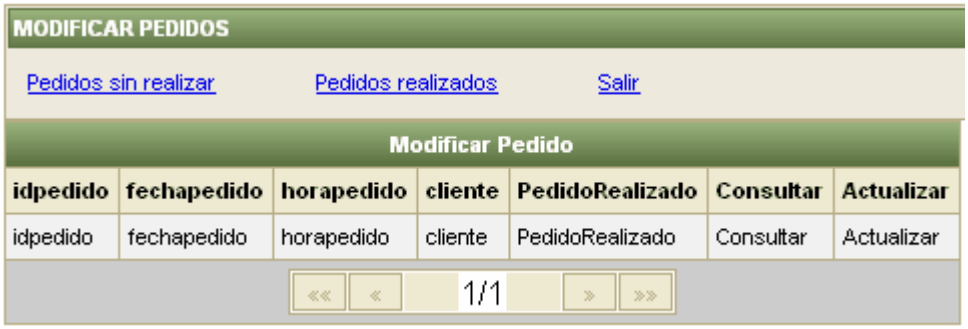


Figura 4.20 Modificar pedidos.

Figura 4.21 Pedidos sin enviar.

4.3.3.2 Pedidos realizados

Tiene las mismas propiedades que pedido sin realizar, solo que muestra los pedidos realizados.

4.3.3.3 Consultar del pedido

Muestra toda la información del pedido que un usuario ha realizado.

4.3.3.4 Modificación del pedido

Al pulsar sobre este botón, se pasa este pedido no procesado a pedido ya realizado y procesado o a la inversa.

4.3.3.5 Salir de la aplicación

Sale de este menú y vuelve al menú principal del administrador (Figura 4.16).

Conclusiones y líneas futuras

El proyecto realizado, ha aportado un conocimiento sobre la tecnología J2EE bastante actual, ya que es una mejora con respecto a la tecnología utilizado por Sun para el desarrollo de aplicaciones empresariales.

En nuestra opinión esta es una de las tendencias más fuertes para el desarrollo de aplicaciones empresariales, ya que como se comento en el apartado introductorio, tiende a desarrollar objetos más simples y más manejables.

Es sorpréndete los resultados que se llegan a tener una vez realizada la aplicación, y sobre todo la división en capas utilizando el modelo MVC (Modelo Vista Controlador), esto nos ha permitido dividir en un principio cada parte del proceso, pero con un resultado unificado al final.

Los problemas principales que considero que se han tenido han sido debidos a la gran cantidad de Framework que hay en el mercado para el desarrollo de aplicaciones J2EE, y considerar cuál es el más acto para el desarrollo de mi aplicación ha sido un poco difícil. Otro de los problemas fue el hecho de que la documentación encontrada es muy escasa en español, ya que casi toda ella está en inglés.

Es una tecnología bastante interesante para aprender y usar, debido a su facilidad para el desarrollo de aplicaciones empresariales y su facilidad de uso una vez aprendido el funcionamiento. Sólo hay que ver las ofertas de trabajos de algunas empresas del ámbito de la tecnología de la información aplicada a entornos web, una gran parte demanda algunos de estos Frameworks utilizados anteriormente.

Para aquellos estudiantes, ingenieros y programadores que se quieran meter en este mundo, considero que hay un gran abanico de posibilidades para el desarrollo de aplicaciones de este tipo, aparte del gran incremento de Framework que se añade a los ya anteriores y que se especializa en alguna parte del desarrollo de estos tipos de sistema de información.

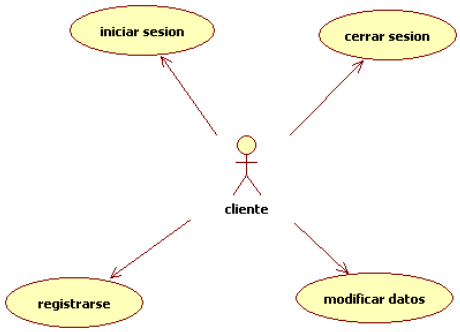
No considero que éste sea el mejor entorno de desarrollo de aplicaciones web, ya que hay más entornos de desarrollo como son Lamp (Linux, Apache y MySQL), Asp.Net, Groovy, Ajax, Flex, etc., pero sí considero que es unos de los mejores debido a la gran potencia de Java como lenguaje de programación, y a la gran cantidad de programadores y empresas que esta aportando constantemente documentación y posibles mejoras sobre esta tecnología.

Por los que se refiere al programa considero que el programa estaría listo para funcionar y poder trabajar, aunque siempre se pueden hacer algunas mejoras que permita un mejor trabajo entre cliente y empresa. Entre las mejoras que considero mas importantes destacan el montar un sistema de pago por Internet mediante tarjetas de crédito, o posibles listados de consulta que podrían hacer falta al administrador o al cliente.

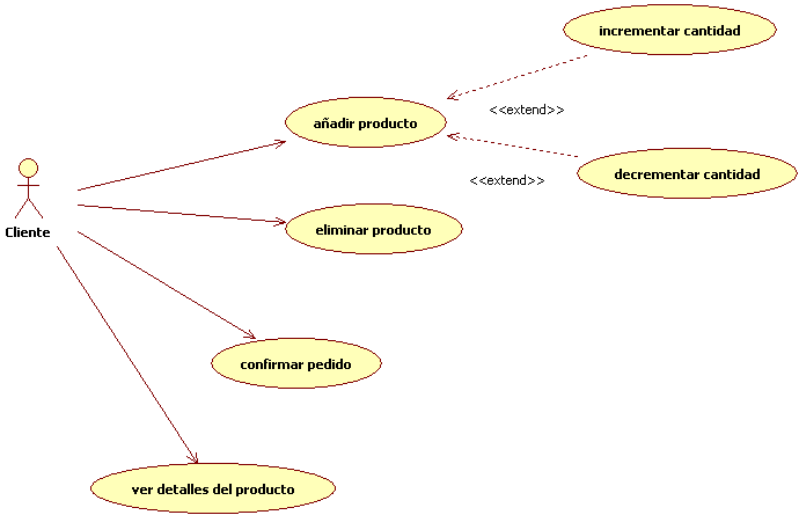
Apéndice A. Diagramas

En este apéndice se mostrarán los distintos diagramas que se obtienen durante el desarrollo de la metodología Métrica 3 en el capítulo 3. En dicho capítulo se referencia los diferentes apartados del apéndice.

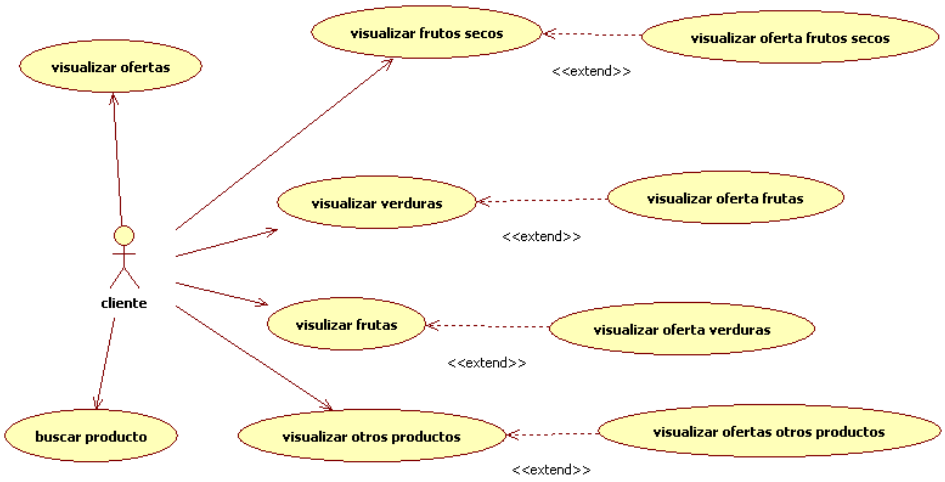
A.1 Diagramas de casos de uso



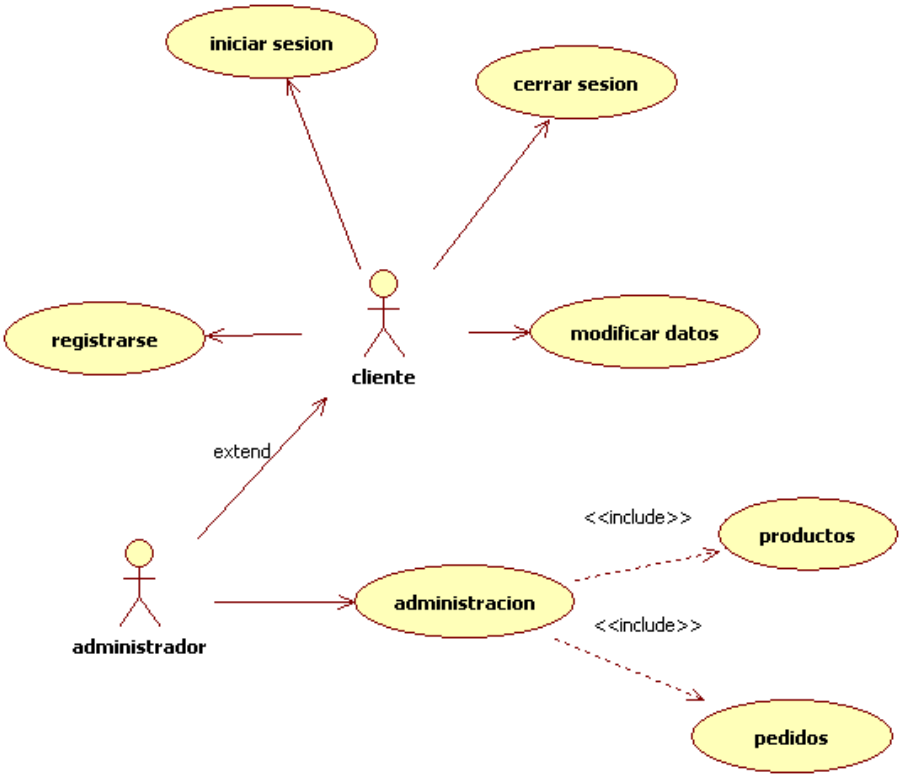
A.1 caso de uso del usuario cliente



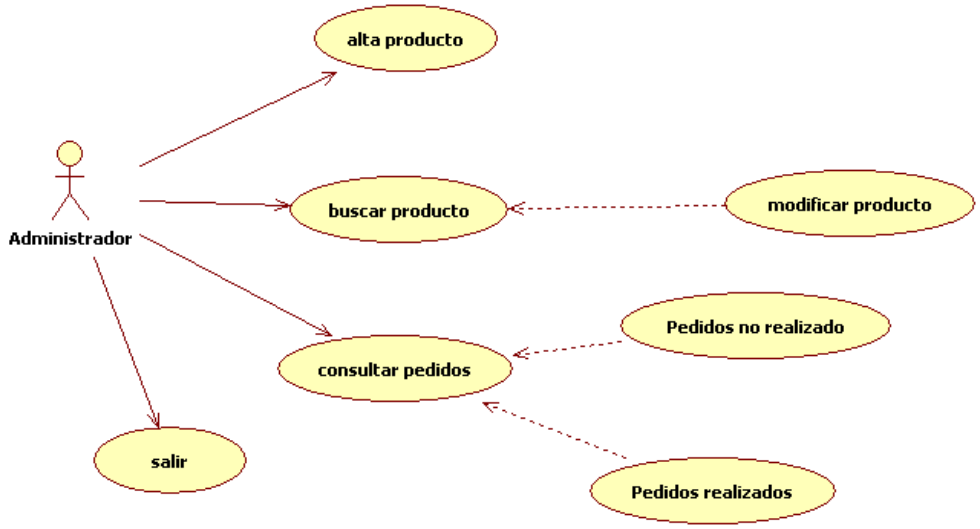
A.2 caso de uso realización de pedido por parte del cliente



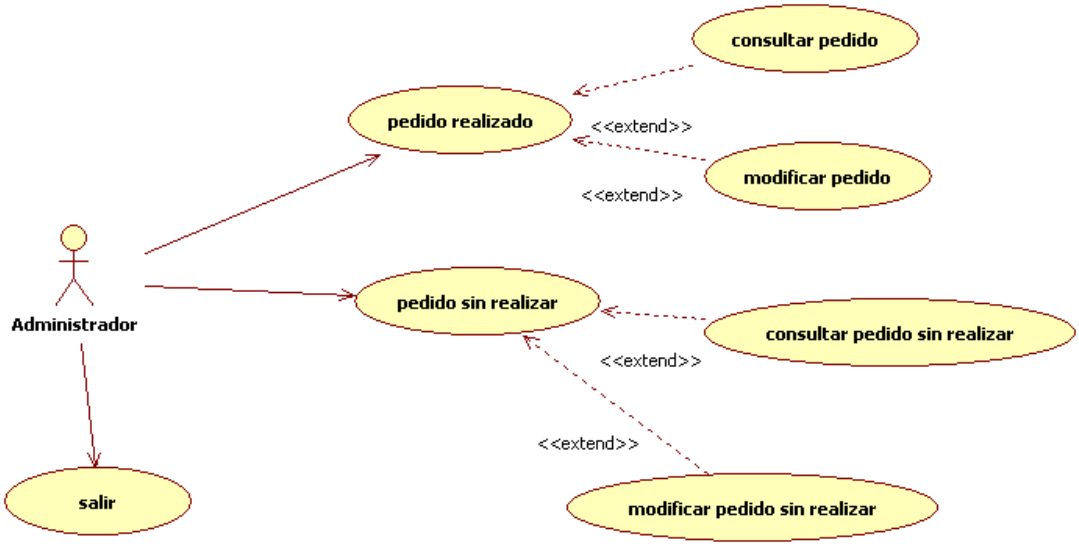
A.3 caso de uso consulta de productos



A.4 caso de uso relación administrador-cliente



A.5 caso de uso del usuario administrador

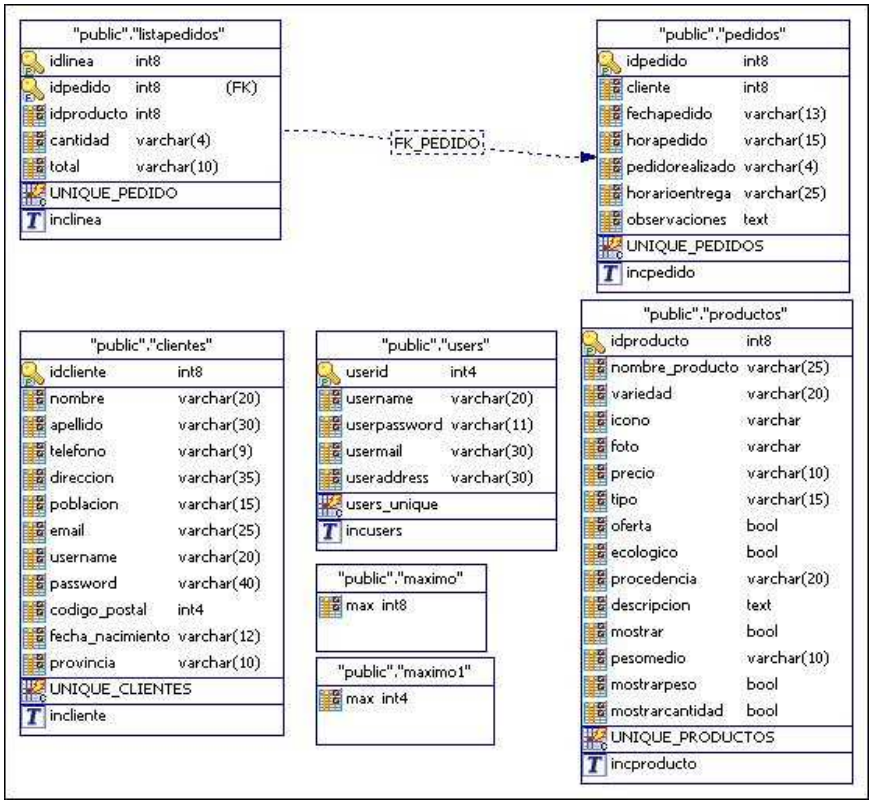


A.6 caso de uso gestión de pedidos

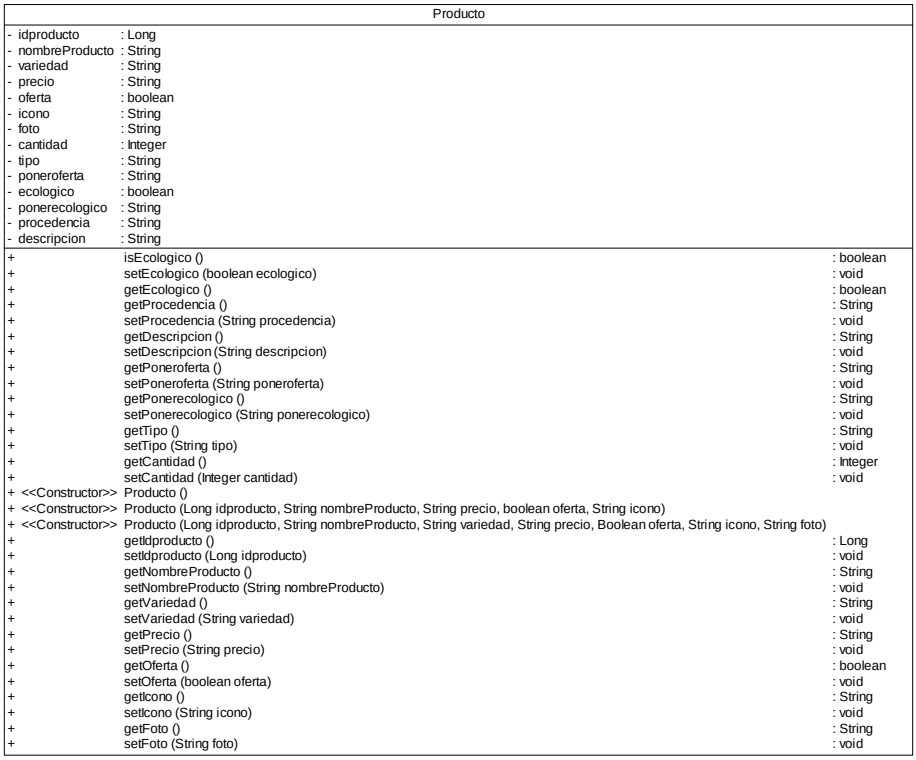
A.2 Diagramas de clases de dominio

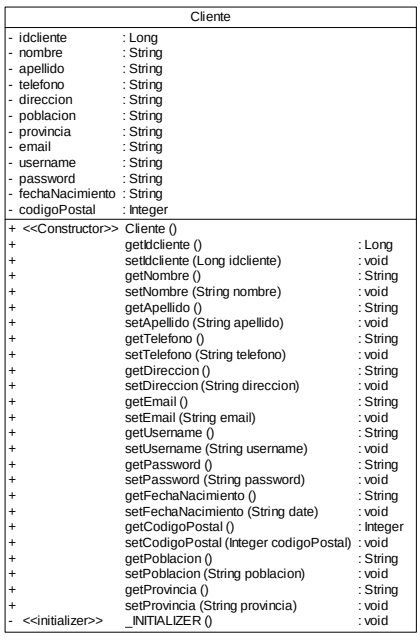
En este apartado se muestran los diagramas de clases obtenidos durante el desarrollo del proyecto.

Los diagramas están divididos en dos partes, la primera corresponde a la base de datos con sus tablas y sus respectivas relaciones (Figura B1), y la segunda parte corresponde a las clases y sus relaciones (Figura C1-C15)

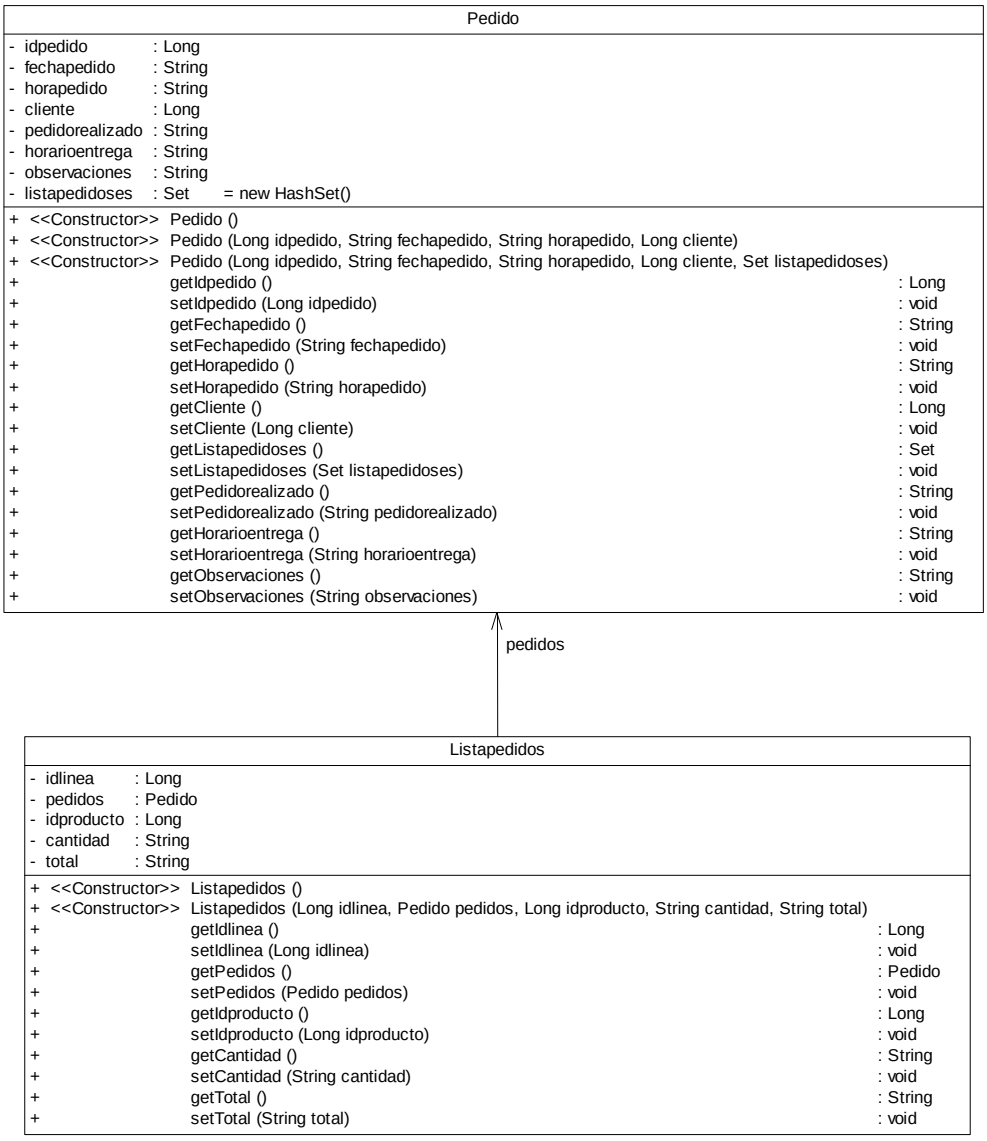


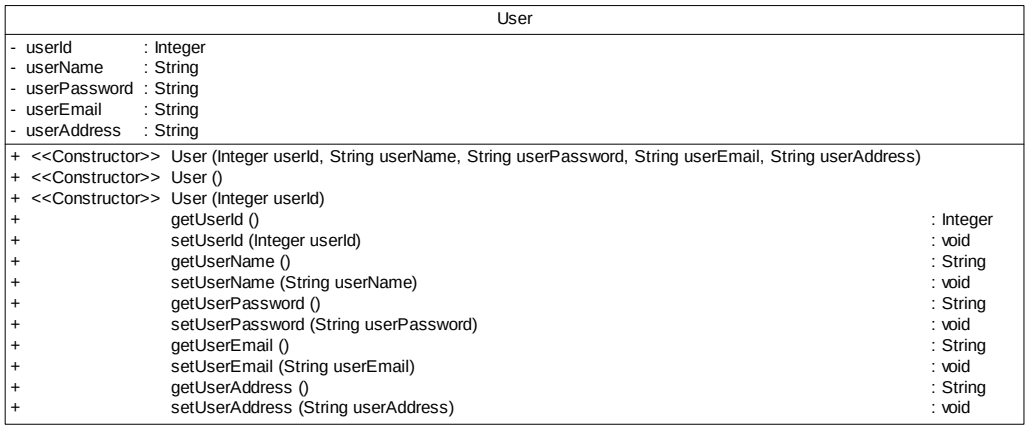
A.7 Diagrama de clases de la gestión de datos



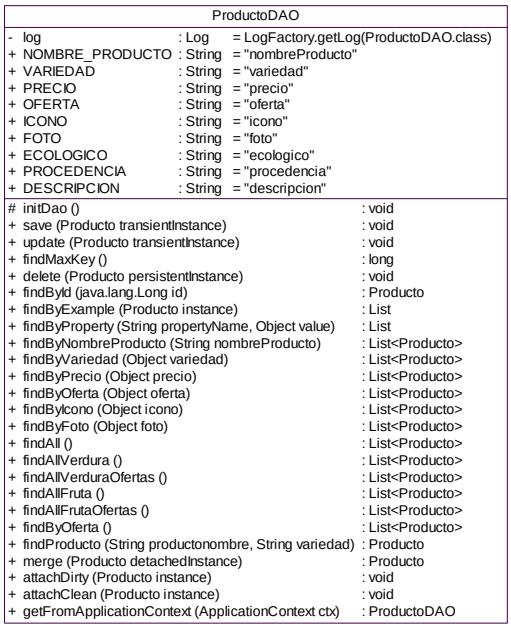


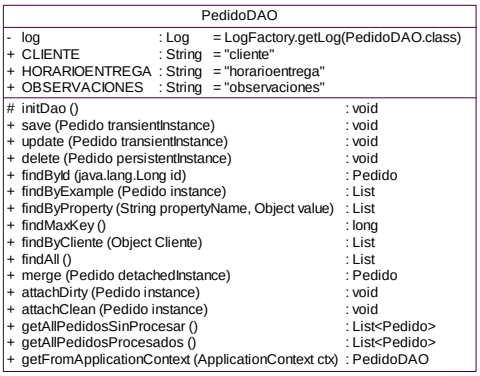
A.8 Clases perteneciente al paquete com.plantilla.pojo

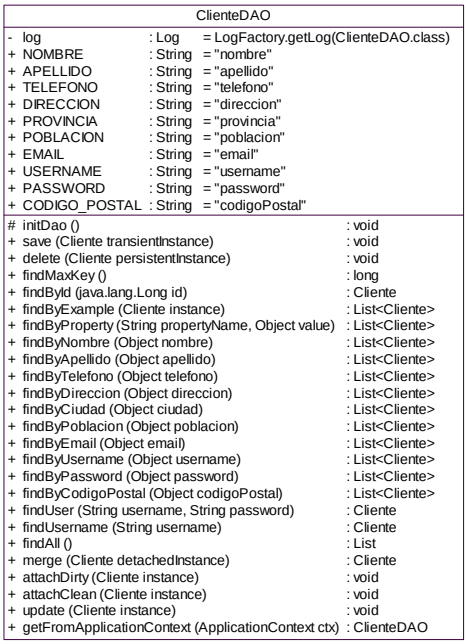


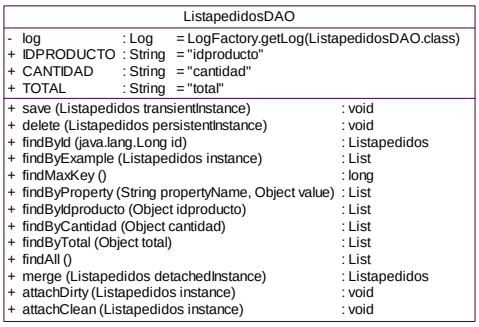


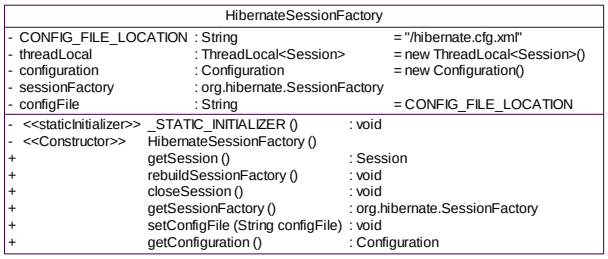
A.9 Clases perteneciente al paquete com.plantilla.pojo



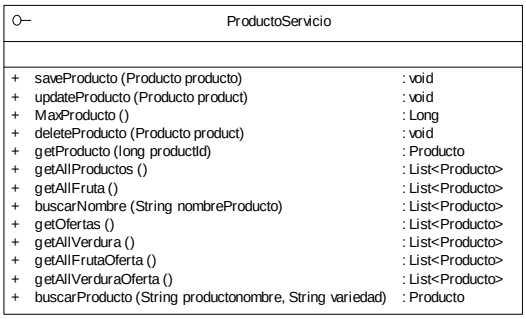


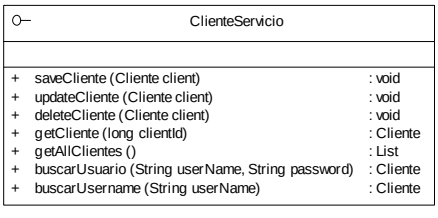


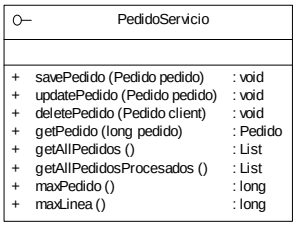




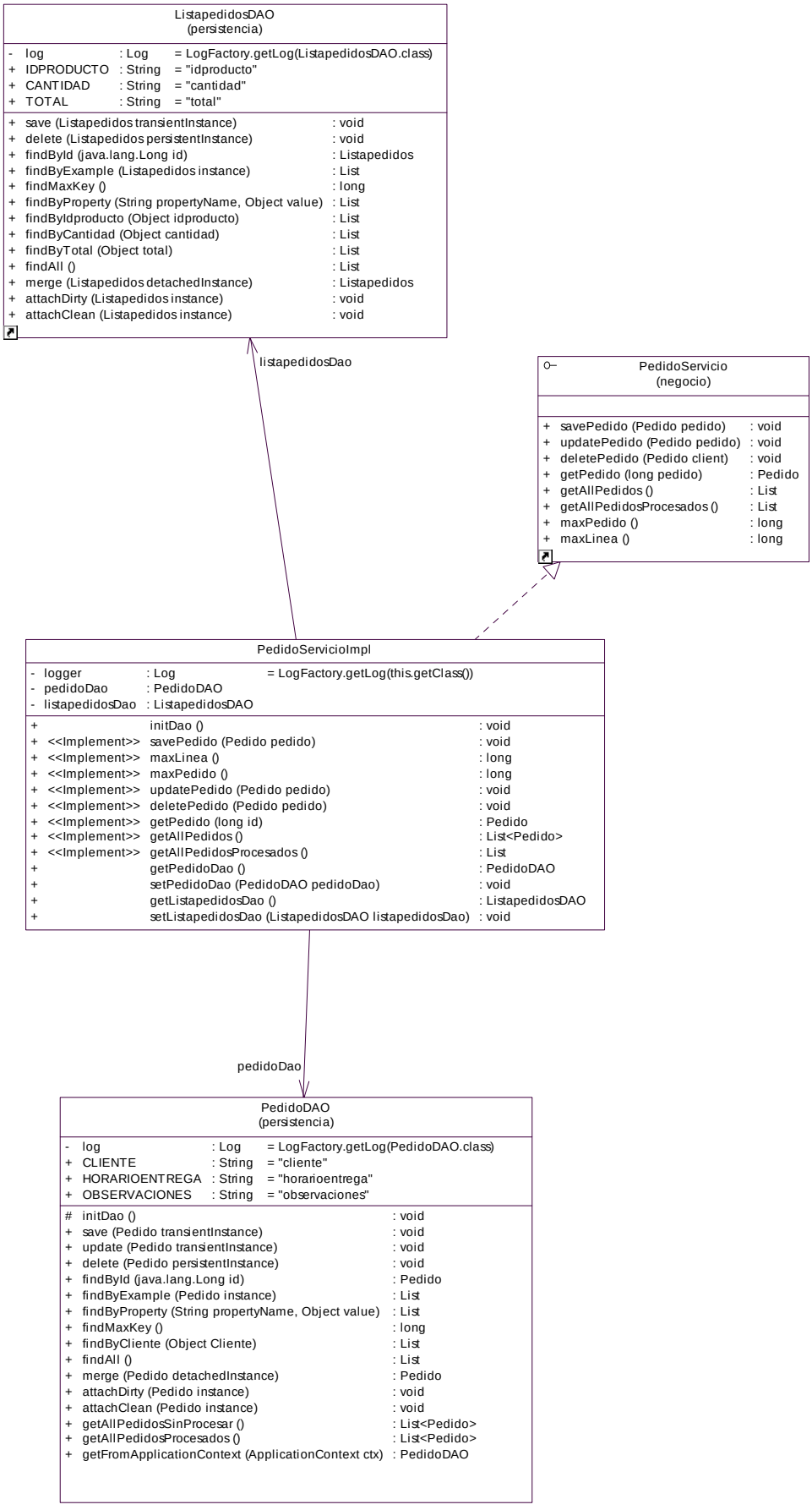
A.10 Clases perteneciente al paquete com.plantilla.persistencia



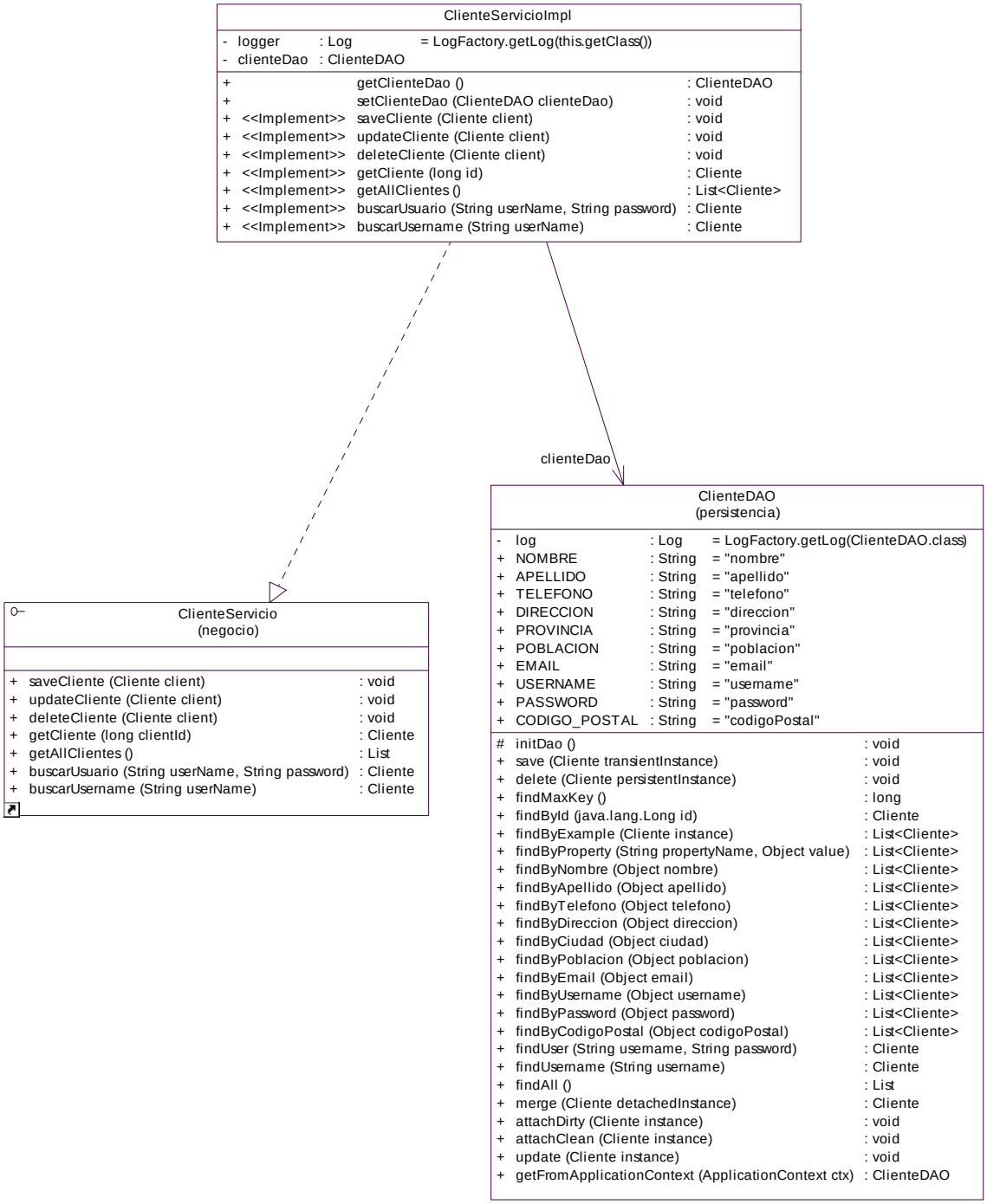




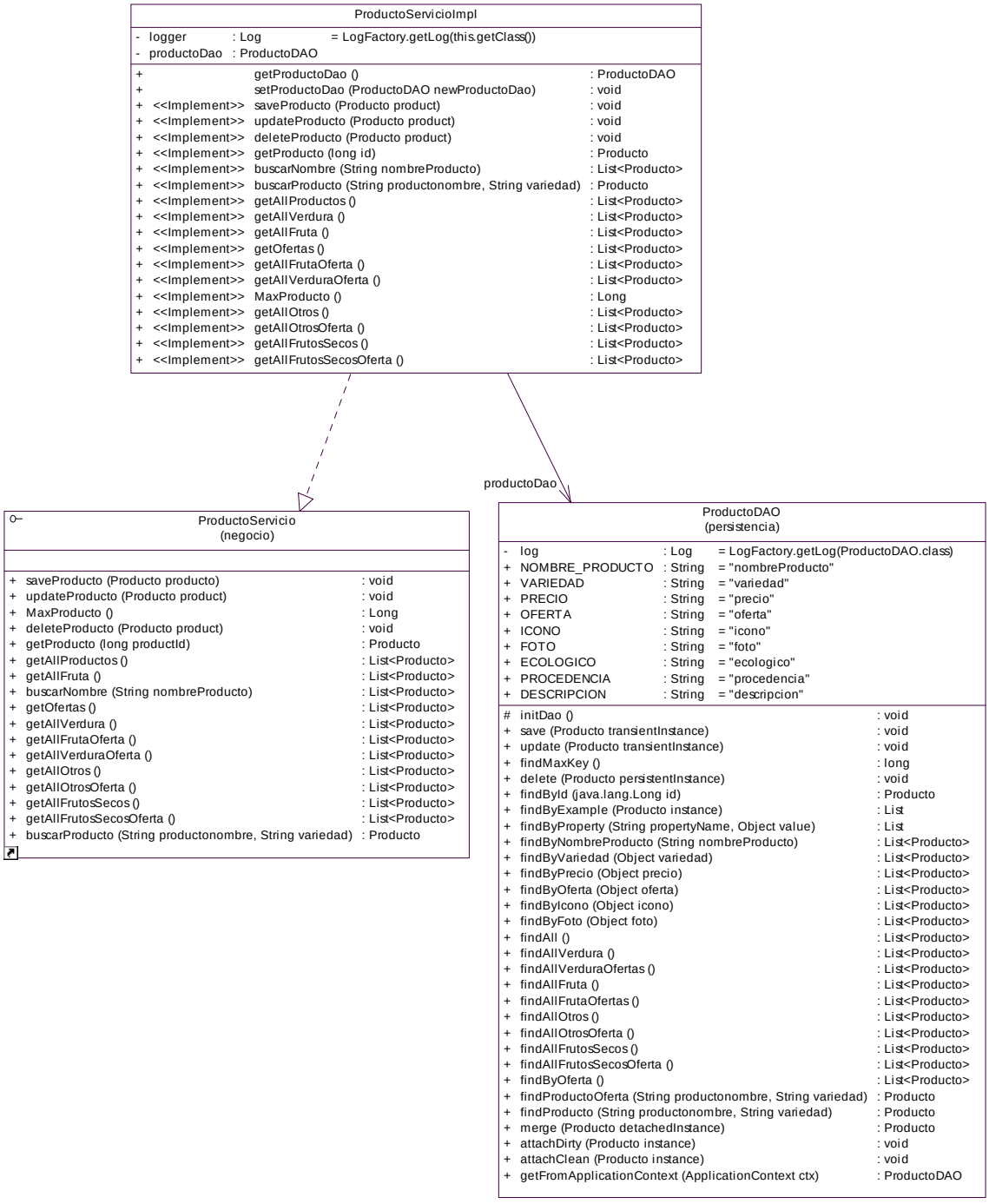
A.11 Clases perteneciente al paquete com.plantilla.negocio



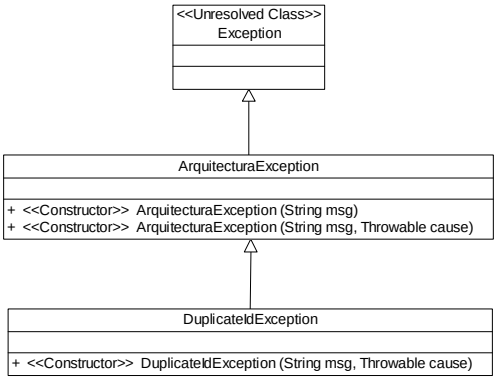
A.12 Clases perteneciente al paquete com.plantilla.negocio.implementacion



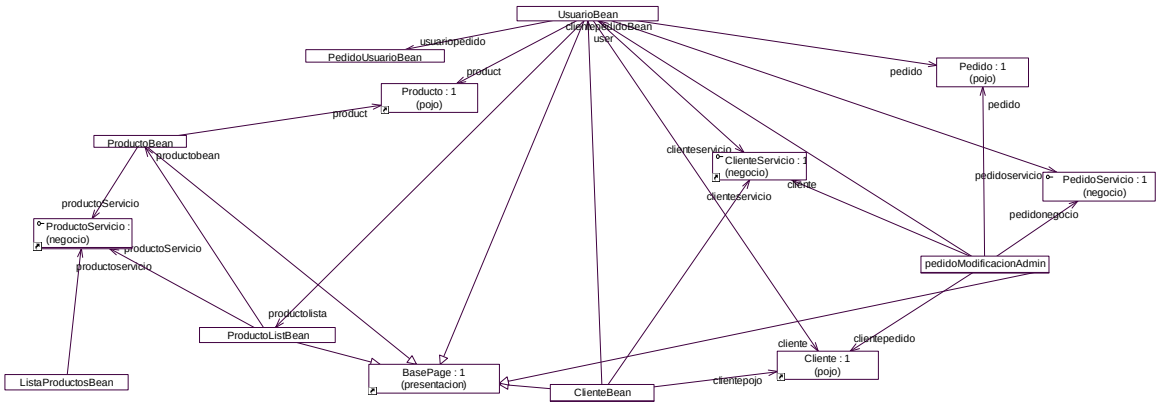
A.13 Clases perteneciente al paquete com.plantilla.negocio.implementacion



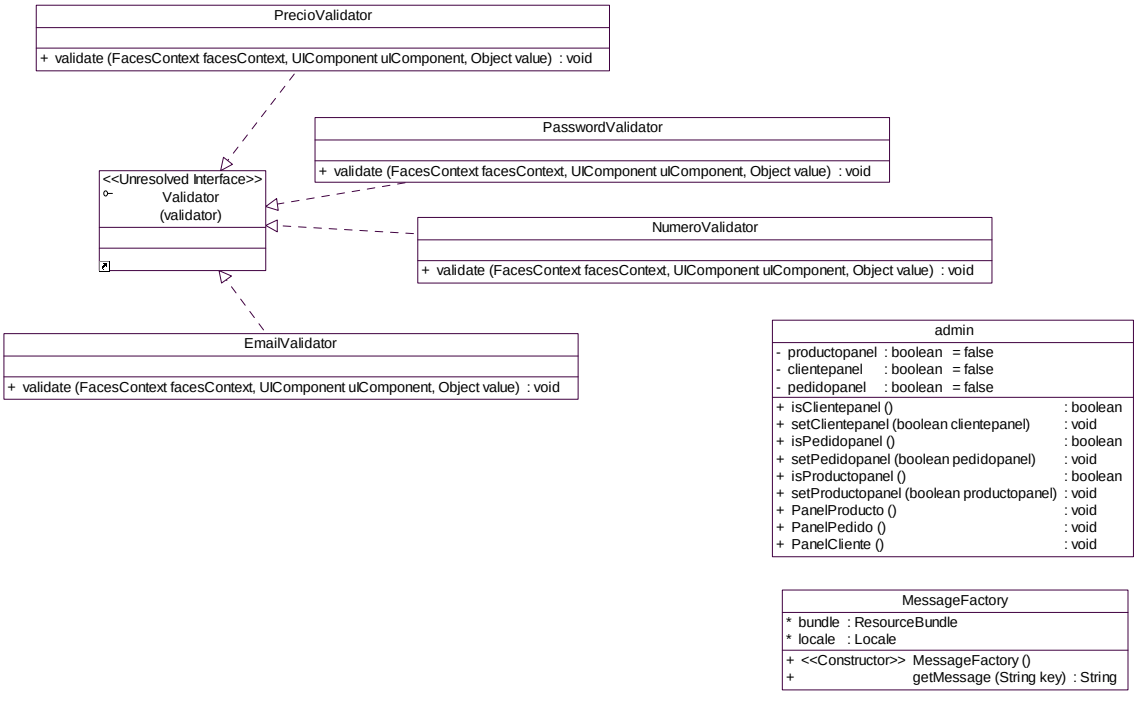
A.14 Clases perteneciente al paquete com.plantilla.negocio.implementacion



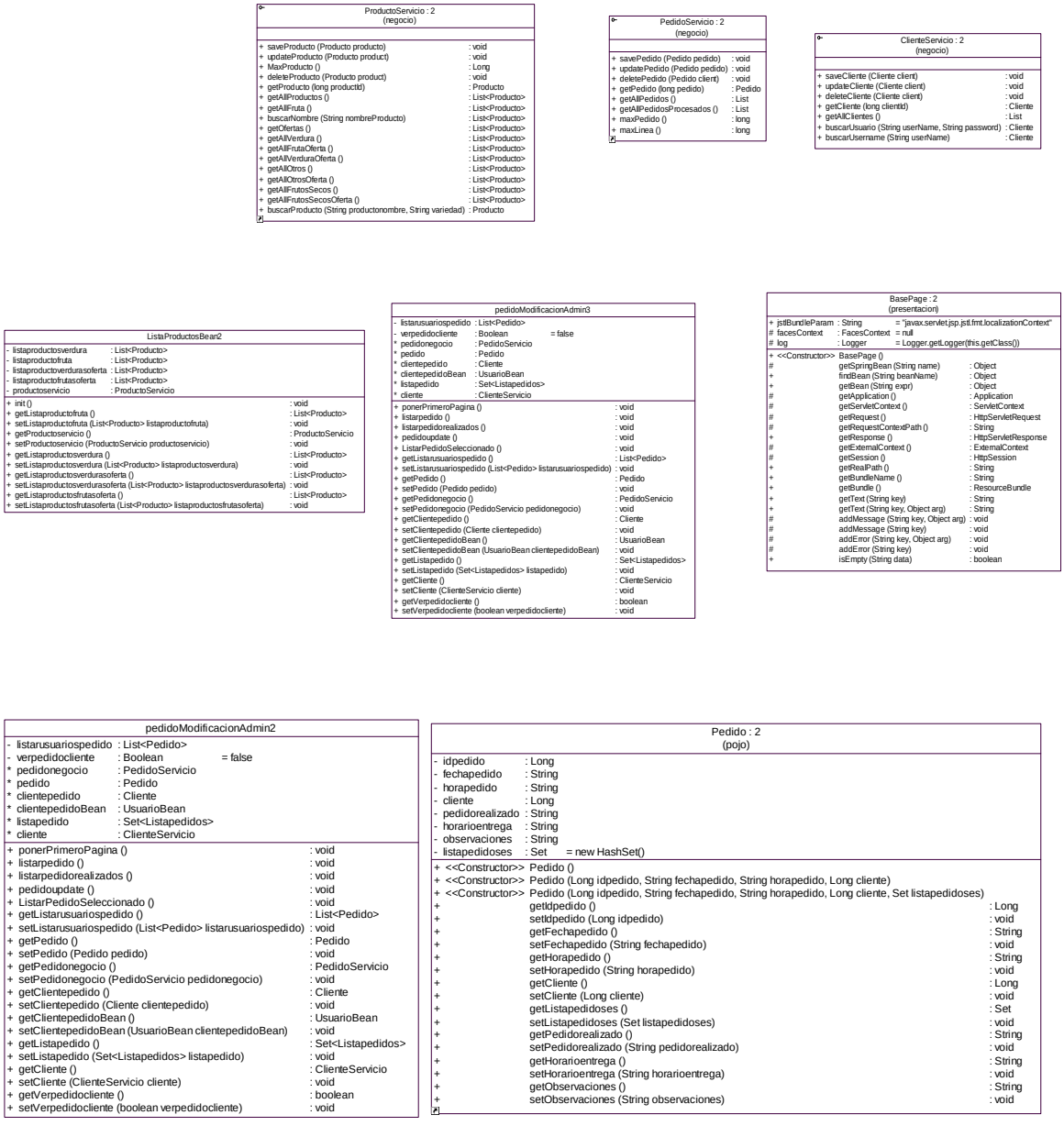
A.15 Clases perteneciente al paquete com.plantilla.excepciones



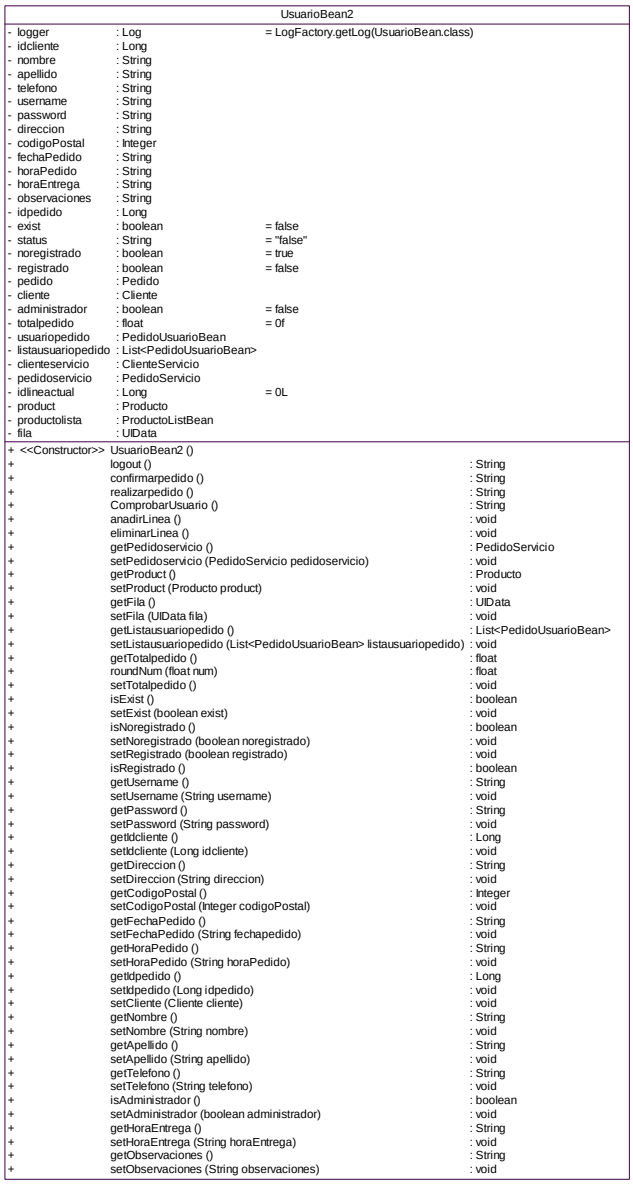
A.16 Modelo relación del paquete com.plantilla.presentacion



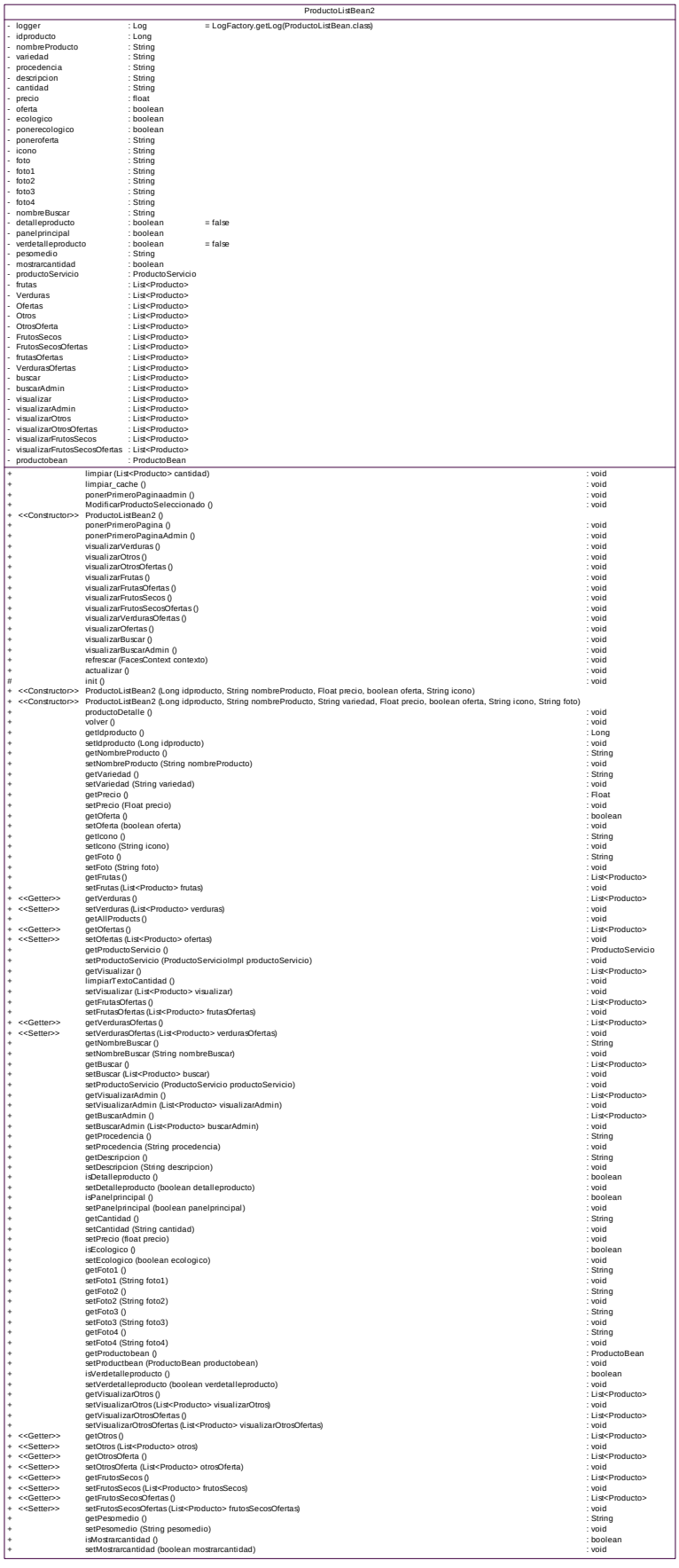
A.17 Clases perteneciente al paquete com.plantilla.presentacion



A.17 Clases perteneciente al paquete com.plantilla.presentacion

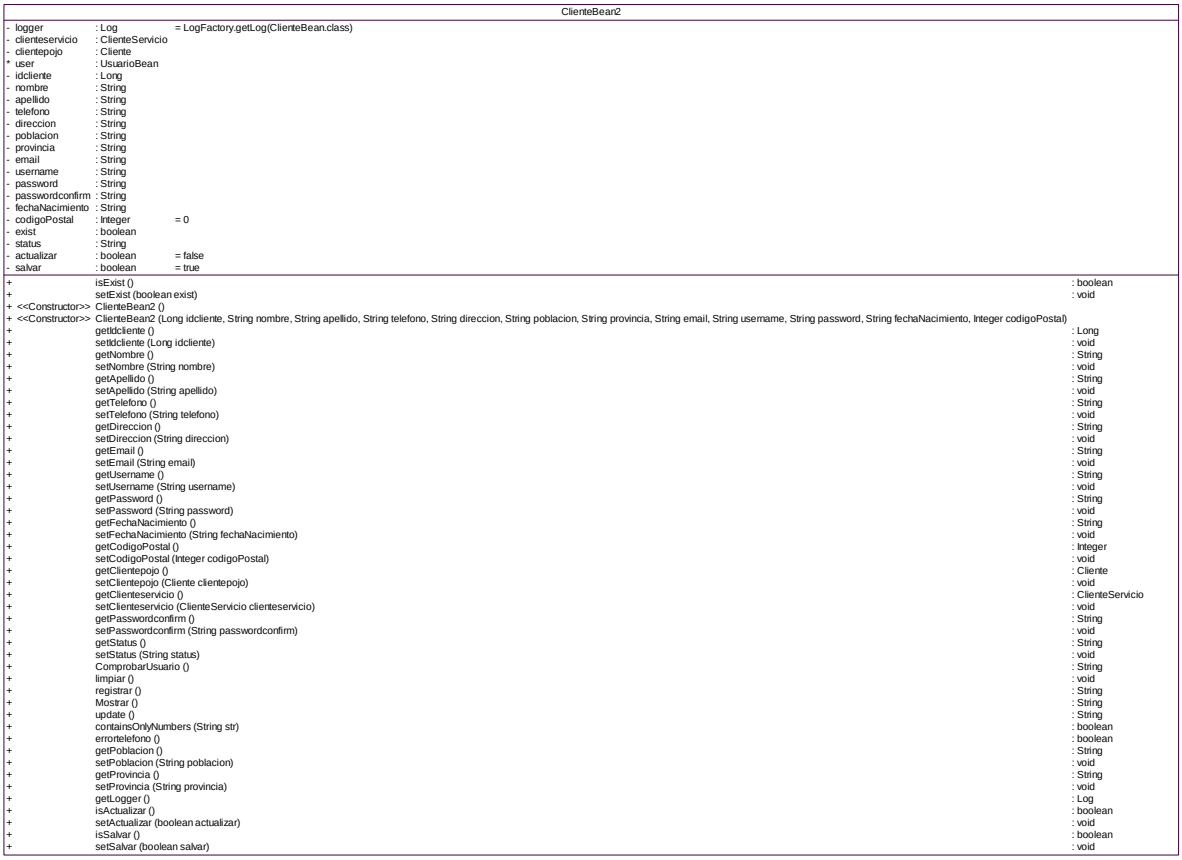


A.18 Clases perteneciente al paquete com.plantilla.presentacion



A.19 Clases perteneciente al paquete com.plantilla.presentacion

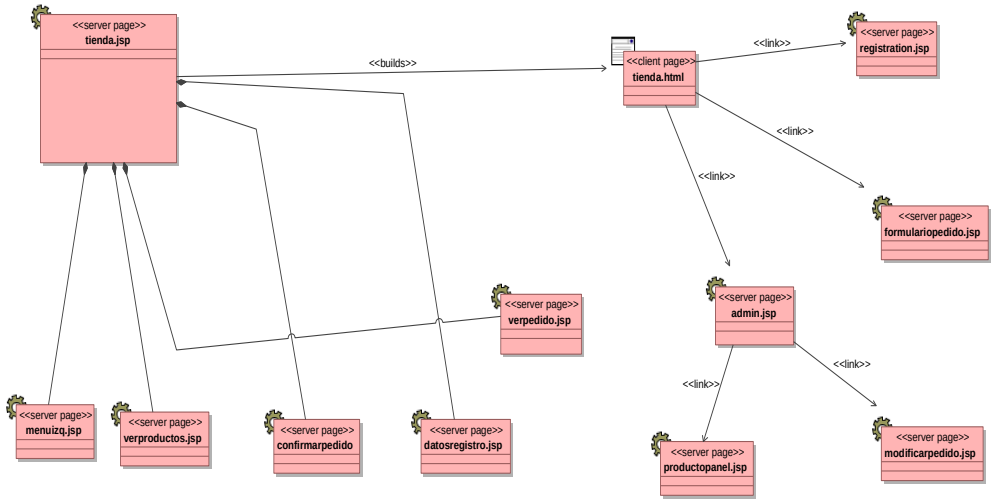
A.20 Clases perteneciente al paquete com.plantilla.presentacion



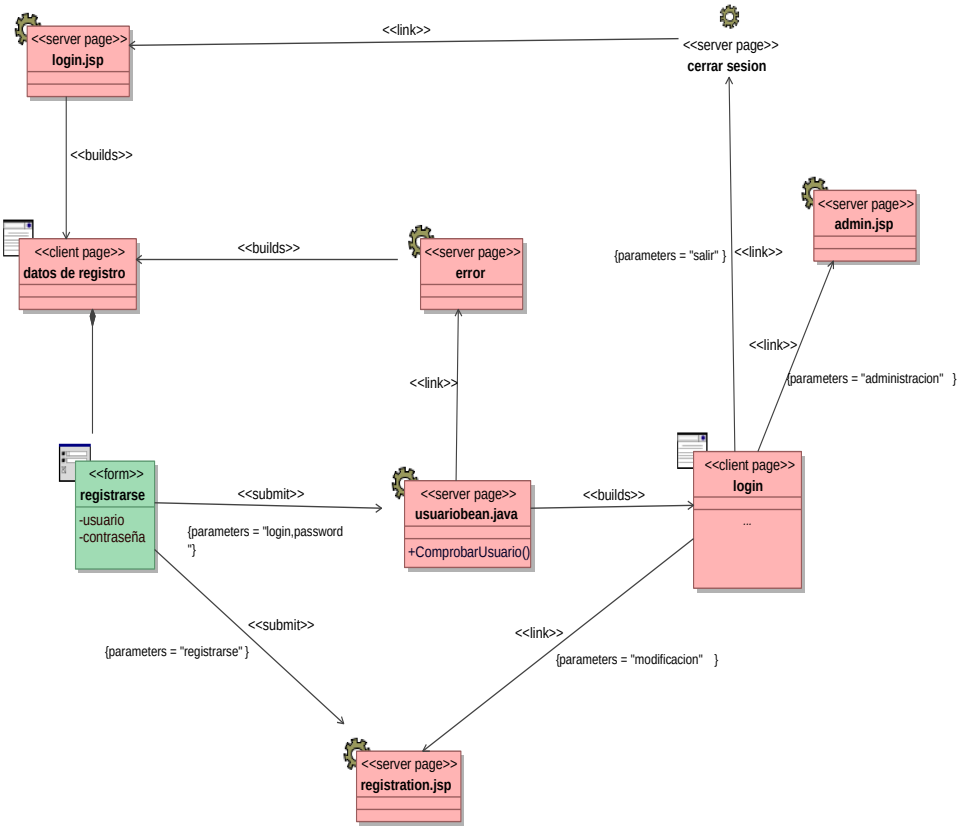
A.21 Clases perteneciente al paquete com.plantilla.presentacion

A.3 Diagramas de transición de estados

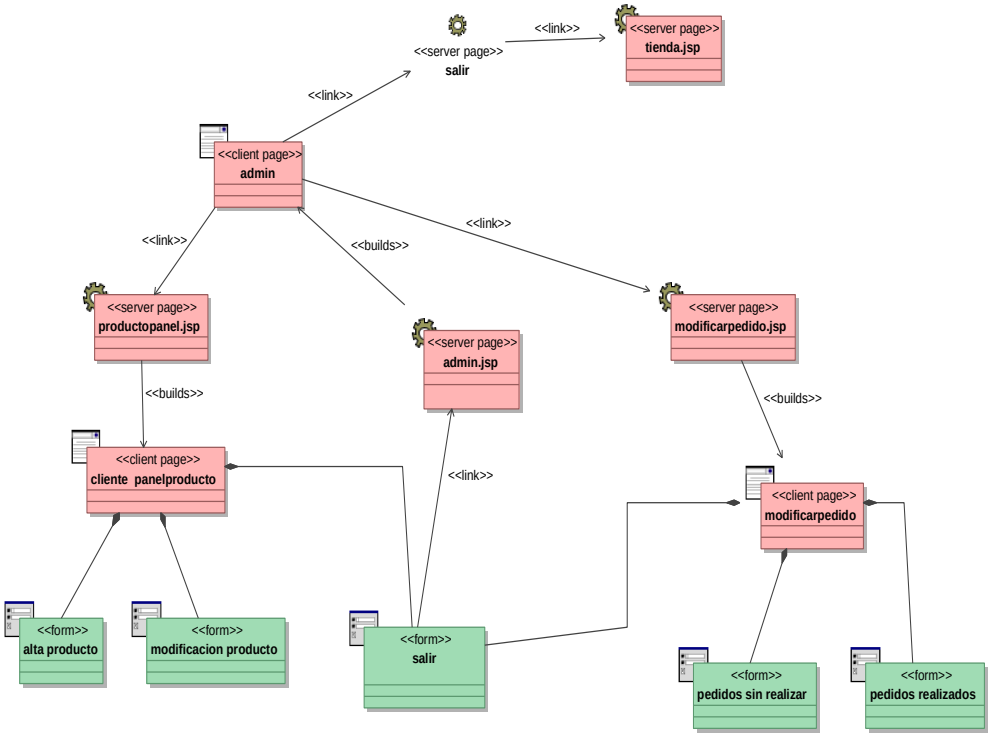
En el caso de los sistemas web, los bloques de construcción estándar que viene con UML no son suficientes para expresar la relación que existe entre los distintos elementos de una aplicación. Por ello, Conallen desarrolló una extensión UML, llamada WAE-UML (Web Application Extensión for UML, Extensión de Aplicación Web para UML), para el modelado de aplicaciones web.
Esta extensión es la que hemos utilizado para el modelado de las transiciones de las páginas web de la aplicación.



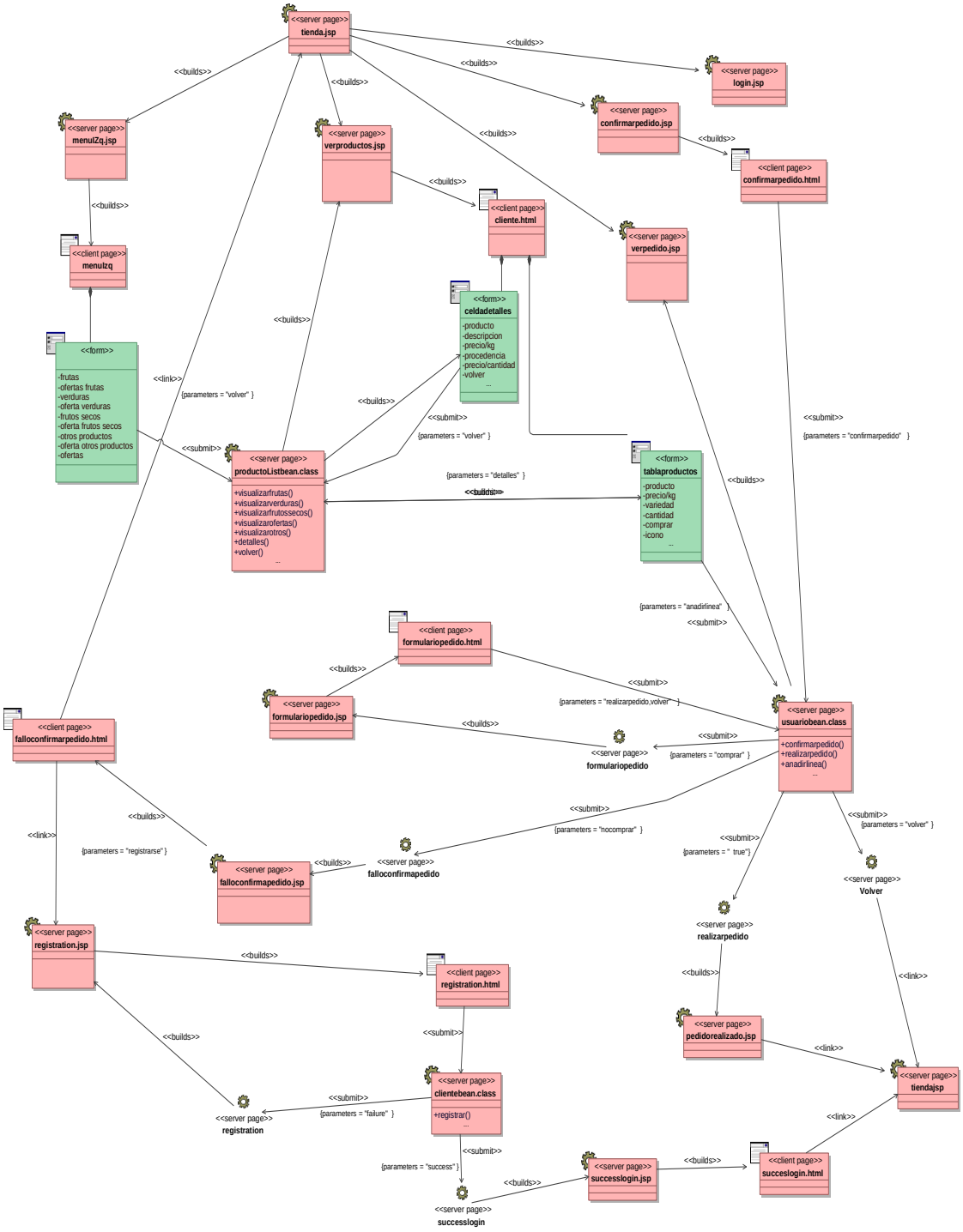
A.22 Esquema de relación de las distintas paginas web



A.23 Esquema de transición del registro de usuario



A.24 Esquema de transición del administrador



A.25 Esquema de transición de la página principal

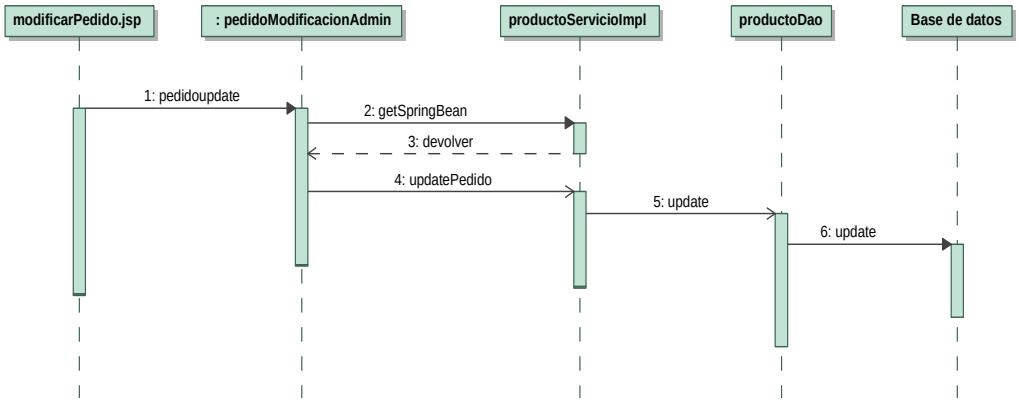
A.4 Diagramas de interacción

El diagrama de interacción, representa la forma en como un Cliente (Actor) u Objetos (Clases) se comunican entre si en petición a un evento. Esto implica recorrer toda la secuencia de llamadas, de donde se obtienen las responsabilidades claramente.

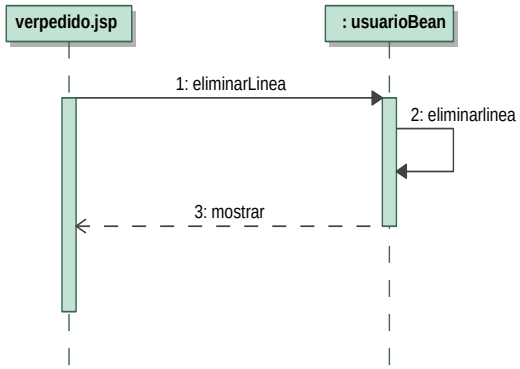
Existen dos tipos de mensajes: síncronos y asíncronos. Los mensajes síncronos se corresponden con llamadas a métodos del objeto que recibe el mensaje. El objeto que envía el mensaje queda bloqueado hasta que termina la llamada. Este tipo de mensajes se representan con flechas con la cabeza llena. Los mensajes asíncronos terminan inmediatamente, y crean un nuevo hilo de ejecución dentro de la secuencia. Se representan con flechas con la cabeza abierta.

También se representa la respuesta a un mensaje con una flecha discontinua.

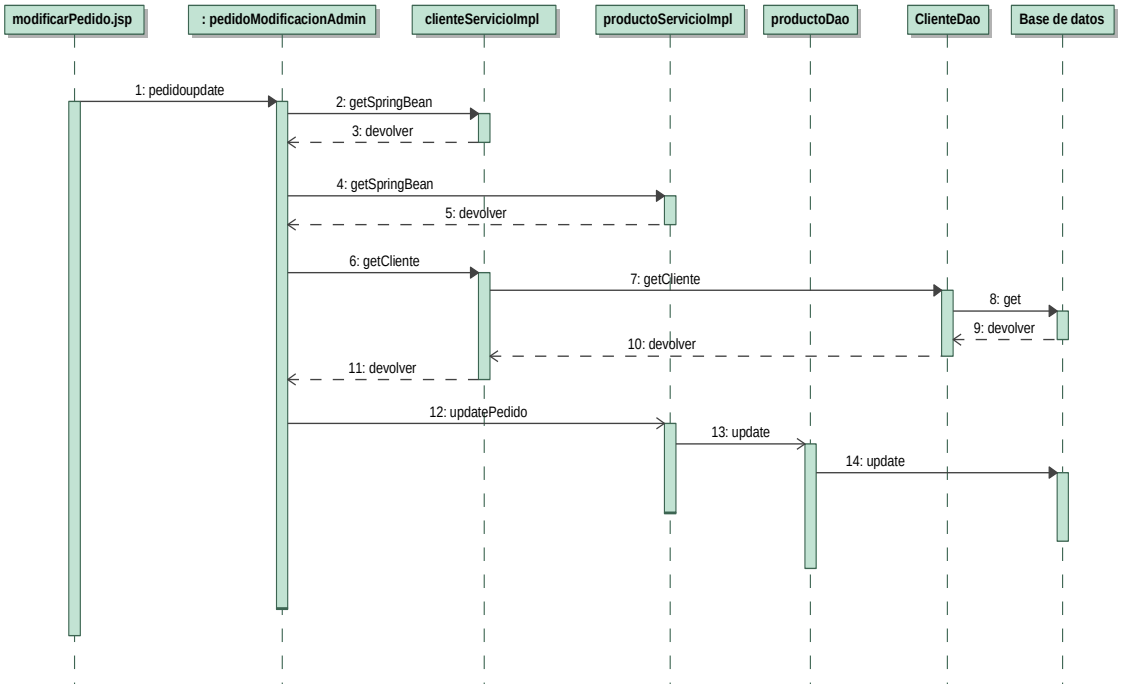
Los mensajes se dibujan cronológicamente desde la parte superior del diagrama a la parte inferior; la distribución horizontal de los objetos es arbitraria.



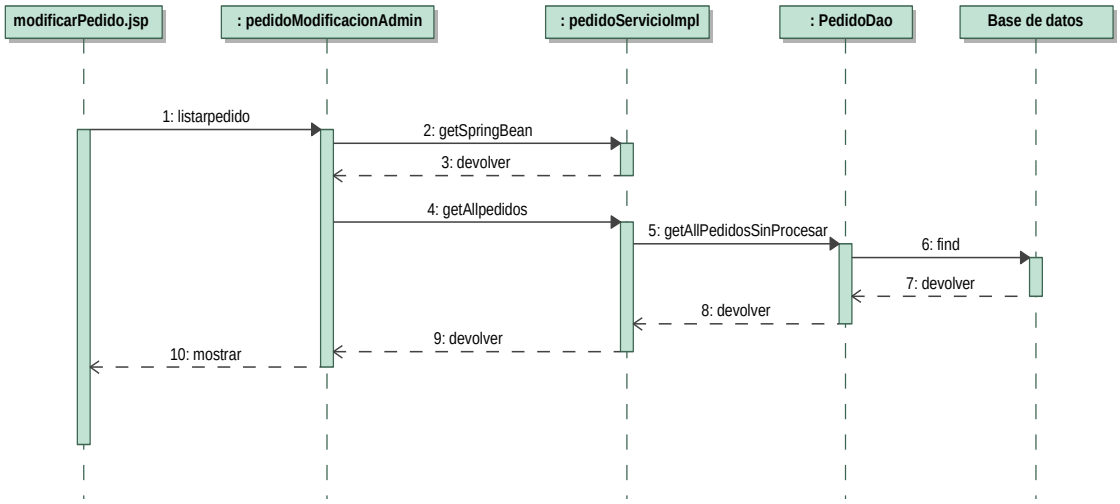
A.26 Diagrama de interacción de actualización de un pedido



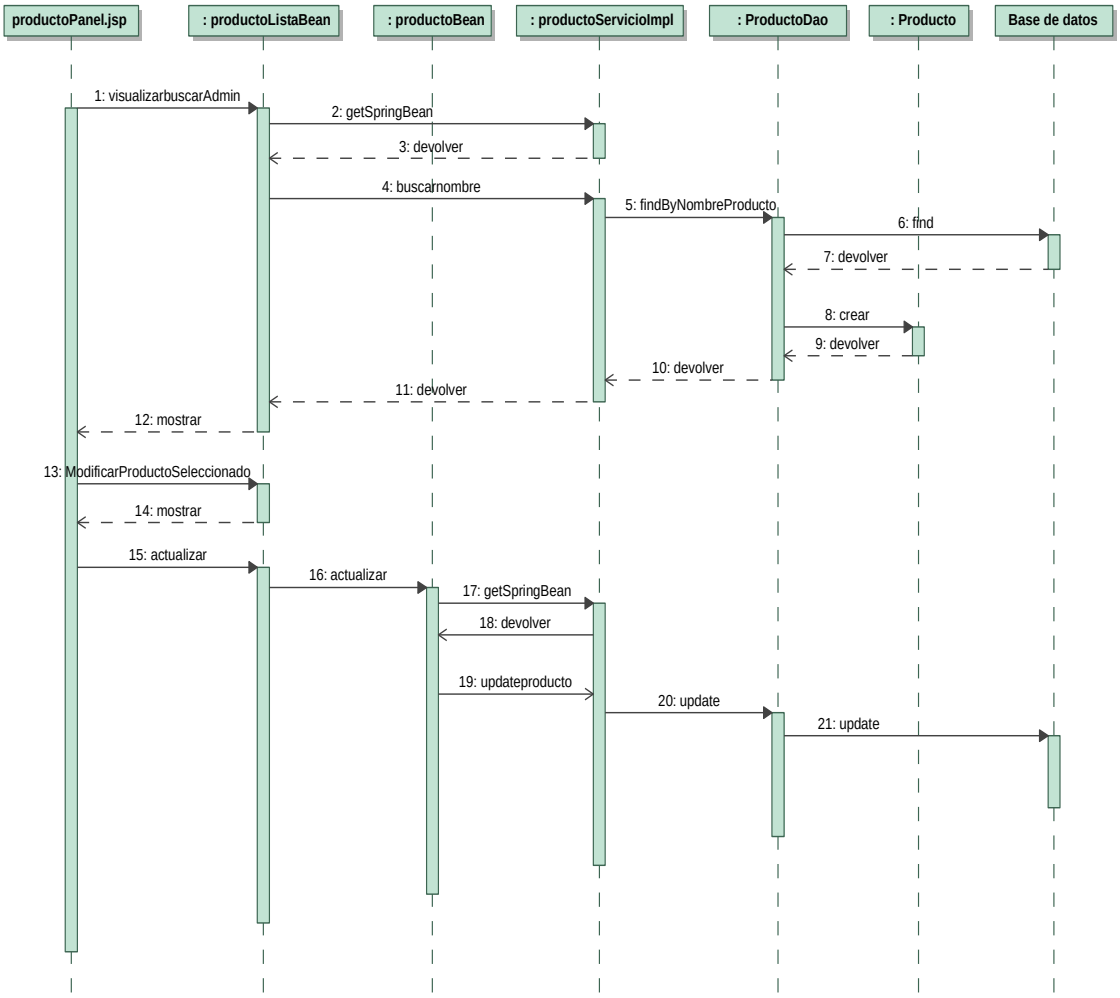
A.27 Diagrama de interacción de eliminación de un producto



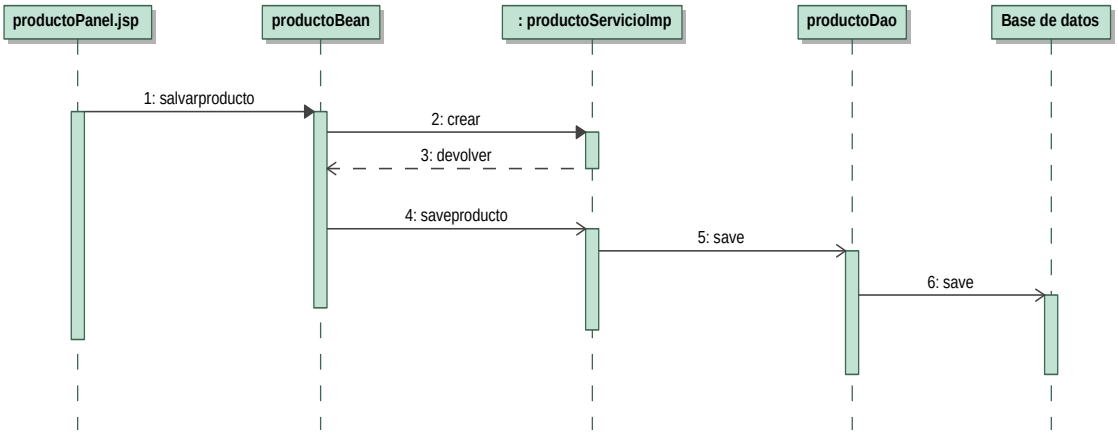
A.28 Diagrama de interacción de consulta de un pedido



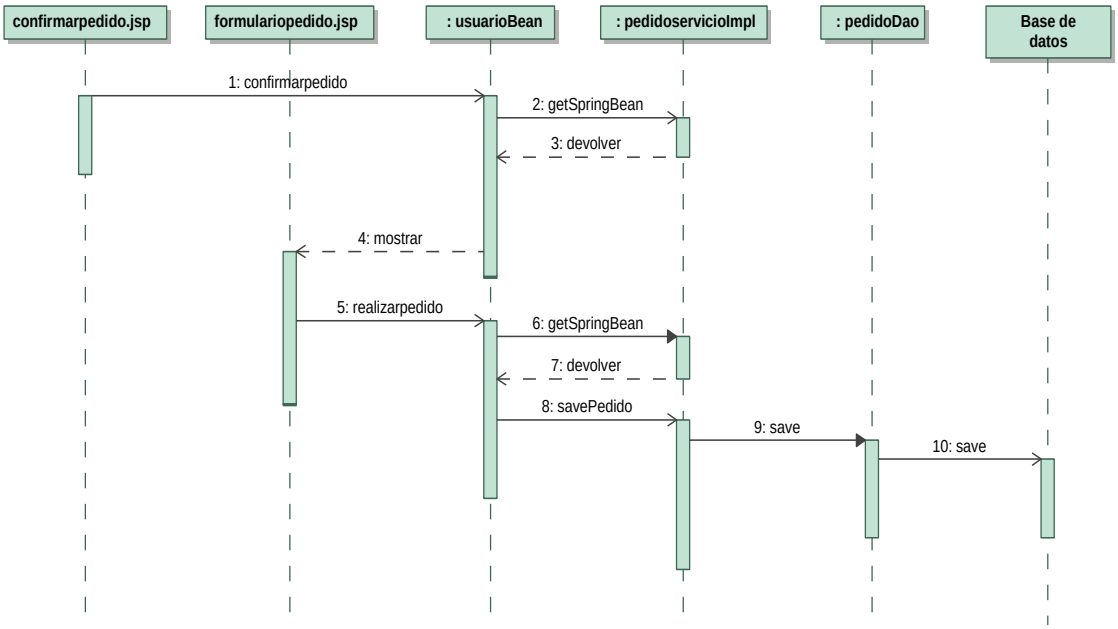
A.29 Diagrama de interacción de listado de los pedidos



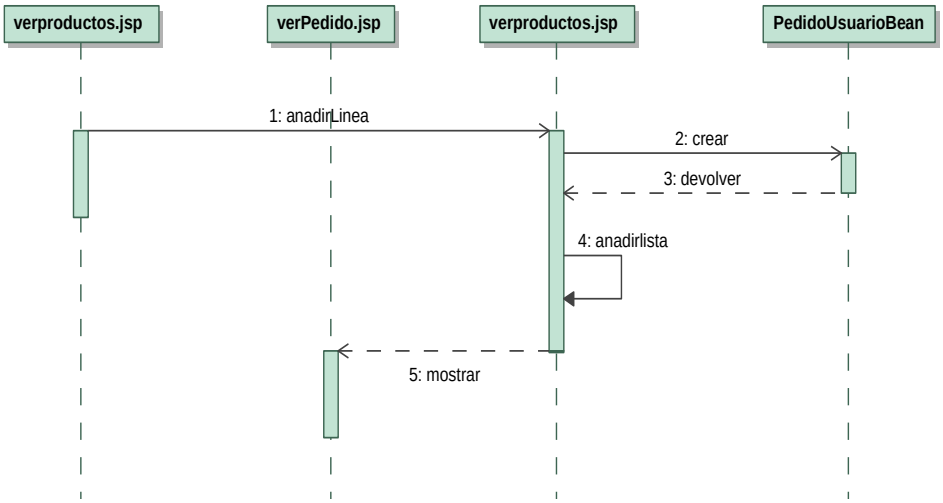
A.30 Diagrama de interacción de modificación de un producto



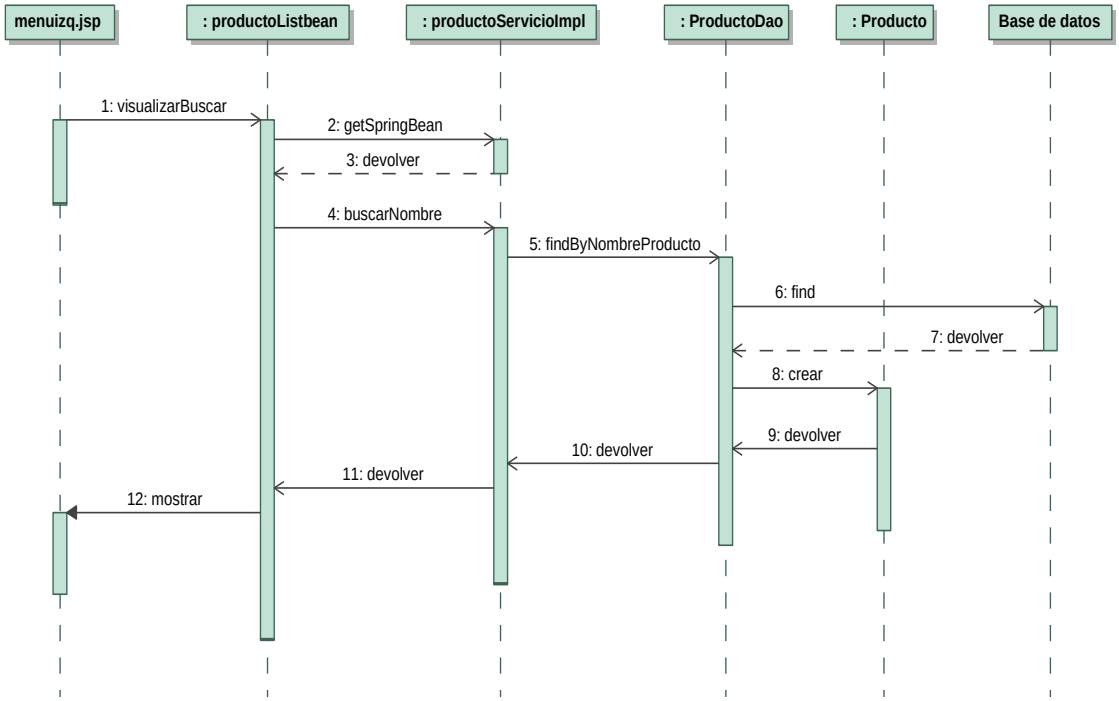
A.31 Diagrama de interacción de alta de un producto



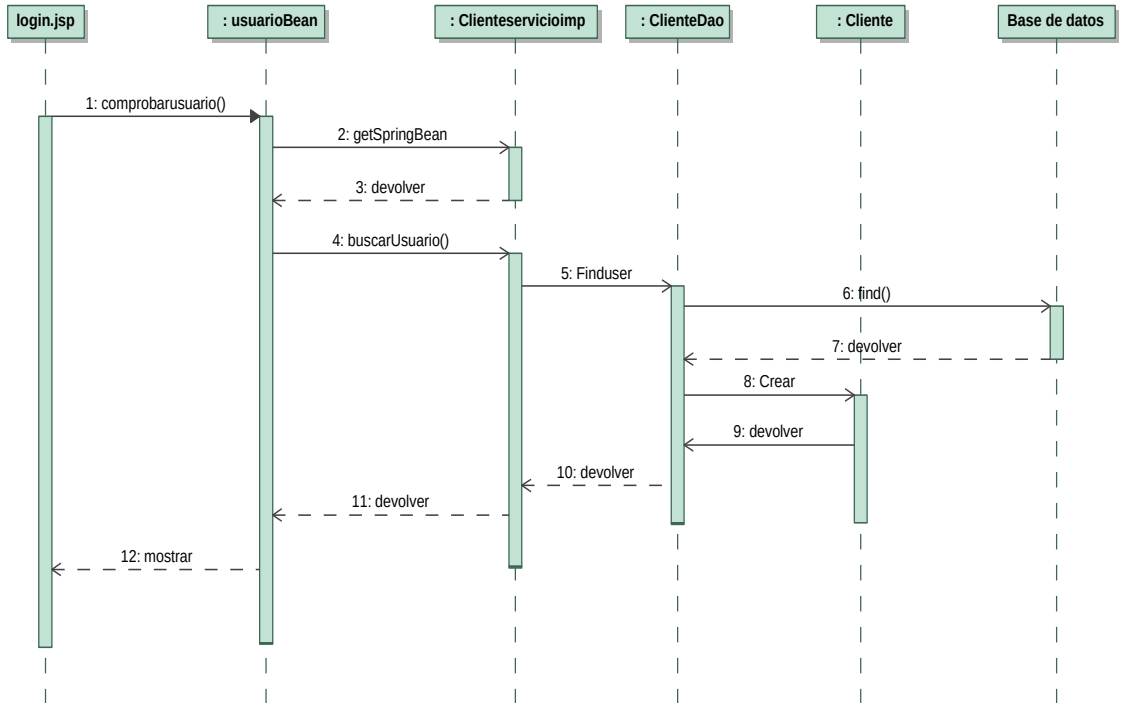
A.32 Diagrama de interacción de realización de un pedido



A.33 Diagrama de interacción de selección de un producto



A.34 Diagrama de interacción de busqueda de un producto



A.35 Diagrama de interacción de registro del usuario

Apéndice B. Glosario

Backing Beans son *JavaBeans* especializadas que colectan valores de componentes UI e implementan métodos que escuchen eventos (**listeners**). También pueden contener referencias a componentes UI

Bean: Un Bean es una clase que tiene un constructor público sin parámetros y "propiedades". Las propiedades exponen su estado al mundo exterior, y la manera de implementarlas es mediante métodos de acceso, conocidos como *getters* y *setters*. El *getter* es un método **público** de solo lectura que me devuelve el valor de la propiedad, y cuyo nombre tiene el prefijo "*get*". El setter, en cambio, es un método **público** que sirve para establecer el valor de la propiedad, y tiene prefijo "*set*".

Componente UI es un objeto mantenido en el servidor que provee funcionalidades para interactuar con el usuario final. Son *JavaBeans* con propiedades, métodos y eventos. Están organizados en una vista, que es un árbol de componentes usualmente mostrados como una página.

Enterprise JavaBean(EJB) proporcionan un modelo de componentes distribuido estándar del lado del servidor. El objetivo de los *EJBs* es dotar al programador de un modelo que le permita abstraerse de los problemas generales de una aplicación empresarial (concurrencia, transacciones, persistencia, seguridad...) para centrarse en el desarrollo de la lógica de negocio en sí. El hecho de estar basado en componentes permite que éstos sean flexibles y sobre todo reutilizables.

Los *EJBs* se disponen en un contenedor *EJB* dentro del servidor de aplicaciones. La especificación describe cómo el *EJB* interactúa con su contenedor y cómo el código cliente interactúa con la combinación del *EJB* y el contenedor.

Framework es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente suelen incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros para ayudar a desarrollar y unir los diferentes componentes de un proyecto

JavaBeans es una clase que debe obedecer ciertas reglas sobre nomenclatura de métodos, construcción, y comportamiento.

Reglas requeridas:

- La clase debe implementar la interfaz serializable (capaz de salvar persistentemente y de restablecer su estado).
- Debe tener un constructor sin argumentos.
- Sus propiedades deben ser accesibles mediante métodos *get* y *set* que siguen una convención de nomenclatura estándar.
- Debe contener determinados métodos de manejo de eventos.

Modelo Vista Controlador (MVC) 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. El patrón MVC se ve frecuentemente en aplicaciones Web, donde la vista es la página HTML y el código que provee de datos dinámicos a la página, el modelo es el Sistema de Gestión de Base de Datos y la Lógica de negocio y el controlador es el responsable de recibir los eventos de entrada desde la vista.

***POJO (acrónimo de Plain Old Java Object*)** es una sigla utilizada por programadores Java para enfatizar el uso de clases simples y que no dependen de un *framework* en especial. Este acrónimo surge como una reacción en el mundo Java a los *frameworks* cada vez más complejos, y que requieren un complicado andamiaje que esconde el problema que realmente se está modelando. En particular surge en oposición al modelo planteado por los estándares EJB anteriores al 3.0, en los que los "*Enterprise JavaBeans*" debían implementar interfaces especiales

***Prototype*:** El patrón de diseño *Prototype* (Prototipo), tiene como finalidad crear nuevos objetos duplicándolos, clonando una instancia creada previamente.

Este patrón es motivo donde en ciertos escenarios es preciso abstraer la lógica que decide que tipos de objetos utilizará una aplicación, de la lógica que luego usarán esos objetos en su ejecución. Los motivos de esta separación pueden ser variados, por ejemplo, puede ser que la aplicación deba basarse en alguna configuración o parámetro en tiempo de ejecución para decidir el tipo de objetos que se debe crear.

Singletons es un patrón de diseño (instancia única) diseñado para restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto.

Su intención consiste en garantizar que una clase sólo tenga una instancia y proporcionar un punto de acceso global a ella.

El patrón *singleton* se implementa creando en nuestra clase un método que crea una instancia del objeto sólo si todavía no existe alguna. Para asegurar que la clase no puede ser instanciada nuevamente se regula el alcance del constructor (con atributos como protegido o privado).

El patrón *singleton* provee una única instancia global gracias a que:

- La propia clase es responsable de crear la única instancia.
- Permite el acceso global a dicha instancia mediante un método de clase.
- Declara el constructor de clase como privado para que no sea instanciable directamente.

Etiqueta (o *tag*) es una marca con tipo que delimita una región en los lenguajes basados en XML.

Referencias

Referencias Web

[1] MyEclipse. Material educativo. Introducción a Hibernate en MyEclipse.
http://www.myeclipseide.com/documentation/quickstarts/hibernateintroduction/

[2] Hibernate. Documentación sobre Hibernate.
http://www.hibernate.org/5.html/

[3] Manual de Hibernate.
http://www.javahispano.org/contenidos/es/manual_hibernate/

[4] Integración Hibernate, Spring, JSF.
http://www.javajazzup.com/issue7/page51.shtml

[5] Documentación de Spring
http://www.springframework.org/documentation

[6] Documentación de Spring en español
http://www.springhispano.org/

[7] Integración de JSF, Spring e Hibernate para crear una Aplicación Web del Mundo Real.
http://www.programacion.net/tutorial/jap_jsfwork/

[8] Documentación de Sun Microsystems sobre JSF.
http://java.sun.com/javaee/javaserverfaces/

[9] JSF y Myfaces
http://www.coreservlets.com/JSF-Tutorial/

[10] Documentación y descarga de Myfaces.
http://myfaces.apache.org/

[11] Documentación y descarga de RichFaces
http://www.jboss.org/jbossrichfaces/

[12] Documentación y descarga del contenedor servlet Apache tomcat.
http://tomcat.apache.org/

Libros

[13] Patrick Peak ,Nick Heudecker, Editorial Manning, Java Persistence with Hibernate.

[14] Craig Walls Ryan Breidenbach, Editorial Manning, Spring in Action.

[15] Anil Hemrajani**,** Editorial Sams Publishing, Agile Java Development with Spring, Hibernate and Eclipse.

[16] Rod Johnson,Juergen Hoeller, Editorial Wrox, Professional Java Development with the Spring Framework.

[17] Bill Dudney, Jonathan Lehr, Editorial Wiley Publishing,Inc, Mastering JavaServer Faces.

[18] Kito D. Mann, Editorial Manning, Java Server Faces in Action.

[19] Craig Walls, Editorial Anaya, Spring