Essència

Catálogo de productos de perfumería utilizando Microsoft .NET

Estudiante: Bartolomé Oliver Goñalons
ETIG

Consultor: Jordi Ceballos Villach

Fecha de entrega: 11/06/2007

Resumen

El proyecto Essència, consiste en el diseño e implementación de un **catálogo de productos de perfumería** para una empresa del ramo.

Los objetivos son crear una aplicación Web ASP.NET con un tipo de interface de usuario muy similar a los encontrados en aplicaciones Windows. El diseño debe estar estructurado por capas independientes entre sí. Utilizar e integrar para el desarrollo del proyecto componentes externos. Explorar los servicios Web y que posibilidades tienen. Conseguir en el proyecto la independencia de la base de datos y finalmente investigar la utilización de los handler de ASP.NET. Los objetivos marcados tienen como principal razón de ser el crear una sólida base para ampliaciones futuras de este desarrollo. El método seguido es el clásico en cascada con los pasos típicos de análisis previo, análisis de requisitos con prototipo, diseño, programación y pruebas. El único paso que no se realizará es el de mantenimiento por la naturaleza de TFC. El resultado ha sido un producto creado con Microsoft C# 2.0, con un diseño orientado a objetos, dividido en cinco módulos, uno por capa: *Essencia* que contiene la interface de usuario de la Web, *Essencia.App* que contiene la familia de objetos denominados controladores y que se hacen cargo de todas las operaciones que puedan necesitar los interfaces de usuario, *Essencia.Negoci* que contiene los objetos del dominio y la definición de los interfaces a utilizar en el resto de la aplicación, *Essencia.SQLServer* que contiene la implementación de los interfaces de almacenamiento de datos para funcionar con NHibernate y finalmente *Essencia.Util* que contiene utilidades comunes al resto de módulos. Como conclusión indicar que se han cumplido satisfactoriamente todos los objetivos propuestos inicialmente.

Índice

Índice de figuras

1. Introducción

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

El punto de partida del proyecto es la realización de un catálogo de productos de perfumería utilizando para ello una interface de usuario tipo Web. La intención inicial no es crear una aplicación completa, sino más bien crear unos sólidos cimientos para posteriormente poder ampliarla con facilidad hasta llegar a un sistema mucho más complejo que el inicial. Otra premisa importante es crear una interface de usuario a través de la web que permita al usuario "olvidarse" dentro de lo que se pueda que está utilizando un navegador para acceder a la aplicación. Estas condiciones iníciales condicionan en cierta forma el diseño y la programación del proyecto ya que por regla general exigen del uso de tecnologías y recursos que quizás de otra forma no hubiera sido necesario utilizar.

El desarrollo de este proyecto se ha realizado utilizando Microsoft .NET 2.0 y el lenguaje C# para esta plataforma. La selección del C# como lenguaje de desarrollo viene motivada por su similitud con otros lenguajes utilizados por mi previamente (C++, Java, Delphi) y también por el hecho de que es el lenguaje referencia de la plataforma.

La aportación de este TFC es la creación de un catálogo de productos de perfumería totalmente funcional en cuyo diseño ha primado la idea de que sea fácilmente ampliable y que utiliza un buen número de tecnologías para logar sus objetivos. La infraestructura creada alrededor del proyecto es sumamente flexible y ampliable lo que permite utilizar fácilmente este desarrollo como punto de partida para otros proyectos más complejos. La utilización e integración de diversos módulos externos de código libre demuestran la facilidad con la que esto puede realizarse y de los beneficios que conlleva. Se demuestra que la creación de interfaces Web de usuario muy interactivos es hoy por hoy totalmente posible, aunque es verdad que con más esfuerzo que al hacer lo mismo para Windows.

1.2. Objetivos

Los objetivos propuestos a la hora de realizar este proyecto son muchos y variados pero en la mayoría destaca el interés en aprender el funcionamiento de nuevas tecnologías y sus posibles usos prácticos. Los objetivos más importantes son:

- Poner en práctica los conocimientos adquiridos durante los estudios: diseño, programación, bases de datos, etc.
- Conocer en profundidad el funcionamiento de Microsoft .NET en general y del C# en particular
- Crear una aplicación ASP.NET con un tipo de interface de usuario semejante a los encontrados en aplicaciones Windows
    - Aprender a utilizar JavaScript
    - Conocer el funcionamiento y posibilidades de AJAX
- Aprender el funcionamiento de los servicios web y sus posibilidades
- Averiguar que son los handlers del ASP.NET y sus usos
- Diseñar una arquitectura de software por capas desacopladas entre sí.
- Conseguir la independencia de la base de datos
- Explorar el funcionamiento y posibilidades de diversas librerías de código abierto

- Crear una sólida base para desarrollos futuros

1.3. Enfoque y método seguido

El enfoque y método seguido para el desarrollo de este proyecto es el que corresponde con el ciclo de vida clásico, también llamado en cascada o Waterfall. En la Figura 1 podemos ver las fases y la documentación que ha generado cada una de ellas. Debido al tiempo disponible para realizar el proyecto las fases han sido posiblemente más breves de lo que deberían haber sido y la documentación generada ha incluido quizás menos detalles de lo deseado. A pesar de ello el resultado final ha sido satisfactorio y el producto generado cumple los objetivos propuestos.

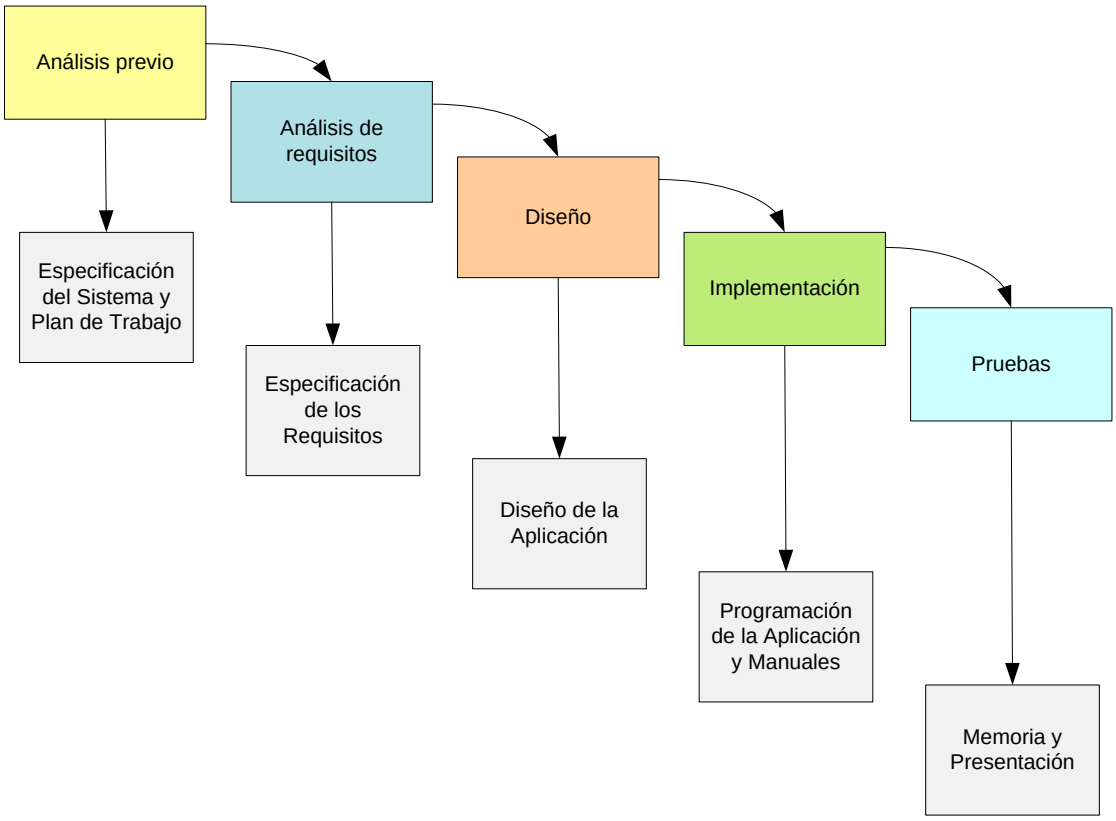


Figura 1: Fases de desarrollo y documentación generada

1.4. Planificación del proyecto

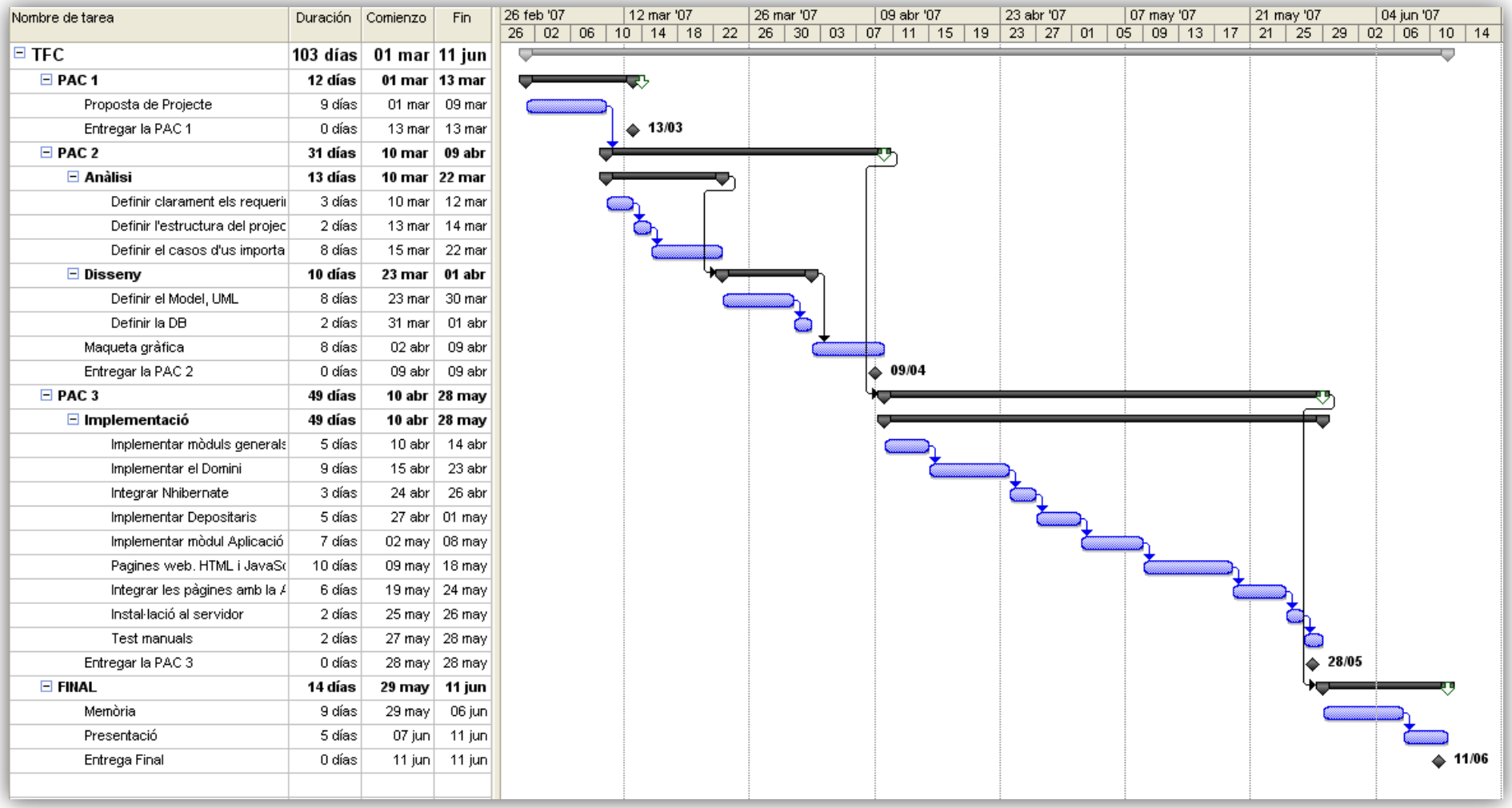
La planificación del proyecto ha venido condicionada por los requerimientos de la asignatura. En la Figura 2 podemos observar las fechas clave, la documentación entregada así como una pequeña descripción del trabajo realizado.

Fecha entrega	Documento	Descripción
PAC 1**: 13/04/2007**	Plan de trabajo	El documento presenta una descripción general del proyecto y su planificación
PAC 2**: 09/04/2007**	Análisis, Diseño y Prototipo	Documento de análisis de la aplicación que especifica lo que va a realizar el proyecto. Diseño tanto de clases como de la base de datos. Prototipo Web que enseña lo más detalladamente posible el funcionamiento previsto del proyecto
PAC 3**: 28/05/2007**	Implementación	Aplicación completa y funcional. Manual de instalación de la aplicación y de su funcionamiento.
PAC 4**: 11/06/2007**	Memoria y Presentación Virtual	Memoria del proyecto en la que se sintetiza el trabajo realizado. Presentación en PowerPoint explicando el proyecto.

Figura 2: Planificación del Proyecto

En la Figura 3 podemos ver el diagrama de Gantt definido inicialmente para la realización de este proyecto. En él se pueden observar con detalle la previsión inicial de la duración de cada una de las fases, duración siempre supeditada a la fecha de entrega de la correspondiente PAC, y el detalle del trabajo a realizar en cada una de ellas.

Por regla general se puede afirmar que la previsión se ha cumplido razonablemente bien salvo quizás en el momento de la implementación en la que motivos externos a este proyecto han obligado a hacer algunos reajustes.



Nombre de tarea	Duración	Comienzo	Fin
TFC	103 días	01 mar	11 jun
PAC 1	12 días	01 mar	13 mar
Proposta de Projecte	9 días	01 mar	09 mar
Entregar la PAC 1	0 días	13 mar	13 mar
PAC 2	31 días	10 mar	09 abr
Anàlisi	13 días	10 mar	22 mar
Definir clarament els requerii	3 días	10 mar	12 mar
Definir l'estructura del projec	2 días	13 mar	14 mar
Definir el casos d'us importa	8 días	15 mar	22 mar
Disseny	10 días	23 mar	01 abr
Definir el Model, UML	8 días	23 mar	30 mar
Definir la DB	2 días	31 mar	01 abr
Maqueta gràfica	8 días	02 abr	09 abr
Entregar la PAC 2	0 días	09 abr	09 abr
PAC 3	49 días	10 abr	28 may
Implementació	49 días	10 abr	28 may
Implementar mòduls generals	5 días	10 abr	14 abr
Implementar el Domini	9 días	15 abr	23 abr
Integrar Nhibernate	3 días	24 abr	26 abr
Implementar Depositaris	5 días	27 abr	01 may
Implementar mòdul Aplicació	7 días	02 may	08 may
Pagines web. HTML i JavaSc	10 días	09 may	18 may
Integrar les pàgines amb la A	6 días	19 may	24 may
Instal·lació al servidor	2 días	25 may	26 may
Test manuals	2 días	27 may	28 may
Entregar la PAC 3	0 días	28 may	28 may
FINAL	14 días	29 may	11 jun
Memòria	9 días	29 may	06 jun
Presentació	5 días	07 jun	11 jun
Entrega Final	0 días	11 jun	11 jun

Figura 3: Diagrama de Gantt del proyecto

1.5. Productos obtenidos

Los productos obtenidos durante la realización de este proyecto son los habituales durante el desarrollo siguiente el modelo clásico. En la Figura 4 podemos ver una tabla detallando todos los productos obtenidos junto con una breve explicación de los mismos.

Producto	Descripción
Plan de Trabajo	Descripción inicial del proyecto donde se presenta la descripción del proyecto junto con sus objetivos y requerimientos. Planificación temporal detallada del proyecto mediante un diagrama de Gantt incluyendo las fechas de entrega de las PACs.
Análisis	El documento del análisis presenta una descripción muy detallada del proyecto. Se explica cual debe ser la filosofía general del proyecto y detalla los procesos más importantes a implementar mediante diagramas de casos de uso.
Diseño	El documento de diseño hace especial mención a la arquitectura general de la aplicación. Contempla la utilización de diversas librerías de código abierto para así poder crear una infraestructura sólida para el proyecto. En este documento se incluye también el diseño de los objetos de la aplicación así como el de la base de datos.
Prototipo	Prototipo de la página Web que muestra detalladamente cual será el funcionamiento y el aspecto final de la aplicación.
Implementación	Implementación en C# del proyecto, incluyendo los diversos módulos y referencias externas ajenas a la instalación normal del framework de Microsoft. Está incluido todo el código fuente y también la Web compilada y lista para ser utilizada. Se incluye el fichero de la base de datos en SQL Server 2005 listo para ser usado bien con SQL Server 2005 o con la versión Express del mismo.
Manual	Manual detallado de cómo instalar la aplicación y de su puesta en funcionamiento. Este documento también incluye una guía básica del funcionamiento de la Web para así poder familiarizarse fácilmente con su filosofía de funcionamiento.
Memoria del proyecto	Este documento.

Presentación del proyecto	Presentación en PowerPoint complementaria a la memoria que muestra los puntos más destacables del proyecto dando una visión global del mismo.

Figura 4: Productos obtenidos durante el proyecto

1.6. Breve descripción del resto de capítulos de la memoria

A continuación un resumen del resto de capítulos de la memoria.

Capítulo 2: Análisis. Análisis del proyecto incluyendo una descripción del mismo. Se incluyen los diagramas de casos de uso más relevantes así como una descripción detallada de los casos de uso importantes. Finalmente se presenta un diagrama conceptual del proyecto en el que se puede ver claramente la relación entre las diversas entidades del mismo.

Capítulo 3: Diseño. Diseño del proyecto en el que se detalla detenidamente la arquitectura de la solución así como una explicación de las decisiones de diseño tomadas. Análisis de diversas librerías externas utilizadas en el proyecto que han permitido o facilitado el diseño actual. Diagrama de la arquitectura del hardware. Se muestran los diagramas de clases y la distribución de las mismas según el modelo de diseño por capas utilizado. Descripción de los interfaces y clases más relevantes. Se muestra el diseño del interface, su filosofía de uso y controles implementados para mejorar su funcionamiento. Diseño de la base de datos.

Capítulo 4: Conclusiones. Se exponen las conclusiones a las que se ha llegado durante el desarrollo del proyecto.

Capítulo 5: Líneas de desarrollo futuro. Se esbozan varias ideas para desarrollar este proyecto en un futuro.

Capítulo 6: Glosario. Glosario con los términos utilizados en esta memoria.

Capítulo 7: Bibliografía. Bibliografía consultada para el desarrollo del proyecto.

2. Análisis

2.1. Descripción del proyecto

Essència es un catalogo de productos de perfumería y similares. Incluye los módulos necesarios para dar de alta, modificar o eliminar los perfumes así como todos los datos que estos necesitan para darse de alta y clasificarse.

Todos los perfumes pertenecen a una familia según su tipo, así por ejemplo hay perfumes que pertenecen a la familia de los vaporizadores, otros son EDPs, otros son lotes, etc. Así mismo todos los perfumes pertenecen a una línea. Por líneas entendemos productos que pertenecen a un mismo fabricante y que comparten un nombre comercial, por ejemplo “Agua de Rosas”. Bajo este nombre podemos encontrar vaporizadores de 120 ml, de 60 ml, promocionales, etc. Las líneas y las familias también dispondrán de mantenimientos para así poder darlos de alta, modificarlo o darlos de baja si ya no nos sirven.

Otro de los módulos incluidos en el proyecto es una gestión simplificada del stock de los perfumes. Cada perfume tendrá asociado un stock que a través del módulo correspondiente se podrá modificar. Si en algún momento un perfume llega a un nivel de stock por debajo del mínimo indicado para ese perfume se podrá realizar un pedido automático a una central de compras utilizando un servicio Web que pone a nuestra disposición dicha central de compras. Las unidades a pedir del perfume serán las necesarias para llegar al stock óptimo del mismo.

Se incluye la posibilidad de realizar consultas de los datos del programa, concretamente se podrán listar los perfumes filtrando opcionalmente por la familia o la línea, se podrán listar las familias y líneas y también los movimientos de stock ocurridos entre dos fechas dadas por el usuario. Todos los listados tendrán la posibilidad de exportarse como PDFs.

Para facilitar el control sobre la evolución de los precios de los perfumes y su nivel de stock se dispone de un módulo que muestra una gráfica con la evolución de estas variables entre dos fechas.

Se distinguen tres tipos de usuarios, los administradores con acceso a todo el programa, los usuarios normales con acceso a todo menos a las áreas de creación de usuarios y los usuarios de sólo lectura que solamente podrán consultar los datos.

La aplicación dispone de un sencillo servicio Web que permite descargar información básica de los perfumes.

2.2. Diagramas de casos de uso

El proyecto contiene un gran número de diagramas de casos de uso y a continuación se mostraran los más relevantes así como una descripción de los casos de uso más importantes.

2.2.1. Visualización y mantenimiento de los perfumes

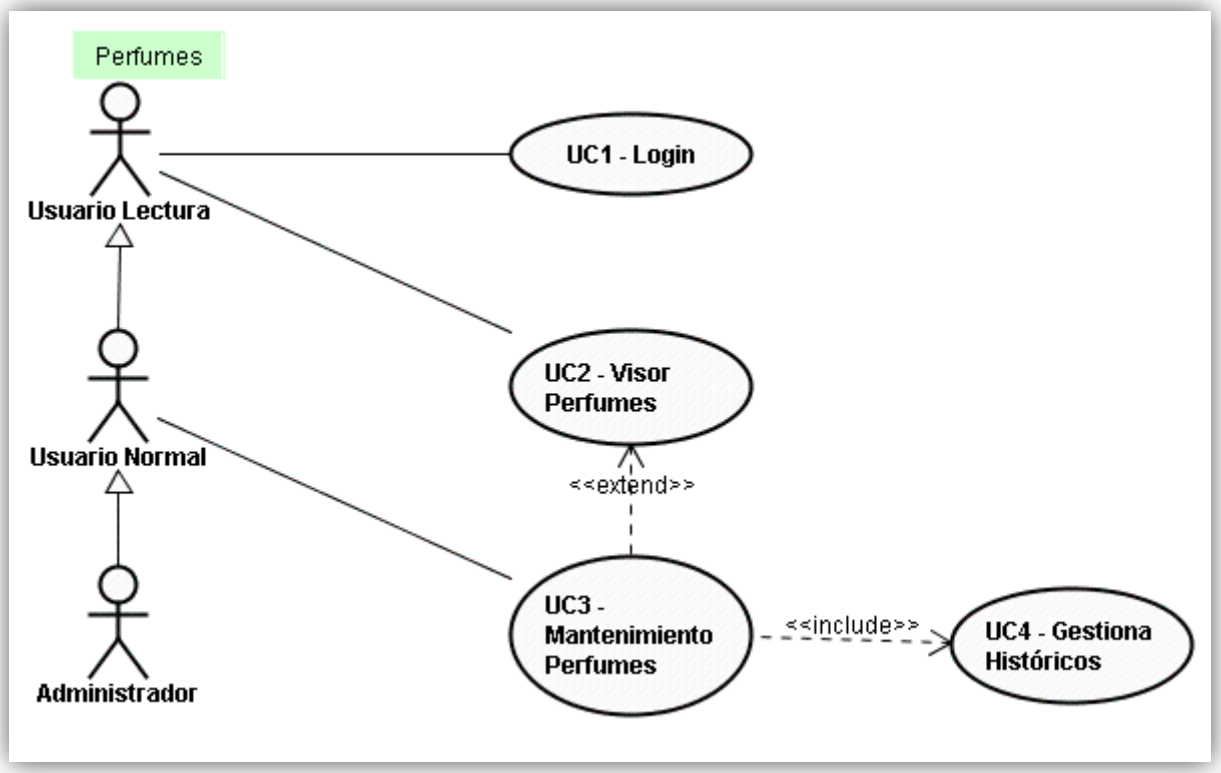


Figura 5: Diagrama de casos de uso. Visualización y mantenimiento de los perfumes

2.2.2. Visualización y mantenimiento de las líneas

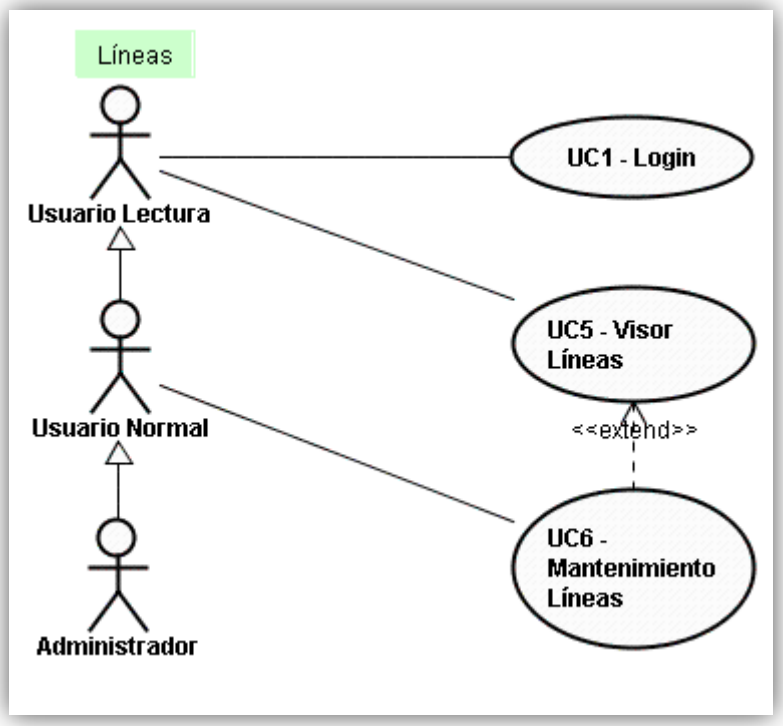


Figura 6: Diagrama de casos de uso. Visualización y mantenimiento de las líneas

2.2.3. Visualización y mantenimiento de las familias

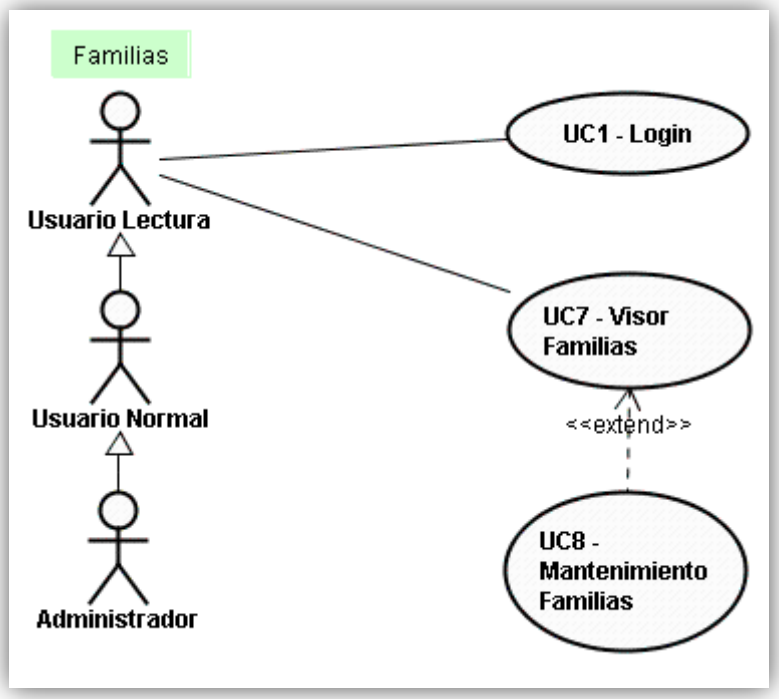


Figura 7: Diagrama de casos de uso. Visualización y mantenimiento de las familias

2.2.4. Existencias

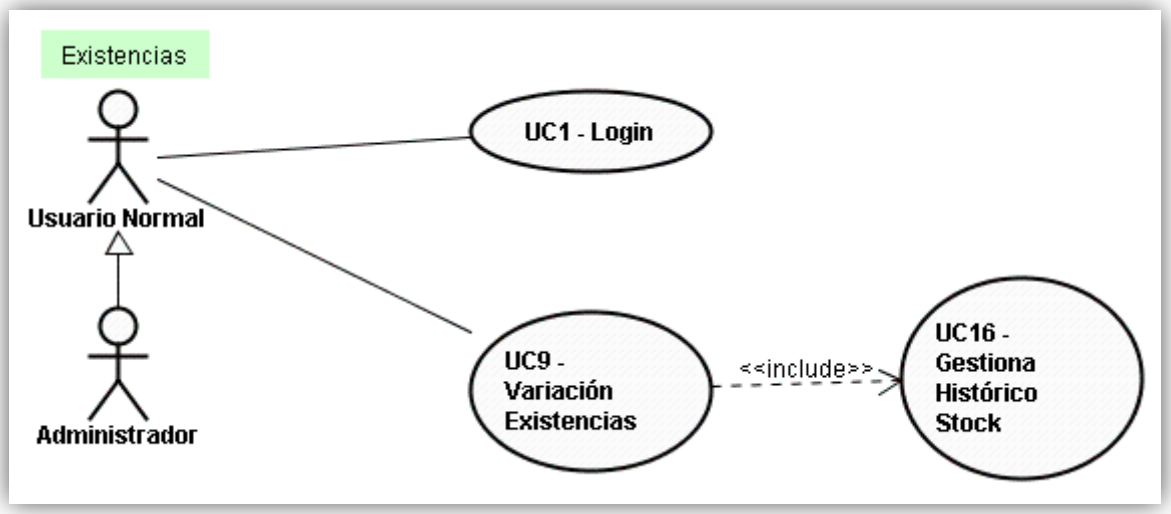


Figura 8: Diagrama de casos de uso. Existencias

2.2.5. Estadísticas

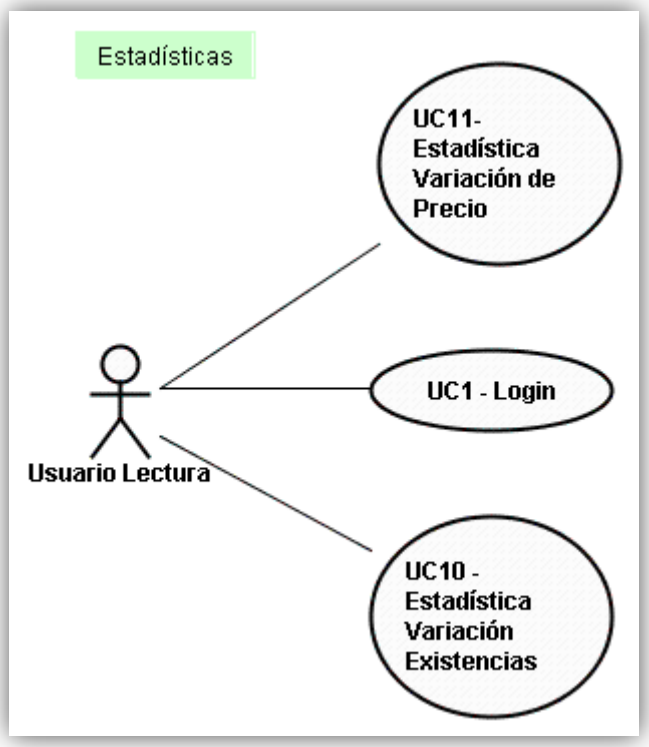


Figura 9: Diagrama de casos de uso. Estadísticas

2.2.6. Listados

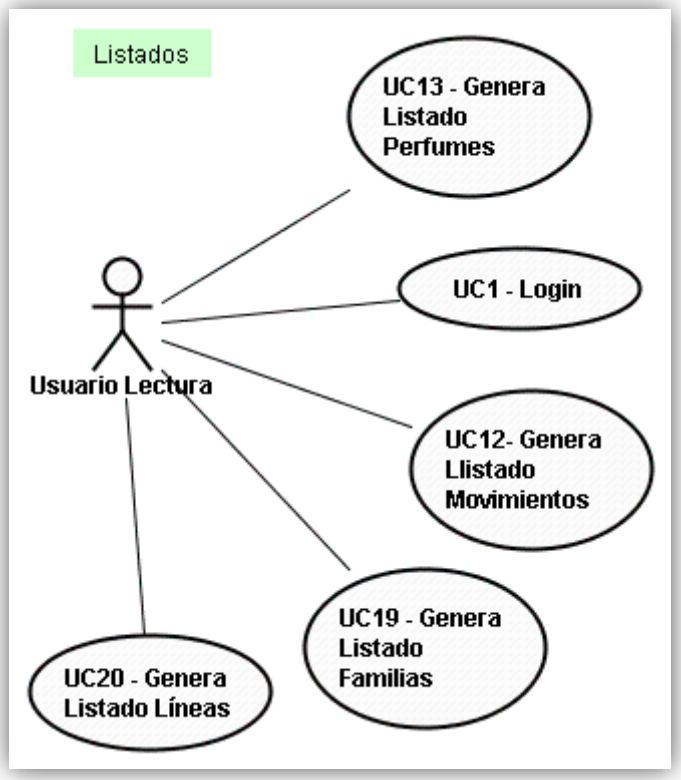


Figura 10: Diagrama de casos de uso. Listados

2.2.7. Servicio Web

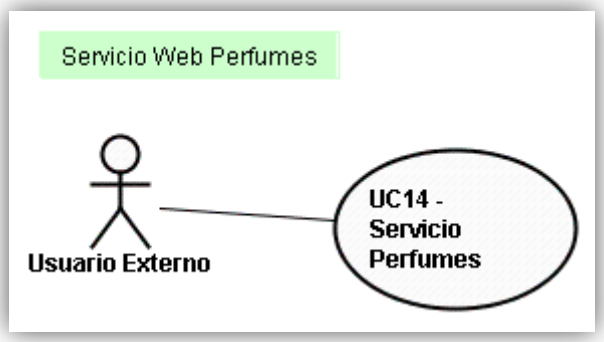


Figura 11: Diagrama de casos de uso. Servicio Web

2.2.8. Usuarios

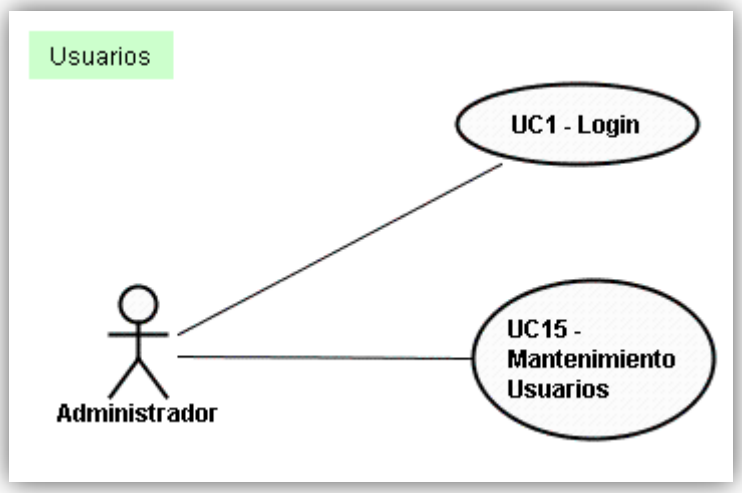


Figura 12: Diagrama de casos de uso. Usuarios

2.3. Descripción de los casos de uso

2.3.1. UC1 – Login

Resumen de la funcionalidad: Identificar a los usuarios y asignarles el rol adecuado dentro del sistema
Actores: Los usuarios del sistema
Pre condición: El usuario no está identificado
Pos condición: Los usuarios están correctamente identificados y tienen el rol asignado
Escenario principal:

1. El usuario se conecta a el sistema
2. El usuario introduce el su código y su clave de acceso
3. El sistema carga los datos del usuario y según su nivel de acceso le asigna el rol de usuario de lectura, usuario normal o bien de administrador

Alternativas del proceso:

3a. La palabra clave o el código de usuario no es igual al que tenemos almacenado

1. Mostramos un mensaje de error
2. Volvemos al punto 2 del escenario principal

2.3.2. UC2 - Visor de Perfumes

Resumen de la funcionalidad: Mostrar al usuario información sobre los perfumes
Actores: Usuario Lectura
Pre condición: Usuario identificado
Pos condición: La información de los perfumes ha sido mostrada al usuario
Escenario principal:

1. Mostramos la pantalla del mantenimiento de perfumes
2. El usuario entra el código del perfume (punto de extensión)
3. El sistema carga los datos del perfume y los enseña al usuario (punto de extensión)

(El usuario puede repetir los puntos 1-3)

4. El usuario sale del mantenimiento

Alternativas del proceso:

2a. El usuario puede localizar el código mediante un buscador

2b. El usuario localiza el perfume utilizando el navegador de registros

3a. Código de perfume no reconocido

2.3.3. UC3 - Mantenimiento de Perfumes

Resumen de la funcionalidad: Mostrar al usuario información sobre los perfumes y permitir hacer cambios
Actores: Usuario
Casos de uso utilizados: UC4 – Gestiona Históricos
Extensión de: UC2 – Visor de Perfumes (**puntos 2 i 3**)
Pre condición: Usuario identificado como usuario o administrador
Pos condición: La información de los perfumes ha sido mostrada al usuario y este ha podido hacer los cambios
Escenario principal:

1. Mostramos la pantalla del mantenimiento de perfumes
2. El usuario entra el código del perfume (punto de extensión)
3. El sistema carga los datos del perfume y los enseña al usuario (punto de extensión)

(El usuario puede repetir los puntos 1-3)

4. El usuario cambia los datos
5. El usuario ordena guardar los datos y la aplicación los guarda en la DB

(El usuario puede repetir los puntos 1-5)

6. El usuario sale del mantenimiento

Alternativas del proceso:

2a. El usuario puede localizar el código mediante un buscador

2b. El usuario localiza el perfume utilizando el navegador de registros

3a. Código de perfume no reconocido. El programa asume que damos una alta

4a. El usuario da orden de eliminar el perfume. Eliminamos y volvemos al punto 1

1 Hay datos que dependen del perfume. Mostramos error y volvemos al punto 1

5a. Algún dato está mal. Mostramos mensaje de error

5b. El precio del producto ha cambiado, lanzamos UC4 - Gestiona Históricos

2.3.4. UC4 – Gestiona Históricos

Resumen de la funcionalidad: Guarda un histórico con los cambios de precio
Pre condición: Hay un cambio de precio
Pos condición: Se ha guardado un registro en el histórico con el precio antes del cambio
Escenario principal:

1. Carga los datos del perfume
2. Inserta en la base de datos los datos del precio
3. Acaba el proceso

Alternativas del proceso:

2a. Ya hay guardada en la DB un cambio con la misma fecha, lo actualizamos

2.3.5. UC5 - Visor de Líneas

Resumen de la funcionalidad: Mostrar al usuario información sobre las líneas
Actores: Usuario Lectura
Pre condición: Usuario identificado
Pos condición: La información de las líneas ha sido mostrada al usuario
Escenario principal:

1. Mostramos la pantalla del mantenimiento de líneas
2. El usuario entra el código de la línea (punto de extensión)
3. El sistema carga los datos y los muestra al usuario (punto de extensión)

   (El usuario puede repetir los puntos 1-3)
4. El usuario sale del mantenimiento

Alternativas del proceso:

2a. El usuario puede localizar el código mediante un buscador

2b. El usuario localiza la línea utilizando el navegador de registros

3a. Código no reconocido.

2.3.6. UC6 – Mantenimiento de Líneas

Resumen de la funcionalidad: Mostrar al usuario información sobre las líneas y permitir hacer cambios
Actores: Usuario
Extensión de: UC5 – Visor de Líneas (**puntos 2 i 3**)
Pre condición: Usuario identificado como usuario o administrador
Pos condición: La información de las líneas ha sido mostrada al usuario y este ha podido hacer los cambios
Escenario principal:

1. Mostramos la pantalla del mantenimiento

2. El usuario entra el código de la línea (punto de extensión)
3. El sistema carga los datos y los enseña al usuario (punto de extensión)

(El usuario puede repetir los puntos 1-3)

4. El usuario cambia los datos
5. El usuario ordena guardar los datos y la aplicación los guarda en la DB

(El usuario puede repetir los puntos 1-5)

6. El usuario sale del mantenimiento

Alternativas del proceso:

2a. El usuario puede localizar el código mediante un buscador

2b. El usuario localiza la línea utilizando el navegador de registros

3a. Código no reconocido. El programa asume que damos una alta

4a. El usuario da orden de eliminar la línea. Eliminamos y volvemos al punto 1

1 Hay datos que dependen de la línea. Mostramos error y volvemos al punto 1

5a. Algún dato está mal. Mostramos mensaje de error

2.3.7. UC7 - Visor de Familias

Resumen de la funcionalidad: Mostrar al usuario información sobre las familias
Actores: Usuario Lectura
Pre condición: Usuario identificado
Pos condición: La información de las familias ha sido mostrada al usuario
Escenario principal:

1. Mostramos la pantalla del mantenimiento
2. El usuario entra el código de la familia (punto de extensión)
3. El sistema carga los datos y los muestra al usuario (punto de extensión)

(El usuario puede repetir los puntos 1-3)

4. El usuario sale del mantenimiento

Alternativas del proceso:

2a. El usuario puede localizar el código mediante un buscador

2b. El usuario localiza la familia utilizando el navegador de registros

3a. Código no reconocido.

2.3.8. UC8 – Mantenimiento de Familias

Resumen de la funcionalidad: Mostrar al usuario información sobre las familias y permitir hacer cambios
Actores: Usuario
Extensión de: UC7 – Visor de Líneas (**puntos 2 i 3**)
Pre condición: Usuario identificado como usuario o administrador
Pos condición: La información de las familias ha sido mostrada al usuario y este ha podido hacer los cambios
Escenario principal:

1. Mostramos la pantalla del mantenimiento
2. El usuario entra el código de la familia (punto de extensión)

3. El sistema carga los datos y los enseña al usuario (punto de extensión)

(El usuario puede repetir los puntos 1-3)

4. El usuario cambia los datos
5. El usuario ordena guardar los datos y la aplicación los guarda en la DB

(El usuario puede repetir los puntos 1-5)

6. El usuario sale del mantenimiento

Alternativas del proceso:

2a. El usuario puede localizar el código mediante un buscador

2b. El usuario localiza la familia utilizando el navegador de registros

3a. Código no reconocido. El programa asume que damos una alta

4a. El usuario da orden de eliminar la familia. Eliminamos y volvemos al punto 1

1 Hay datos que dependen de la familia. Mostramos error y volvemos al punto 1

5a. Algún dato está mal. Mostramos mensaje de error

2.3.9. UC9 – Variación de existencias

Resumen de la funcionalidad: Permitir la variación del stock de los perfumes
Actores: Usuario
Casos de uso utilizados: UC16- Gestiona Histórico Stock
Pre condición: Usuario identificado como Usuario o Administrador
Pos condición: Las existencias de los perfumes han sido modificadas
Escenario principal:

1. Mostramos la pantalla del mantenimiento de existencias
2. El usuario asigna un número al movimiento y puede cambiar la fecha, por defecto la del día de hoy
3. El usuario entra los códigos los perfumes y la variación de stock
4. Guardamos los datos en la DB
5. Lanzamos UC16 para que guarde las variaciones del stock

(El usuario puede repetir los pasos 1-5)

6. El usuario sale del mantenimiento

Alternativas del proceso:

3ª. El usuario puede buscar los códigos del perfume usando un buscador

2.3.10. UC10 – Estadística variación de existencias

Resumen de la funcionalidad: Mostrar estadísticas de variación de existencias por producto
Actores: Usuario lectura
Pre condición: Usuario identificado
Pos condición: Gráfica con la evolución de las existencias
Escenario principal:

1. Mostramos la pantalla de la estadística
2. El usuario entra el código del perfume a analizar
3. El usuario delimita las fechas

4. Generamos la estadística.

(Se pueden repetir 2-4)

5. El usuario sale del módulo

Alternativas del proceso:

2a El usuario puede buscar los códigos de perfumes utilizando un buscador

2.3.11. UC11 – Estadística variación de Precio

Resumen de la funcionalidad: Mostrar estadísticas de variación de precios por producto
Actores: Usuario lectura
Pre condición: Usuario identificado
Pos condición: Gráfica con la evolución de los precios
Escenario principal:

1. Mostramos la pantalla de la estadística
2. El usuario entra el código del perfume a analizar
3. El usuario delimita las fechas
4. Generamos la estadística

   (Se pueden repetir 2-4)
5. El usuario sale del módulo

Alternativas del proceso:

2a El usuario puede buscar los códigos de perfumes utilizando un buscador

2.3.12. UC12 – Genera listado de Movimientos

Resumen de la funcionalidad: Mostrar un listado de los movimientos de stock
Actores: Usuario lectura
Pre condición: Usuario identificado
Pos condición: Listado con los movimientos
Escenario principal:

1. Mostramos la pantalla de selección
2. El usuario delimita el listado por fechas
3. Generamos el listado

   (Se pueden repetir 2-3)
4. El usuario sale del módulo

Alternativas del proceso:

3a. El usuario puede exportar el listado como PDF

2.3.13. UC13 - Genera listado de perfumes

Resumen de la funcionalidad: Mostrar un listado perfumes
Actores: Usuario lectura
Pre condición: Usuario identificado

Pos condición: Listado con los movimientos
Escenario principal:

1. Mostramos la pantalla de selección
2. El usuario delimita por familia
3. El usuario delimita por línea
4. Generamos el listado

(Se pueden repetir 2-4)

5. El usuario sale del módulo

Alternativas de proceso:

2a El usuario puede buscar los códigos de familia con un buscador

3a El usuario puede buscar los códigos de línea con un buscador

5a El usuario puede exportar el listado como PDF

2.3.14. UC14 - Servicio Perfumes

Resumen de la funcionalidad: Permitir acceder remotamente a los datos no confidenciales de los perfumes
Actores: Usuario externo
Pre condición: Ninguna
Pos condición: Los datos de los perfumes han sido recibidos por el usuario externo
Escenario principal:

1. El usuario se conecta con nuestro sistema
2. El usuario solicita la lista de perfumes
3. El sistema recopila y devuelve la información pedida
4. El usuario se desconecta

2.3.15. UC15 – Mantenimiento Usuarios

Resumen de la funcionalidad: Mostrar los datos de los usuarios y permite hacer cambios
Actores: Administrador
Pre condición: Usuario identificado como administrador
Pos condición: La información de los usuarios se ha mostrado y el administrador la ha podido cambiar
Escenario principal:

1. Mostramos la pantalla del mantenimiento
2. El usuario entra el código a editar
3. El sistema carga los datos y los muestra

   (Se pueden repetir los pasos 2-3)
4. El usuario realiza cambios en los datos
5. Se guardan los datos en la DB
6. (Se pueden repetir los pasos 2-6)
7. El usuario sale del mantenimiento

Alternativas de proceso:

2a. El usuario puede usar un buscador para encontrar el código

2b. El usuario puede usar el navegador para encontrar el código

3a. El código no existe, el sistema asume una alta

4a. Se ordena eliminar al usuario. Se borra y se vuelve al paso 1

5a. Algún dato está mal, mostramos error

2.3.16. UC16 – Gestiona Histórico Stock

Resumen de la funcionalidad: Guarda un histórico con los cambios de stock
Pre condición: Hay un cambio de stock
Pos condición: Se ha guardado un registro en el histórico con el stock antes del cambio
Escenario principal:

1. Carga los datos del perfume
2. Inserta en la base de datos los datos del stock
3. Acaba el proceso

Alternativas del proceso:

2a. Ya hay guardada en la DB un cambio con la misma fecha, lo actualizamos

2.4. Diagrama de clases del modelo conceptual

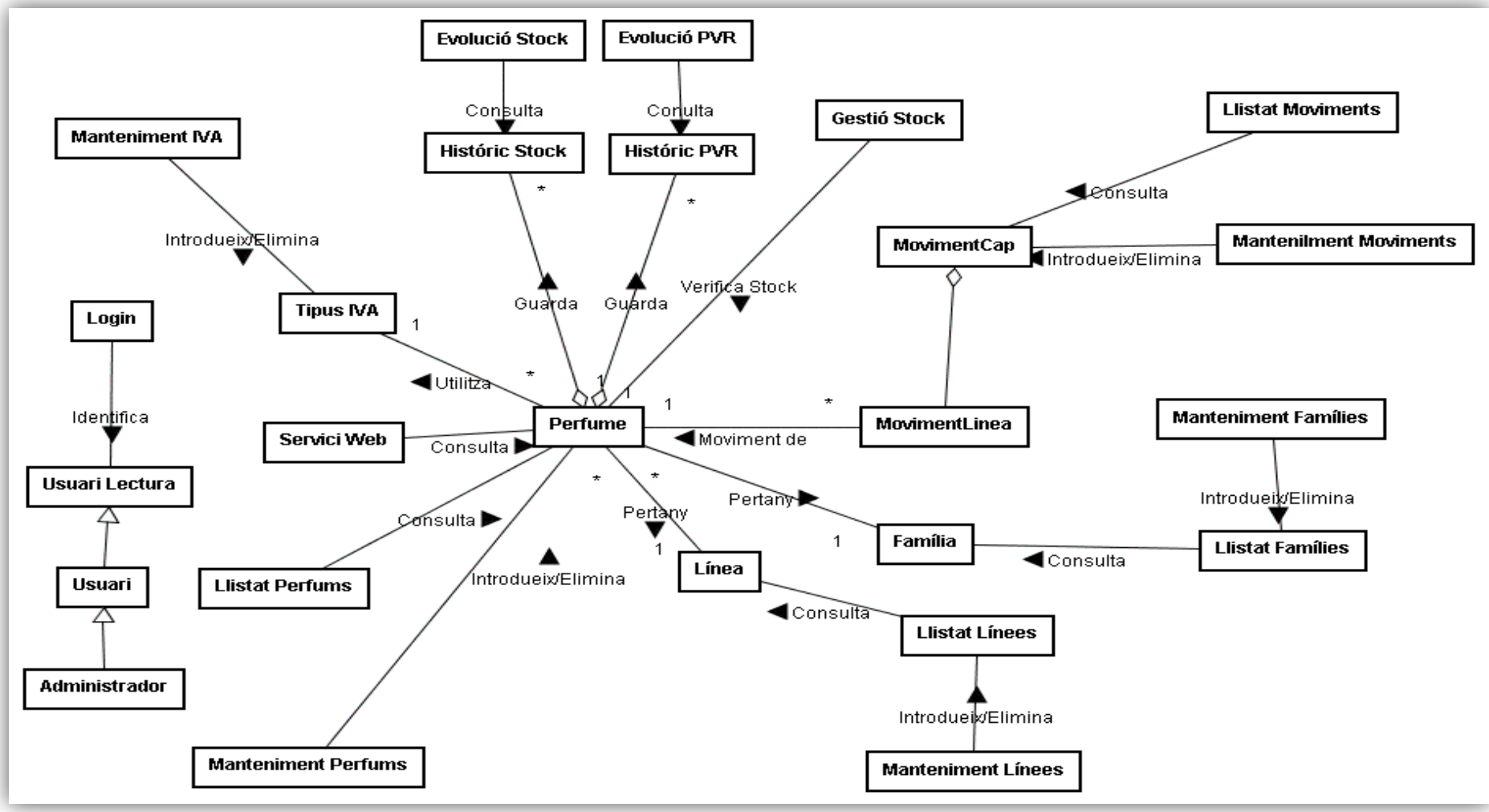


Figura 13: Diagrama de clases del modelo conceptual

3. Diseño

3.1. Arquitectura del software

3.1.1. Introducción

La arquitectura de la aplicación es similar al patrón MVC, (Erich Gamma, 1995) pero con varias diferencias remarcables. Está compuesta por los mismos componentes, modelo, controlador y vista pero varía el papel de la vista y del controlador

La vista se encarga de gestionar la interface con el usuario, de recibir los eventos generados por este y si es necesario reenviarlos al controlador. Su responsabilidad en este caso es mayor que en el patrón MVC ya que como se ha dicho se encarga directamente de gestionar las peticiones del usuario y de responder a ellas. El objetivo principal de la vista es que el usuario tenga un fácil acceso a la información y que dicho acceso le resulte agradable. La vista está totalmente optimizada y vinculada a la plataforma en la que funciona, páginas Web en este caso, y por tanto puede utilizar todos los recursos que esta dispone. Su responsabilidad se limita a como se mostrará la información pero no como se va a procesar. Cuando un usuario hace una petición que precisa de algún tipo de procesamiento, es decir una petición que precisa de una respuesta no sólo visual, la vista redirige está petición al controlador y finalmente muestra si es necesaria la respuesta de este al usuario. La comunicación de la vista con el controlador se realiza mediante los métodos invocados del controlador por la vista y por los eventos generados por el controlador que puede recoger la vista. La vista puede acceder al modelo, pero siempre bajo la supervisión del controlador lo que nos permite garantizar la validez del modelo. Para evitar atar a la vista con un controlador determinado se ha optado por trabajar mediante interfaces que apuntan a objetos creados mediante Spring.NET. Esta decisión desacopla totalmente la vista de cualquier implementación de los controladores con lo que si fuera necesario se podrían volver a implementar sin necesidad siquiera de recompilar los módulos que contienen la vista.

El controlador es el encargado de manipular el modelo y, si es necesario, de leerlo y guardarlo en la base de datos. Su diseño ignora a la interface de usuario a diferencia del MVC, no sabe realmente si hay una y por tanto no precisa de ella para su correcto funcionamiento. El controlador gestiona como se deben leer los datos de la base de datos, como se deben grabar (siempre a través de los depósitos de objetos), de verificar que el estado del modelo sea correcto para su persistencia y de iniciar y terminar las transacciones. Todo objeto no agregado tiene su propio controlador que se encarga de su correcta manipulación. La principal ventaja de este diseño es la facilitad para reutilizar a los controladores en todo tipo de vistas, así por ejemplo el controlador de perfumes se utiliza en la vista del mantenimiento de perfumes, en los listados y también en el servicio web de información de perfumes. Todos los controladores se han diseñado con una misma interface base para facilitar su uso, este hecho facilita muchísimo la creación de las vistas ya que todos ellos comparten un gran número de métodos y eventos así como la filosofía de funcionamiento.

El modelo está diseñado siguiendo el patrón Domain Model (Fowler, 2003) y a partir de dicho diseño se ha creado la base de datos. Los objetos que componen el dominio se han creado con la filosofía POCO (Plain Old CLR Objetct), es decir que son objetos sin ningún tipo de atadura a algún tipo de infraestructura ajena al proyecto e ignoran el concepto de persistencia. Esta filosofía nos da una gran libertad a la hora de diseñar el modelo ya que no hay que preocuparse de ningún requisito previo.

Otro punto que se ha tenido en cuenta a la hora de diseñar la aplicación es intentar mantener en lo posible la independencia de la base de datos y a ser posible también de la tecnología utilizada para acceder a ella. La solución adoptada ha sido la de crear una capa de software que actúa como intermediario entre la solución utilizada para la persistencia y el resto de la aplicación (Nilsson, 2006). Esta capa de software define una serie de interfaces (IDeposit<T>, IDBManager y IContextDB) que deben ser implementados para la solución de persistencia que se vaya a utilizar en el proyecto. Si en el futuro fuera necesario de cambiar de tecnología de acceso a la base de datos bastará con definir estos interfaces para la nueva tecnología que se vaya a utilizar sin necesidad de modificar nada más del resto

del proyecto, en realidad ni siquiera importará recompilar el resto de módulos. En este proyecto los depósitos de objetos son usados exclusivamente por los controladores.

Finalmente señalar que la aplicación se ha diseñado por capas según su funcionalidad. Los objetos de una capa determinada no pueden utilizar directamente objetos de una capa superior, pero si de una capa inferior aunque esto también se ha minimizado mucho en este proyecto. La distribución por capas y los objetos que pertenecen a cada una de ellas se pueden encontrar en los puntos siguientes de esta memoria.

Para poder satisfacer estos requerimientos técnicos se ha optado por utilizar diversos módulos y librerías de código abierto cuyo papel en el proyecto también se comentarán en los puntos siguientes de esta memoria.

3.1.2. Diseño por capas

La aplicación se ha diseñado por capas según su funcionalidad y de tal forma que lo objetos que componen una capa determinada pueden utilizar objetos de las capas inferiores (aunque se ha minimizado también este hecho al máximo), pero no de las superiores evitándose así interdependencias no deseadas. En la Figura 14 podemos ver un gráfico mostrando las capas en las que se divide el proyecto y los objetos que las componen.

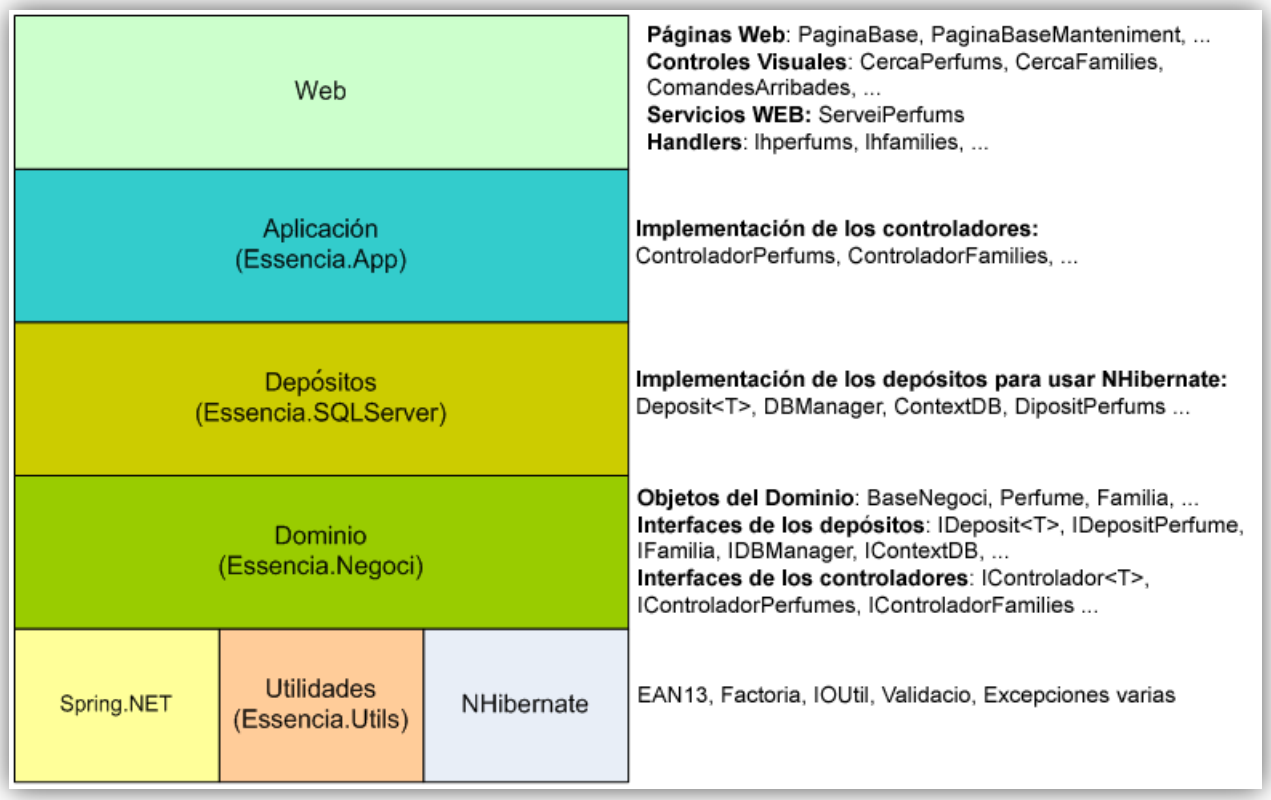


Figura 14: Diseño por capas

Tal como muestra la Figura 14 la capa Web es la única capa que se encarga del interface del usuario, contiene las páginas web, los controles visuales, servicios web y handlers. Esta capa no contiene ningún tipo de lógica de negocio y precisa de los controladores para ello. En el diseño actual la capa de la Web realmente sólo depende del dominio (Essencia.Negoci) y de la capa de utilidades (Essencia.Utils) ya que la utilización de interfaces junto con Spring.NET ha permitido separarla totalmente del resto de módulos.

La capa de la aplicación contiene la implementación de los controladores. Para acceder a la base de datos utiliza los depósitos a través de interfaces. En este caso también se utiliza Spring.NET para separar esta capa de cualquier implementación concreta de los depósitos con lo que si fuera necesario cambiar de tecnología para acceder a la base de datos también se podría hacer sin necesidad siquiera de recompilar este módulo. Como en el caso de la capa Web, realmente sólo depende de Essencia.Negoci y de Essencia.Utils.

La capa de depósitos (Essencia.SQLServer) contiene las clases necesarias para acceder a la base de datos utilizando en este caso a NHibernate. El uso de NHibernate ha simplificado mucho esta capa ya que esta librería incluye todo lo necesario para acceder fácilmente a la base de datos. También depende solamente de Essencia.Negoci y de Essencia.Utils.

La capa del dominio (Essencia.Negoci) define todos los objetos pertenecientes al dominio y los interfaces necesarios para manipularlos. Sólo tiene dependencias con Essencia.Utils.

Finalmente la capa de utilidades (Essencia.Utils) contiene clases y utilidades diseñadas para facilitar la programación del resto de módulos. No tiene dependencias dentro del proyecto.

3.1.3. NHibernate

NHibernate, una librería de código abierto, es una herramienta Objeto/Relacional (O/R) que nos permite mapear nuestros objetos .NET como tablas en una base de datos, más conocido por sus siglas en ingles ORM (object/relational mapping). NHibernate no sólo se hace cargo del mapeado de nuestros objetos sino que también incluye un potente lenguaje de consulta y carga de datos.

NHibernate es la versión para Microsoft .NET de la famosa librería Hibernate existente en el mundo Java. En la Figura 15 podemos ver a grandes rasgos su arquitectura.

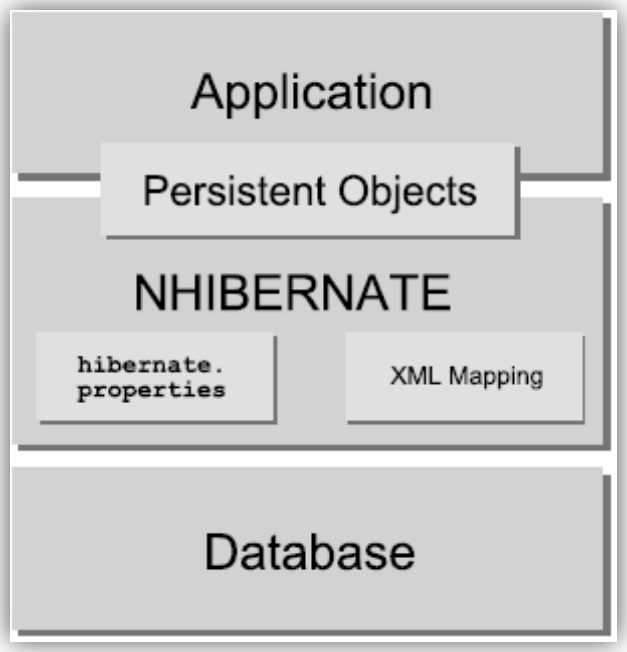


Figura 15: Arquitectura de NHibernate

El proyecto presentado utiliza NHibernate para la persistencia de los objetos en la base de dados. El diseño de NHibernate hace que nuestra aplicación sea realmente independiente de la base de datos, siempre y cuando este soportada por la librería. La aplicación funciona sobre SQL Server 2005 pero basta cambiar el fichero de configuración de NHibernate (hibernate.cfg.xml) para que funcione sin

más cambios sobre cualquiera de las otras bases de datos soportadas. El la Figura 16 podemos ver el fichero de configuración utilizado actualmente.

```
<?xml version='1.0' encoding='utf-8'?>
<hibernate-configuration xmlns="urn:nhibernate-configuration-2.2">
  <!-- an ISessionFactory instance -->
  <session-factory>
    <!-- properties -->
    <property name="connection.provider">NHibernate.Connection.DriverConnectionProvider</property>
    <property name="connection.driver_class">NHibernate.Driver.SqlClientDriver</property>
    <property name="show_sql">false</property>
    <property name="dialect">NHibernate.Dialect.MsSql2005Dialect</property>
    <property name="max_fetch_depth">1</property>
  </session-factory>
</hibernate-configuration>
```

Figura 16: Fichero de configuración de NHibernate

NHibernate permite diseñar los objetos que vamos a guardar en la base datos con muy pocos prerrequisitos, por no decir ninguno. Esto nos da una gran libertad a la hora de crear nuestro dominio ya que en principio no tenemos que condicionar el diseño de los objetos a como se guardarán en la base de datos sino exclusivamente en función de que queremos hacer con ellos. Esta libertad que nos da NHibernate puede tener algunos efectos no deseados en el rendimiento de la aplicación, este problema es fácilmente comprensible con un ejemplo.

Imaginemos que tenemos las clases de la Figura 17. En un momento dado la aplicación quiere mostrar por pantalla el nombre de una instancia de la clase Padre, para ello la aplicación ordena a NHibernate que cargue el objeto de la base de datos y devuelva la referencia a dicho objeto. NHibernate sabe que el objeto Padre tiene una propiedad llamada Nombre y una lista de objetos del tipo Hija, por tanto para devolver correctamente el resultado deberá cargar el registro correspondiente a Padre y todos los registros Hija asociados aunque en este caso la aplicación no los vaya a utilizar para nada.

```
public class Padre
{
    private string nombre;
    public string Nombre
    {
        get
        {
            return nombre;
        }
        set
        {
            nombre = value;
        }
    }

    private IList<Hija> hijas = new List<Hija>();
    public IList<Hija> Hijas
    {
        get
        {
            return hijas;
        }
    }
}

public class Hija
{
    private Padre padre;
    public Padre Padre
    {
        get
        {
            return padre;
        }
        set
        {
            padre = value;
        }
    }
...
...
}
```

Figura 17: Ejemplo de clases para explicar la problemática del mapeado de clases

Este comportamiento de NHibernate puede hacer que el rendimiento de la aplicación sufra considerablemente sobre todo en aquellos casos de dominios de objetos fuertemente asociados entre sí pudiendo llegarse en el caso más extremo a cargarse toda la base de datos en memoria. NHibernate nos da dos alternativas para solucionar este problema:

La primera solución es que los objetos que vamos a guardar en la base de datos sean la implementación de uno o más interfaces. En este caso cuando NHibernate no devuelve la referencia al objeto/interface pedido en una consulta sino que realmente nos devolverá, si es necesario, su propia implementación de dicho interface que no es más que un proxy a nuestro objeto original. De esta forma cuando se accede a una propiedad del objeto que hace referencia a otro objeto almacenado en la base de datos NHibernate puede comprobar si esta ya cargado en memoria y si no lo está cargarlo antes de que la aplicación lo use.

La otra solución que nos da NHibernate es que definamos todos los métodos y propiedades no privadas de nuestros objetos persistentes como virtuales, de esta forma cuando pedimos a NHibernate un objeto de la base de datos, la librería, siempre que sea necesario, creará en tiempo de ejecución un objeto derivado del nuestro con todos los métodos y propiedades sobrescritos de tal forma que cuando se accede a una propiedad que hace referencia a otro objeto persistente, si este no está cargado en memoria, NHibernate lo cargará antes de que pueda ser utilizado. En la Figura 18 podemos ver un esquema del funcionamiento. Todo el trabajo de creación de proxys NHibernate lo hace utilizando la librería de código abierto Castle.DynamicProxy.

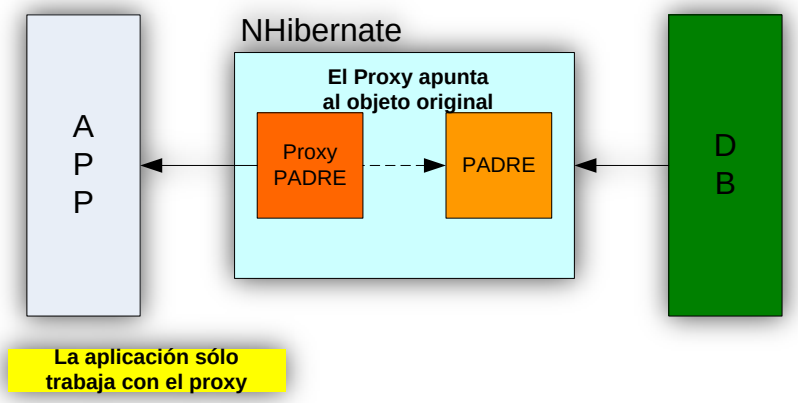


Figura 18: Esquema de funcionamiento de los proxys en NHibernate

El mapeado de los objetos en NHibernate se define a través de un fichero de configuración en formato XML tal como podemos ver en la Figura 19. Se han creado tantos ficheros de mapeado como clases persistentes. Dichos ficheros se ha incluido como recursos en Essencia.SQLServer y por tanto están incluidos dentro de dicho Assembly. El formato y opciones de estos ficheros se escapan del alcance de esta memoria pero están totalmente documentados en el manual de NHibernate.

```
<?xml version="1.0" encoding="utf-8" ?>
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2" assembly="Essencia.Negoci"
      namespace="Essencia.Negoci.Objectes" default-access="field.camelcase">
  <class name="Familia" table="Familia" lazy="false">
    <id name="OID" column="OID" type="Int32">
      <generator class="hilo"/>
    </id>
    <version name="Versio" type="Int32"/>
    <property name="Codi" not-null="true"/>
    <property name="Nom" not-null="true" length="50"/>
    <set name="Perfums" inverse="true" lazy="false" cascade="none">
      <key column="Familia"/>
      <one-to-many class="Perfum"/>
    </set>
  </class>
</hibernate-mapping>
```

Figura 19: Ejemplo de fichero de mapeado

El proyecto aquí presentado define todos los métodos y propiedades de los objetos persistentes como virtuales para aprovechar la capacidad de NHibernate de cargar los objetos según la demanda que se hace de ellos. Otra concesión que se ha hecho a NHibernate en el diseño de la aplicación es la de incluir una propiedad llamada “Versio” en todos los objetos persistentes. Esta propiedad la utiliza NHibernate para controlar la concurrencia, no es imprescindible, pero si muy útil. Su funcionamiento es sencillo, cada vez que NHibernate guarda cualquier modificación de un objeto incrementa el valor de “Versio” de ese objeto en 1 y antes de hacer esto verifica que el valor de esta propiedad sea igual en el objeto en memoria y en su representación en la base de datos. Si los valores son diferentes significa que alguien ha modificado el registro desde que nosotros lo cargamos y por tanto podría producirse una pérdida de información.

3.1.4. Spring.Net

La inversión del control (Inversion of Control, IOC en ingles) (Fowler, 2004) explica la manera con la cual un objeto obtiene las referencias a sus dependencias. La aproximación habitual es obtener estas dependencias desde dentro del propio objeto. Este método tiene la desventaja de que el objeto tiene una dependencia explicita sobre el método de creación o localización de esas dependencias y cualquiera que llame a este objeto no sólo dependerá del propio objeto, sino también de todas sus dependencias internas. Con la inversión del control las dependencias de un objeto se pasan o bien a través del constructor o bien a través propiedades con lo que el objeto no tiene ninguna dependencia sobre cómo se localizan o crean dichas dependencias. El resultado final es la creación de objetos poco acoplados entre sí y por tanto más fácilmente reutilizables especialmente si trabajamos mediante interfaces.

Spring.Net es un contenedor que facilita la IOC permitiendo configurar y relacionar los objetos fácilmente desde ficheros de configuración. En la Figura 20 podemos ver la configuración de DBManager con propiedades existentes en la configuración de la aplicación y del controlador de usuarios que usa el DBManager.

```
<!-- DB -->
<object id="IDBManager" type="Essencia.SQLServer.DBManager, Essencia.SQLServer" singleton="false">
  <constructor-arg name="cadenaConexio" value="${cadenaConexio}" />
  <constructor-arg name="shema" value="${shemaConexio}" />
</object>

<!-- Controladors -->
<object id="IControladorUsuaris" type="Essencia.App.Controladors.ControladorUsuaris, Essencia.App" singleton="false">
  <constructor-arg name="dBManager" ref="IDBManager" />
</object>
```

Figura 20: Ejemplo de la configuración de Spring.Net

Cuando la aplicación necesita utilizar un controlador basta que lo solicite a Spring.Net tal como muestra Figura 21 Spring.Net se encarga de crear y configurar el objeto DBManager que necesita para su funcionamiento el controlador de usuarios, asociarlo con el controlador y finalmente devolver la instancia del objeto a la aplicación. El objeto que ha solicitado el controlador no sabe ni tiene relación con la implementación que le ha sido proporcionada mediante Spring.Net, el controlador creado esta inicializado con una referencia a un objeto que implementa el interface IDBManager pero no sabe realmente cual es. Todo esto nos permite crear objetos muy reutilizables y que no estan asociados entre sí.

```
IControlador<Usuari> ctrl=(IControlador<Usuari>)Essencia.Utils.Factoria.Context["IControladorUsuaris"];
```

Figura 21: Obteniendo una referencia a un objeto en Spring.NET

El proyecto presentado hace uso de las facilidades de Spring.Net para crear y configurar los objetos que permiten el acceso a la base de datos y a los controladores. Esto permite que las diferentes capas de objetos de la aplicación puedan utilizarse entre sí sin mantener ningún tipo de referencia entre ellos, salvo los interfaces. En la Figura 22 las fechas sólidas muestran la dirección de las dependencias entre módulos a **nivel de compilación y uso** y las fechas discontinuas muestran las “dependencias” en tiempo de **ejecución.** Concretamente las vistas obtienen utilizando Spring.Net las referencias a sus controladores los cuales han sido configurados también mediante Spring.NET para que usen el sistema de persistencia que estemos utilizando, todo ello sin referencias explícitas. La falta de referencias permite que se puedan modificar o incluso cambiar esos módulos sin necesidad de recompilar el resto de la aplicación. Además nos permite crear y integrar muy facilmente nuevas familias de objetos para acceder a la base de datos, para controlar a la aplicación o bien ampliar los ya existentes.

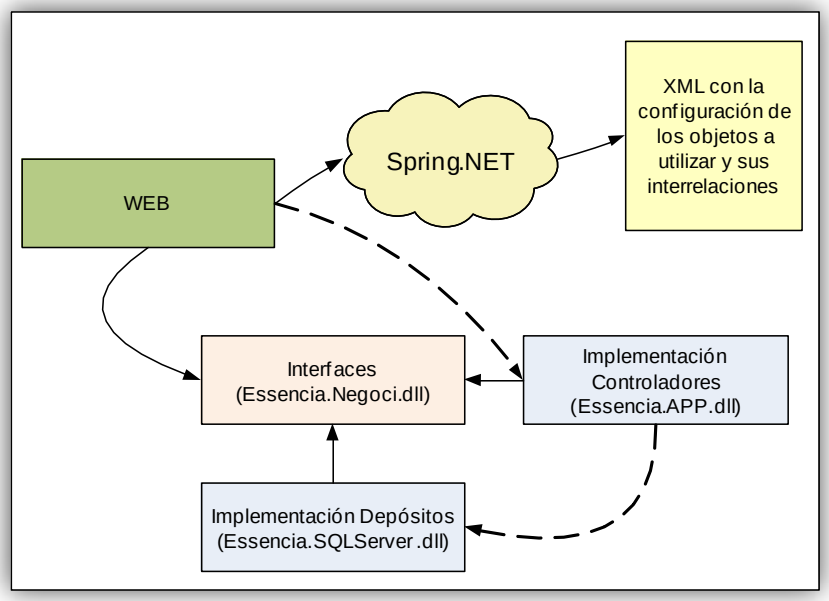


Figura 22: Mínimas interdependencias gracias a Spring.NET

3.1.5. Otras librerías utilizadas

Gios.PDF es la librería utilizada para la generación de los ficheros PDF. Es una librería muy sencilla cuya principal virtud es la facilitad que tiene para crear documentos PDF compuestos por tablas cosa que ha facilitado mucho la creación de los listados. Al contrario de las demás librerías utilizadas en el proyecto, se ha incluido el código fuente de la misma debido a que se ha tenido que modificar para que trabajara correctamente con los caracteres Unicode.

ZedGraph es una librería para generar gráficas en 2D. Se utiliza en el proyecto para mostrar las estadísticas. Dispone de un control Web que facilita la integración en el proyecto.

AJAX Control Toolkit es una librería diseñada para aumentar las posibilidades que brinda ASP.NET AJAX.

3.2. Diagrama de la arquitectura del hardware

La arquitectura de hardware necesaria para el correcto funcionamiento de este proyecto es la generalmente utilizada en cualquier proyecto ASP.NET con acceso a una base de datos.

En Figura 23 podemos ver una instalación típica de una web con acceso tanto exterior como interior.

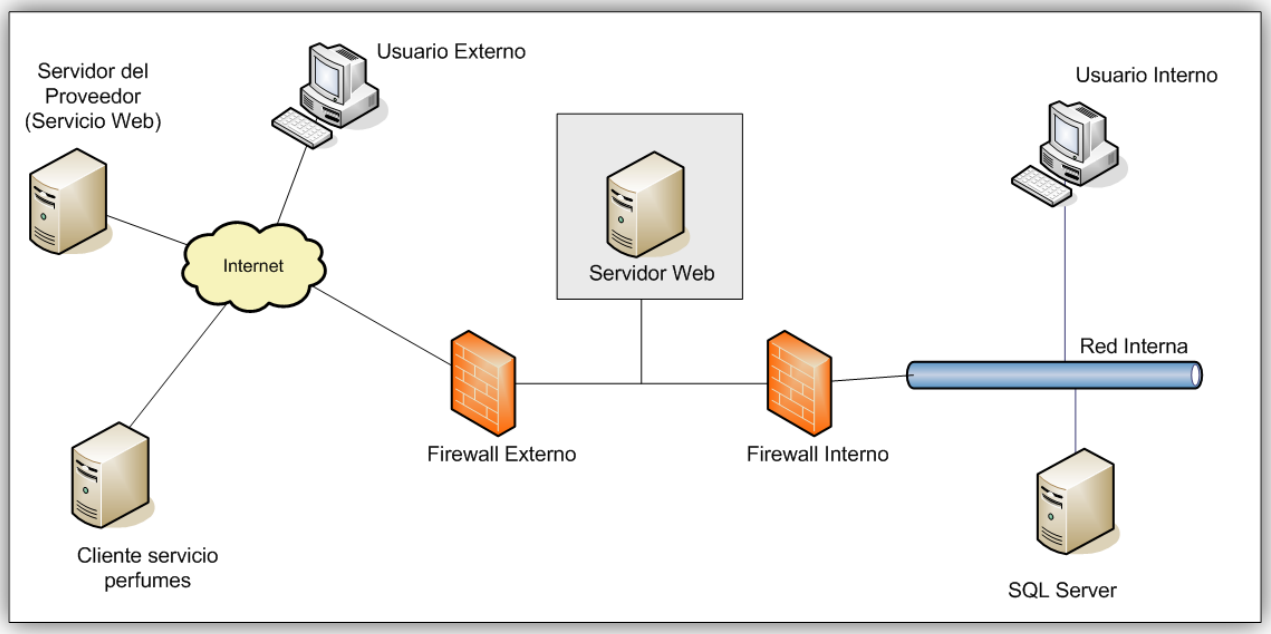


Figura 23: Diagrama de la arquitectura del hardware

3.3. Diagrama de clases

A continuación se presentan una serie de diagramas UML de las clases utilizadas en este proyecto. Se han separado los diagramas en varios para mejorar su comprensión.

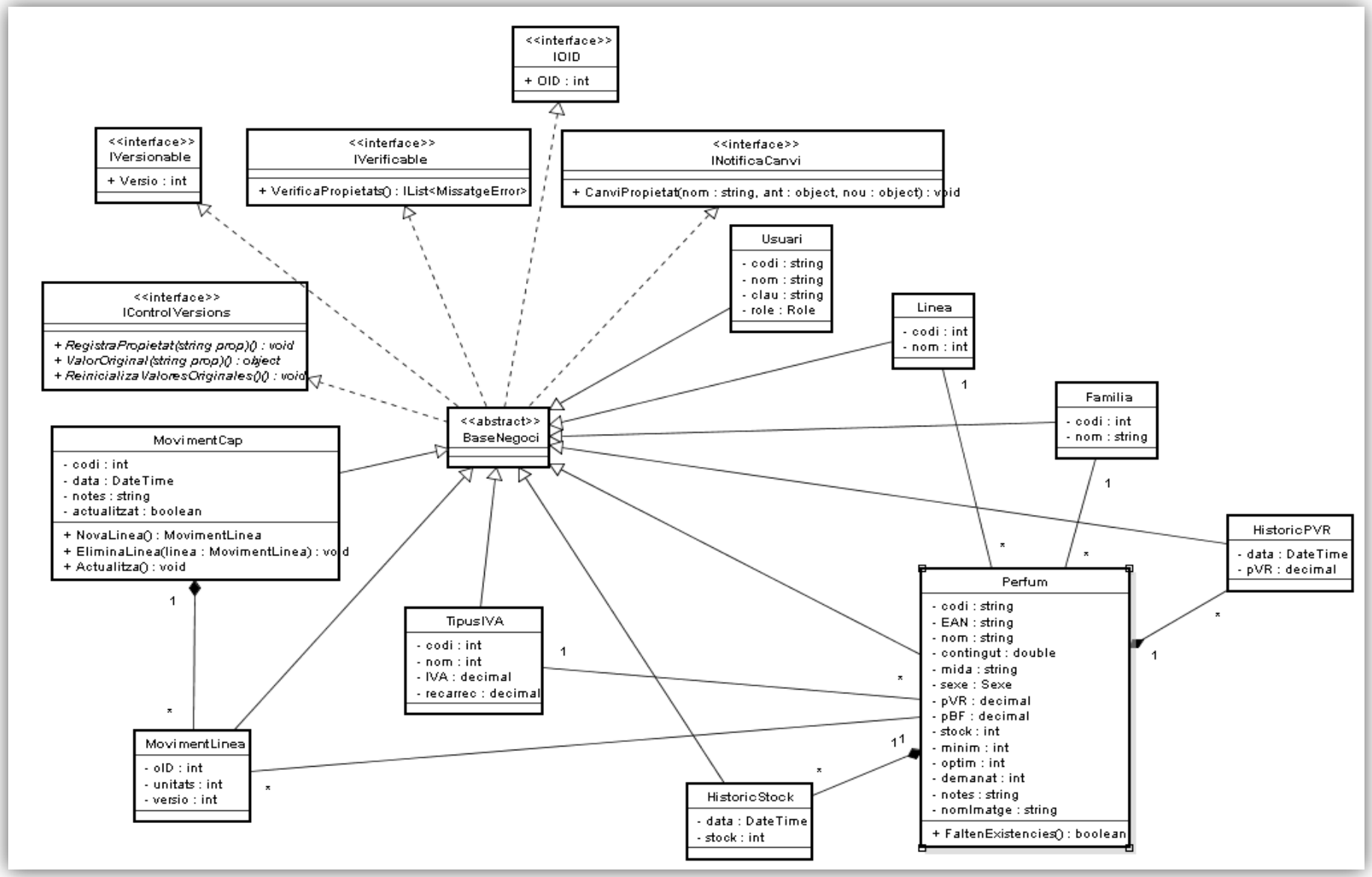


Figura 24: Diagrama de clases del dominio

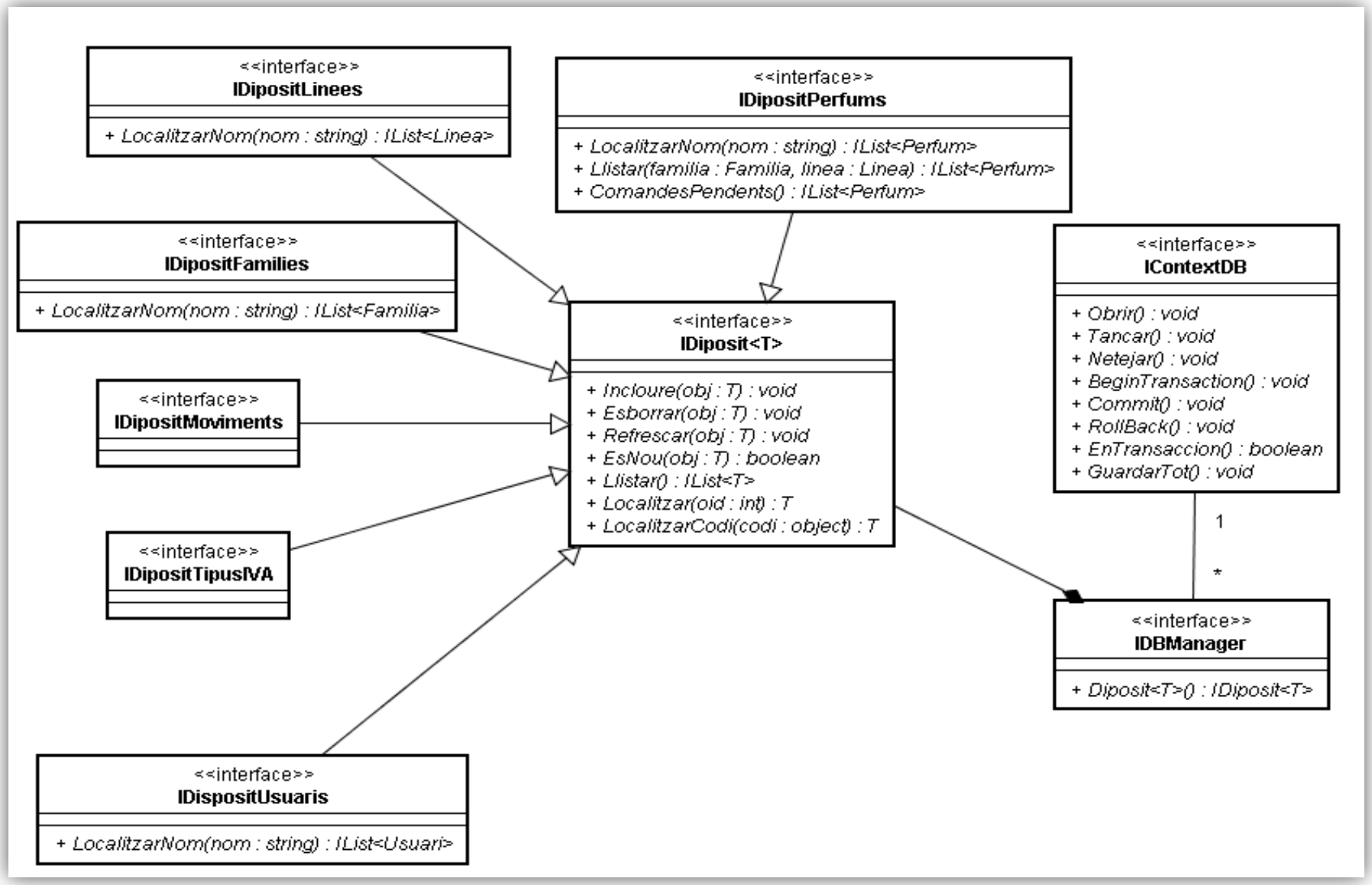


Figura 25: Diagrama de Depósitos

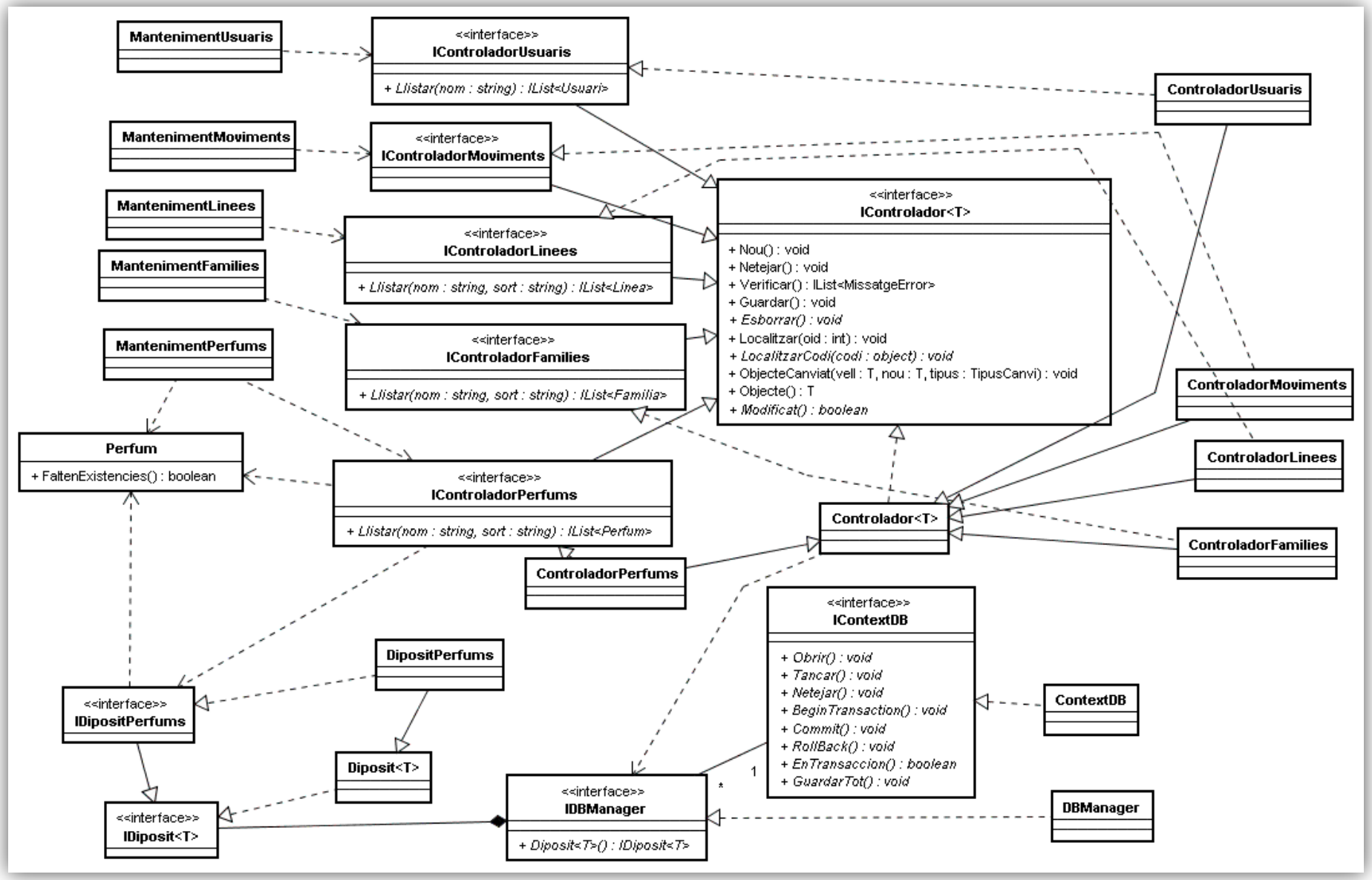


Figura 26: Diagrama de Controladores en su contexto

3.3.1. Interfaces

A continuación se pueden ver las interfaces más importantes desarrolladas para este proyecto junto con una descripción de sus métodos más relevantes:

INotificaCanvi	
Interface que indica que el objeto que lo implementa avisa de los cambios que se producen en sus propiedades	
Modificat	Indica si el objeto se ha modificado o no.
CanviPropietat	Evento lanzado cuando se produce un cambio de valor en alguna de las propiedades del objeto.

IVerificable	
Interface que indica que el objeto que la implementa puede verificarse	
VerificaPropietats():IList<MissatgeError>	Devuelve una lista con todos los errores o problemas del objeto

IDBManager	
Define una factoría de *IDiposits*. Es el punto de central del mecanismo de persistencia utilizado en el proyecto y parte de la capa de software que aísla a la aplicación del método de persistencia elegido.	
Context	Devuelve el contexto en el que trabaja el *IDBManager* y que utilizarán todos los *IDeposit* que se soliciten a través de él.
Diposit<T>(): T	Devuelve el *IDiposit* capaz de gestionar los objetos del tipo T. SI no existe ningún depósito definido para gestionarlos devuelve *NULL*.

IContextDB	
Interface que especifica los métodos a implementar para controlar el mecanismo de persistencia utilizado en el proyecto. Conceptualmente es el equivalente a una conexión a una base de datos. Es parte de la capa de software que aísla a la aplicación del método de persistencia elegido.	
Obrir()	Abre una conexión con la base de datos o con el servicio utilizado para permitir la persistencia de nuestros objetos.
Tancar()	Cierra la conexión previamente abierta.
Netejar()	Elimina todas las actualizaciones pendientes y vuelve al estado inicial.
BeginTransaction()	Inicia una transacción.
Commit()	Escribe todos los cambios en el medio utilizado para la persistencia y finaliza la transacción.
RollBack()	Aborta la transacción.
EnTransaccio(): bool	Indica si actualmente nos encontramos en medio de una transacción o no.
GuardarTot()	Guarda todos los cambios pendientes. Si no estamos en medio de una transacción inicia una y la valida.

IDiposit<T>
Interface utilizado como base para describir a los depósitos de objetos del tipo T. La aplicación

utiliza el concepto de depósito como almacén de objetos creando una abstracción del tipo de mecanismo de almacenamiento.	
Incloure(T obj)	Incluye un objeto en el depósito. El objeto puede ser nuevo o bien ya existir previamente en el medio persistente utilizado.
Esborrar	Elimina un objeto del depósito. La eliminación no se hará efectiva hasta que el contexto guarde los cambios.
Refrescar(T obj)	Relee el objeto del medio persistente.
EsNou(T obj): bool	Indica si el objeto es nuevo o bien ya existe en el medio persistente.
Context(): IContextDB	Contexto en el que trabaja este depósito.
DBManager():IDBManager	DBManager que ha creado este depósito.
Llistar():IList<T>	Devuelve una lista con todos los objetos almacenados en el depósito.
LlistarOIDs: IList<int>	Devuelve una lista con todos los identificadores únicos de los objetos.
Localitzar(int oid): T	Localiza un objeto por su identificador único.
LocalitzarCodi(object obj): T	Localiza un objeto por su código de negocio.

IControlador<T>	
Interface base de los controladores. Los controladores son interfaces de alto nivel entre la UI y los objetos con los que esta trabaja.	
Nou()	Crea un nuevo objeto del tipo T y a partir de ahora el controlador trabajará con él.
Netejar()	Cancela los cambios y el objeto actual pasa a ser el nulo.
Verificar():IList<MissatgeError>	Verifica el objeto actual. Devuelve una lista con los posibles errores.
Guardar()	Guarda el objeto en el depósito correspondiente.
Esborrar()	Elimina el objeto mediante el depósito. El nuevo objeto del controlador será el nulo.
Localitzar(into oid)	Localiza un objeto mediante el depósito según su identificador único.
LocalitzarCodi(object codi)	Localiza un objeto mediante el depósito según su identificador de negocio.
Objecte	Objeto del tipo T con el que está trabajando el controlador actualmente.
Modificat	Indica si se ha producido alguna modificación.
CanviObjecteHandler	Evento lanzado cuando el objeto del controlador cambia de estado.

INavegable	
Interface que indica que el objeto que lo implementa contiene un navegador de registros	
Navegador: INavegador	Devuelve el navegador de registros que contiene el objeto
INavegador	
Definición de un navegador de registros. Utilizado para desplazarse entre los registros de un	

mismo tipo siguiendo un orden lógico preestablecido	
Primer()	Coloca al navegador sobre el primer registro lógico.
Darrer()	Coloca al navegador sobre el último registro lógico.
Anterior()	Coloca al navegador sobre el registro anterior al actual, si este es el primero no se mueve.
Seguent()	Coloca al navegador sobre el registro siguiente al actual, si este es el último no se mueve.

IOID	
Interface que indica que el objeto que lo implementa define la clave artificial que utiliza el sistema como identificador único	
OID	Devuelve el identificador único del objeto. Tiene el papel de clave primaría artificial en los objetos persistentes.

IVersionable	
Interface que indica que el objeto soporta el concepto de versión para distinguir si ha sido modificado en el medio persistente desde que el fue cargado	
Versio	Devuelve el número de la versión del objeto.

IControlVersions	
Interface que especifica que el objeto puede almacenar el valor original de una propiedad y recuperar dicho valor en cualquier momento a pesar de que haya cambiado	
RegistraPropietat(string prop)	Especifica que queremos controlar los cambios de los valores en una propiedad determinada.
ValorOriginal(string prop): object	Devuelve el valor que tenía originalmente una propiedad a pesar de que puede que no sean igual al que tiene ahora.
ReinicializaValoresOriginales()	Vuelve asignar a las propiedades registradas sus valores originales.

IDBSensible	
Indica que el objeto que lo define necesita acceso directo al medio persistente. La infraestructura del proyecto se encargará de proveerle de dicho acceso.	
DBManager	IDBManager que podrá utilizar el objeto para acceder a la persistencia

IControlNavegacio	
Interface para la notificación del interés del usuario en cambiar de registro actual entre el control visual de navegación y la pantalla de mantenimiento que lo contiene.	
Navegacio	Evento que indica que tipo de movimiento quiere hacer el usuario.

IMenu	
Definición de menús.	
RegistraEntrada(string text, string url, string urlImg);	Introduce en el menú una nueva opción.

Selecciona(string opc)	Ordena al menú que marque una de las opciones determinadas.

3.3.2. Clases

Las clases más importantes implementadas en el proyecto son:

BaseNegoci	
Clase base para todos los objetos persistentes. Implementa los interfaces IVerificable, IOID, INotificaCanvi, IVersionable, IControlVersions	
VerificaPropietats(IList<MissatgeError> errors)	Método protegido utilizado por los descendientes para indicar que errores tienen.
AjustaPropietat<T>(string propietat, ref T actual, T nou)	Método auxiliar que permite cambiar el valor de una propiedad y si realmente cambia lanza el evento correspondiente.
AjustaPropietatNoNull(string propietat, ref string actual, string nou)	Método auxiliar que permite cambiar el valor de una propiedad tipo cadena identificando null con cadena vacía. Si realmente cambia lanza el evento correspondiente.

Controlador<T>	
Clase base para los controladores. Implementa Controlador<T>	
Log: ILog	Acceso al fichero de log.
Diposit: IDiposit<T>	Depósito principal utilizado por el controlador para acceder a los datos.
DBManager: IDBManager	DBManager utilizado.
Context: IContextDB	Contexto de la base de datos en el que trabaja el controlador.

Diposit<T>	
Clase base para los depósitos que trabajan usando NHibernate como medio de acceso a los datos. Implementa IDiposit<T>	
DBManager: IDBManager	DBManager utilizado.
Context: IContextDB	Contexto de la base de datos en el que trabaja el controlador.

DBManager	
Implementa IDBManager. Es la implementación del interface para trabajar con NHibernate.	
CadenaConexio: string	Cadena de conexión utilizada para abrir la conexión con la base de datos.
Shema: shema	Shema de la base de datos a utilizar. Si el valor no está definido se busca automáticamente.

ContextDB	
Implementa IContextDB. Es la implementación del interface para trabajar con NHibernate.	
Sessio: ISession	Contiene la sesión de NHibernate utilizada.

CodiEAN13	
Estructura que permite trabajar con los códigos EAN13 como si fueran objetos nativos. Permite compararlos, validarlos, formatearlos y convertirlos.	
Valor: string	Convierte el EAN13 en una cadena.

PaginaBase	
Clase que deriva de System.Web.UI.Page y sirve de base para todas las páginas web del proyecto	
FerTancarFinestres()	Cierra todas las ventanas abiertas de la página.
FinestraOberta()	Permite indicar que hay una ventana abierta.
TancarFinestres()	Permite a las clases derivadas implementar el código necesario para cerrar sus ventanas.
AjustaReadOnly(HtmlControl control, bool readOnly)	Marca un control como sólo lectura o lectura/escritura
Log : ILog	Acceso al fichero de log.
Usuari :Usuari	Usuario actual.
MostrarError(string msg)	Muestra un diálogo con un mensaje de error.

PaginaBaseManteniment<T>	
Clase que deriva de PaginaBase y sirve de base los mantenimientos de objetos del tipo T	
RegistraNavegador(IControlNavegacio controlNavegacio)	Registra el control de navegación de registros/objetos que se va a utilizar.
RegistraControlRegistres(WebControl btnCercar)	Registra las funciones utilizadas para crear y borrar objetos.
RegistraCercador(HtmlControl codi, HtmlControl focus, WebControl btnCercar)	Registramos el control que provocará la búsqueda del objeto cuando pierda el foco de entrada.
GestionaNavegacio(object sender, NavegadorEventArgs e)	Gestiona las órdenes dadas del navegador controlado si se tiene que actualizar el objeto o no.
CercarCodi()	Efectúa la búsqueda del objeto y si lo encuentra lo muestra por pantalla.
EsNou(): bool	Indica si el objeto que estamos editando es nuevo o no.
Netejar()	Limpia la pantalla y efectúa las tareas necesarias para poder editar otro objeto.
ExtreuCodi()	Extrae de la pantalla el código de negocio del objeto. Sobrescribir en las clases derivadas.
CargarCanvis(IControlador<T> controlador)	Carga los valores que hay en pantalla sobre el objeto actual. Sobrescribir en las clases derivadas.
NouCodi(IControlador<T> controlador)	Se ejecuta cuando se está entrando un nuevo código y por tanto vamos a editar un nuevo objeto. Sobrescribir en las clases derivadas si necesitamos inicializar los objetos con nuestros valores.
MostrarObjecte(IControlador<T> controlador)	Mostramos las propiedades del objeto por pantalla.
ModoLectura(bool lec)	Se ejecuta cuando cambiamos el modo de trabajo de lectura a lectura/escritura.

CrearControlador():IControlador <T>	Crea el controlador que se va a utilizar en el mantenimiento. Sobrescribir en las clases derivadas.

ServeiPerfums	
Implementa el servicio web de perfumes.	
LlistaPerfums():ArrayList	Devuelve una lista con los datos de los perfumes.

3.4. Diseño de la interface de usuario

3.4.1. Introducción

La interface de usuario de este proyecto se ha realizado mediante páginas Web. Uno de los objetivos propuestos ha sido que la apariencia y el comportamiento de la Web fuera lo más parecido posible al de los interfaces Windows. Para conseguirlo la Web se ha basado en ASP.NET AJAX y en el JavaScript.

Se han desarrollado una serie de controles visuales para mejorar la apariencia y el funcionamiento de la página web. Estos controles son de diferentes tipos y todos tienen en común que funcionan correctamente con el AJAX.

3.4.2. ASP.NET AJAX

AJAX es el acrónimo de Asynchronous JavaScript And XML y es una técnica de desarrollo web para crear aplicaciones interactivas. Su principal virtud es que permite a los desarrolladores crear páginas web que no precisan de los típicos refrescos de pantalla cada vez que tienen que comunicarse con el servidor. El AJAX se basa en estándares Web bien conocidos como el JavaScript, XML, HTML y CSS.

El núcleo central del AJAX es el objeto JavaScript XMLHttpRequest. Mediante dicho objeto el JavaScript puede interactuar con el servidor sin que por ello se tenga que volver a cargar la página. Todos los navegadores modernos soportan dicho objeto con las mismas propiedades y métodos, pero varía la manera en que lo crean. Microsoft introdujo el objeto XMLHttpRequest en el Explorer 5.0 como un objeto ActiveX, Mozilla copio el diseño de Microsoft, pero lo implementó como un objeto nativo del JavaScript. El resto de navegadores han seguido el ejemplo de Mozilla.

ASP.NET AJAX es la librería de Microsoft para aprovechar el AJAX en la plataforma ASP.NET. Ofrece en la parte cliente una completa librería AJAX multiplataforma con soporte a múltiples navegadores, con JavaScript basado en componentes y controles y además con una serie de objetos de soporte en JavaScript. ASP.NET AJAX ofrece también soporte en la parte servidor que permite ofrecer servicios al JavaScript vía métodos de página o bien vía servicios Web. Otra característica remarcable es la facilidad que ofrece para convertir páginas web existentes para que se beneficien del AJAX.

En este proyecto se ha utilizado extensamente el AJAX para mejorar la experiencia del usuario. Todas las páginas web hacen uso del mismo con la excepción de la de login. Los controles utilizados y el lugar en donde se utilizan se detallan a continuación:

- **ScriptManager**: Es un componente no visual necesario para que una página web pueda utilizar el ASP.NET AJAX. Su función es la de cargar, inicializar y configurar las librerías JavaScript necesarias para el correcto funcionamiento del AJAX. Además permite incluir desde código en el servidor métodos o librerías enteras en JavaScript evitando incompatibilidades con el AJAX. Debido a que su utilización es imprescindible para utilizar el

AJAX se ha incluido dicho control en la página maestra del proyecto (Site.Master) y por tanto es utilizado por todas las páginas web.

- **UpdatePanel**: Este control es sin lugar a dudas en método más sencillo de utilizar el AJAX en una web. Este control se presenta como un contenedor de controles o panel que intercepta las llamadas convencionales al servidor de los controles que contiene, substituye dichas llamadas por llamadas AJAX y finalmente devuelve las órdenes del servidor utilizando el mismo mecanismo. El efecto es que los controles que hay en dicho panel pueden llamar al servidor sin que ello provoque un refresco de la pantalla. Lo más interesante es que los controles contenidos en el panel son controles estándar del ASP.NET que no están preparados por si solos para trabajar con AJAX. Este proyecto **ha utilizado dicho control en todas las páginas** mejorando mucho así la interactividad con el usuario. El la Figura 27 se puede ver el uso del control UpdatePanel, cuando el usuario introduce un código de perfume y sale del campo, se provoca una llamada al servidor para que busque el perfume y si lo encuentra muestre sus datos en pantalla. Si no se utilizara el UpdatePanel el navegador borraría la pantalla y la reharía con los datos enviados por el servidor, el UpdatePanel evita esto y sólo varía el contenido de los campos mejorando visualmente todo el proceso.

- **UpdateProgress**: Las actualizaciones de las páginas web pueden tardar más o menos dependiendo de la velocidad de las comunicaciones o bien de la carga del servidor. Por regla general el navegador informa visualmente con algún texto o animación de que está cargando la página y por tanto el usuario sabe en cierto modo lo que está ocurriendo. Cuando utilizamos AJAX el navegador no nos informa de que hay algún tipo de comunicación, ni de que esté pendiente de recibir dados del servidor por lo que el usuario no sabe lo que está ocurriendo. UpdateProgress es un control que detecta cuando una petición del cliente al servidor tarda más de lo estipulado y presenta al usuario un mensaje definido por nosotros informándole de ello. Este proyecto hace uso de este control en varias páginas web que por la complejidad de sus peticiones son susceptibles a que el servidor tarde más de lo normal en responder, concretamente se usa en:

    - **Página de inicio (Default.aspx):** Se utiliza debido a que en esta página se pueden lanzar los procesos de conexión con el servicio web del proveedor y los de actualización del stock.

    - **Mantenimiento Perfumes (MantPerfums.aspx)**: Es quizás la página más pesada desde el punto de vista de los controles con lo que el UpdatePanel que hay en ella envía mucha información al servidor.

    - **Evolución de existencias (EvolExist.aspx)**: Muestra una estadística de la evolución de las existencias entre dos fechas y por tanto quizás tarde más de lo normal en devolver los datos al cliente.

    - **Evolución de precios (EvolPreus.aspx):** Muestra una estadística de la evolución de los precios entre dos fechas y por tanto quizás tarde más de lo normal en devolver los datos al cliente.

- **Métodos de página**: Son aquellos métodos situados en la página web, en la parte del servidor, que pueden ser llamados por el JavaScript utilizando AJAX. Desde el punto de vista del servidor se tratan de una forma similar a los servicios Web, aunque de una forma algo más simplificada. Los métodos de página son sin duda la forma más eficiente de comunicar el JavaScript y el servidor ya que sólo se transmite lo que realmente necesitamos. En este proyecto se utilizan en varias páginas especialmente para verificar la validez de un dato introducido consultando la base de datos y si es correcto mostrar su descripción. Así por ejemplo en la Figura 29 se muestra al usuario entrando el código de un perfume en un movimiento de mercancía. Cuando ha entrado el código del perfume el JavaScript llama a un método en el servidor para que le devuelva el nombre del perfume. Si el nombre no existe muestra un error y si existe lo muestra en pantalla obteniendo así el usuario confirmación inmediata de que no se está equivocando. El uso de métodos de página por parte del

JavaScript vía AJAX tiene la ventaja sobre el UptadePanel de que son mucho más rápidos, pero también más complejos a la hora de utilizarse. El proyecto hace uso de los métodos de página en:

- **Mantenimiento de perfumes (MantPerfums.aspx)**: Utilizados para validar la familia y línea del perfume y cargar su correspondiente descripción. Permite verificar el EAN13.
- **Movimientos de almacén (MantMov.aspx)**: Utilizado para validar el código del perfume entrado en cada línea y cargar su correspondiente descripción.
- **Evolución de existencias (EvolExist.aspx)**: Utilizado para validar el código del perfume y cargar su correspondiente descripción.
- **Evolución de precios (EvolPreus.aspx)**: Utilizado para validar el código del perfume y cargar su correspondiente descripción.

- **CalendarExtender**: Este control no forma parte de la distribución oficial de ASP.NET si no de una librería semioficial creada por empleados de Microsoft llamada AJAX Control Toolkit. Esta librería contiene una serie de controles que extienden la funcionalidad de los controles estándar de ASP.NET mediante el AJAX y JavaScript. De esta librería este proyecto sólo usa el CalendarExtender para mostrar al usuario un calendario para que pueda seleccionar fácilmente una fecha, tal como muestra la Figura 28. Las páginas que usan este control son:
  - Movimientos de almacén (MantMov.aspx)
  - Listado de movimientos (LlistatMov.aspx)
  - Evolución de existencias (EvolExist.aspx)
  - Evolución de precios (EvolPreus.aspx)

- **Facilidades para el uso del JavaScript**: ASP.NET AJAX define una serie de objetos y utilidades para facilitar la programación en JavaScript. Este proyecto únicamente hace uso de la útil función **$get(id)** que devuelve un control según su identificador. La principal ventaja de este método con respecto al estándar del JavaScript es que es compatible con todos los navegadores. Su uso en el proyecto está generalizado en todo el código en JavaScript.

El uso de AJAX en este proyecto ha sido por regla general sencillo, especialmente respecto a la utilización del control UpdatePanel. La integración del AJAX con ASP.NET es prácticamente completa lo que sin duda ha facilitado su utilización. El único problema encontrado ha sido en el mantenimiento de perfumes, al utilizar el control FileUpload para subir al servidor una fotografía de un perfume ya que este control no es compatible con AJAX.

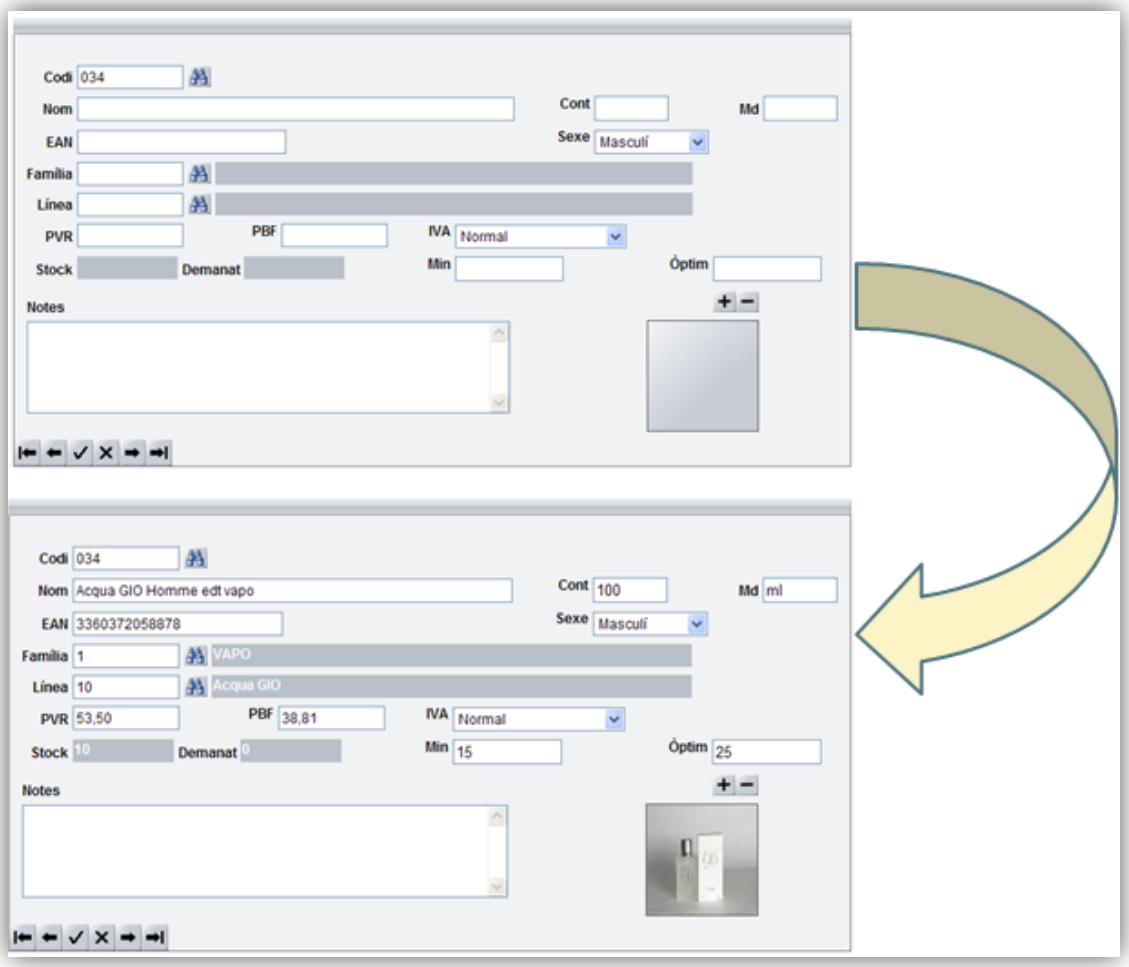


Figura 27: Uso del UpdatePanel de AJAX

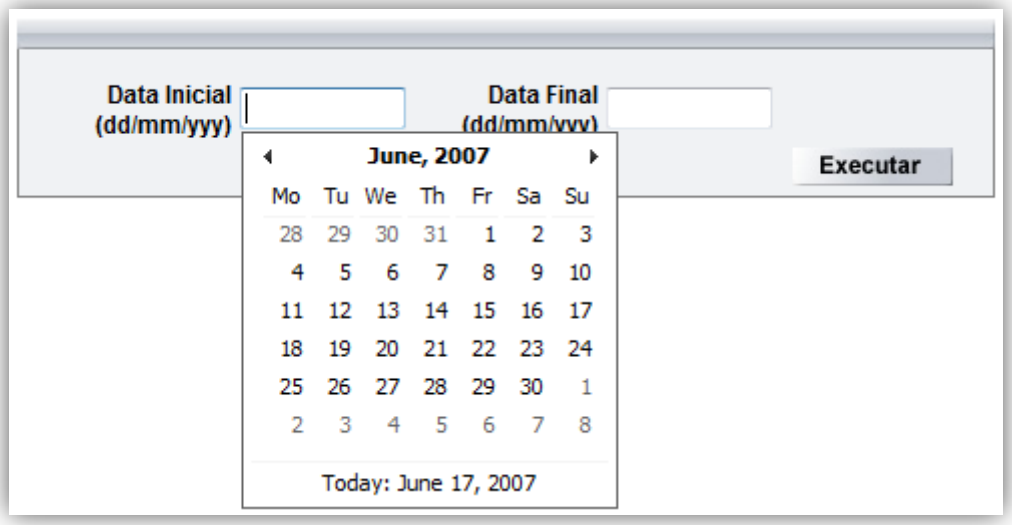


Figura 28: Ejemplo de uso del CalendarExtender

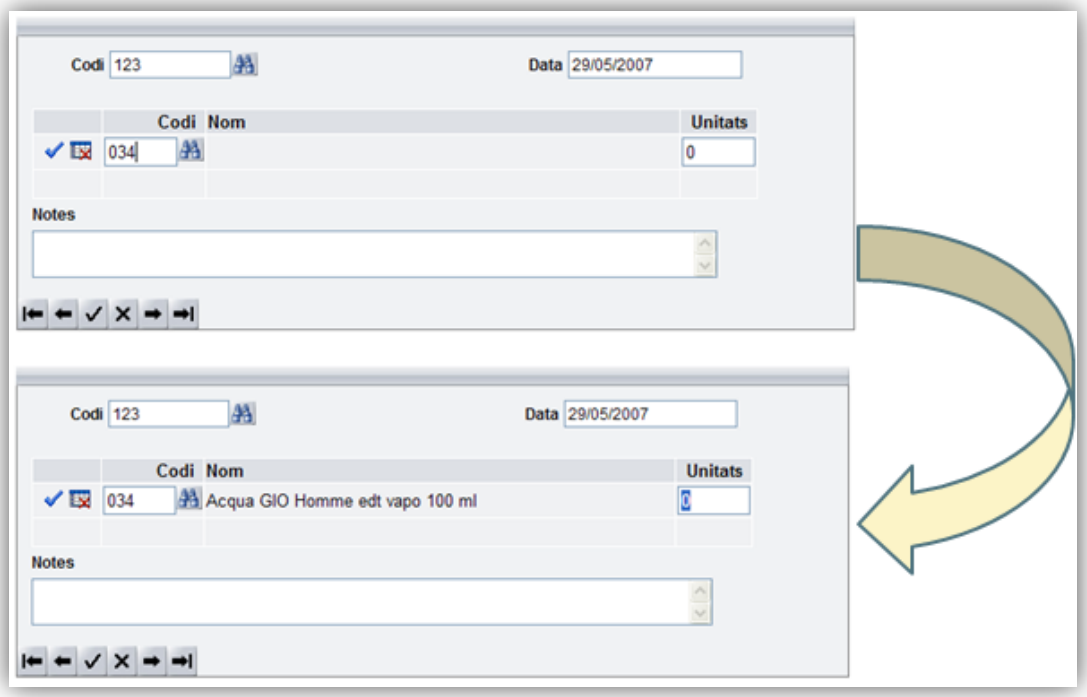


Figura 29: Uso de AJAX y JavaScript

3.4.3. Controles de usuario

Se han diseñado varios controles de usuario para facilitar la programación y mejorar la calidad de la interface de usuario. Estos controles son de varios tipos:

- **Menús**. Hay tres tipos de menús. El principal, como muestra Figura 30, común en todas las pantallas de la Web, el submenú, Figura 31, que varía según la opción del menú principal seleccionada y finalmente el menú contextual, Figura 32, que contiene las opciones propias de cada pantalla.

- **Buscadores**: Se representan como ventanas que permiten buscar un tipo de registro determinado como por ejemplo perfumes, familias, líneas, etc. Cuando se abre un buscador nos enseña los primeros registros que hay en la base de datos, podemos ir viendo el resto de datos pulsando los links correspondientes. Se pueden efectuar búsquedas por el campo nombre y ordenar la información pulsando sobre la columna que queremos utilizar para dicho orden. En el ejemplo de la Figura 33 se puede ver el buscador de perfumes filtrando la información por parte del nombre.

- **Navegador de registros**: El navegador de registros, Figura 34, es un sencillo control que facilita al usuario el desplazarse secuencialmente entre los registros que este visualizando o modificando.

- **Mostrar Estado**: Hay varios controles, como el de la Figura 35, utilizados para mostrar la situación actual de los perfumes. Hay varios de ellos y cada uno muestra un aspecto distinto del estado de los perfumes, como por ejemplo los últimos cambios de precio, la situación del stock, si hay pedidos pendientes, etc.

Figura 30: Menú principal

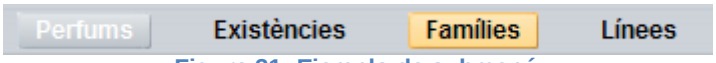


Figura 31: Ejemplo de submenú

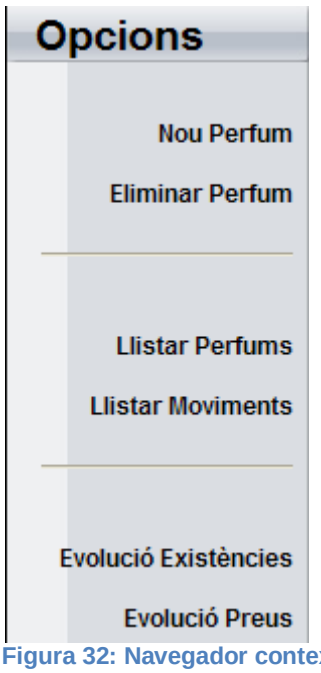


Figura 32: Navegador contextual

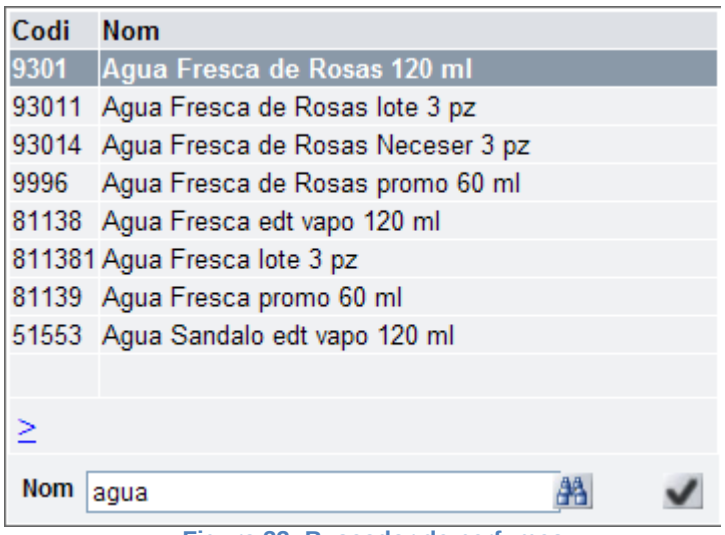


Figura 33: Buscador de perfumes

Figura 34: Navegador de registros

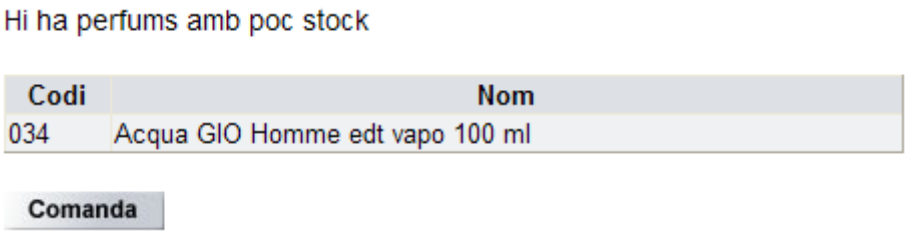


Figura 35: Mostrar estado

3.4.4. Handlers

Los handlers son una pieza fundamental dentro de la arquitectura de ASP.NET. Cada vez que ASP.NET recibe una petición HTTP busca un handler que sea capaz de procesarla y devolver al usuario la respuesta correspondiente. Estos handlers son clases que implementan el interface IHttpHandler. Por ejemplo las páginas .aspx, comunes en ASP.NET, son servidas por el PageHandlerFactory que se encarga de buscar, compilar y ejecutar el código de la página pedida.(Esposito, 2006)

La arquitectura de ASP.NET es lo suficientemente flexible para permitir que se puedan definir nuevos handlers de una manera extremadamente sencilla, esto permite que manejemos ciertas peticiones de los usuarios de una forma no estándar.

En este proyecto se define un handler que devuelve una imagen del perfume pedido con el tamaño indicado en la petición. Este handler se utiliza para mostrar la imagen en miniatura del perfume en el mantenimiento de perfumes y si el usuario pulsa encima de dicha imagen se vuelve a utilizar para

mostrarla a mayor tamaño. Desde el punto de vista del navegador la respuesta del handler es simplemente una imagen JPEG.

Otro tipo de handler utilizado en el proyecto devuelve un listado en formato PDF. El código del handler crea el un fichero PDF en memoria utilizando la librería de código libre **Gios Pdf.NET** y devuelve al navegador la página creada sin necesidad de ficheros temporales.

3.4.5. Diagrama de flujo entre pantallas

El interface de usuario está diseñado de tal forma que prácticamente se puede ir desde cualquier pantalla a cualquier otra directamente. El diagrama de la Figura 36 recalca la facilidad de navegación de la web y muestra la relación que hay entre pantallas debido al resultado de un proceso. Este tipo de transiciones son mínimas en este proyecto ya que el uso del JavaScript y del AJAX ha permitido que las páginas muestren por sí mismas los resultados de sus procesos sin necesidad de recurrir a otras páginas.

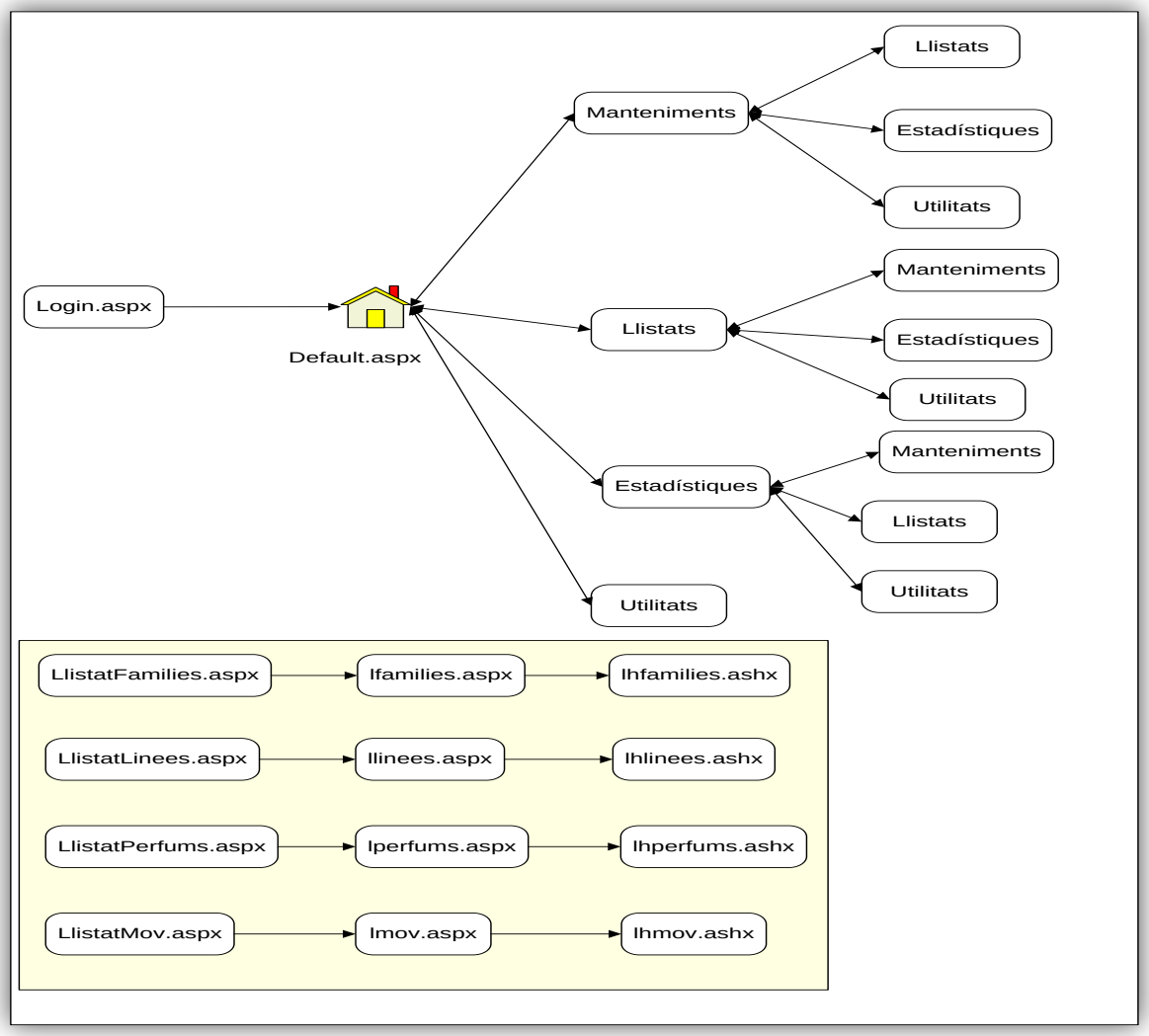


Figura 36: Flujo de pantallas

3.4.6. Detalle de las páginas web

A continuación se mostraran varias de las páginas que componen la Web, no están todas por motivos de espacio, pero sí las quizás más representativas.

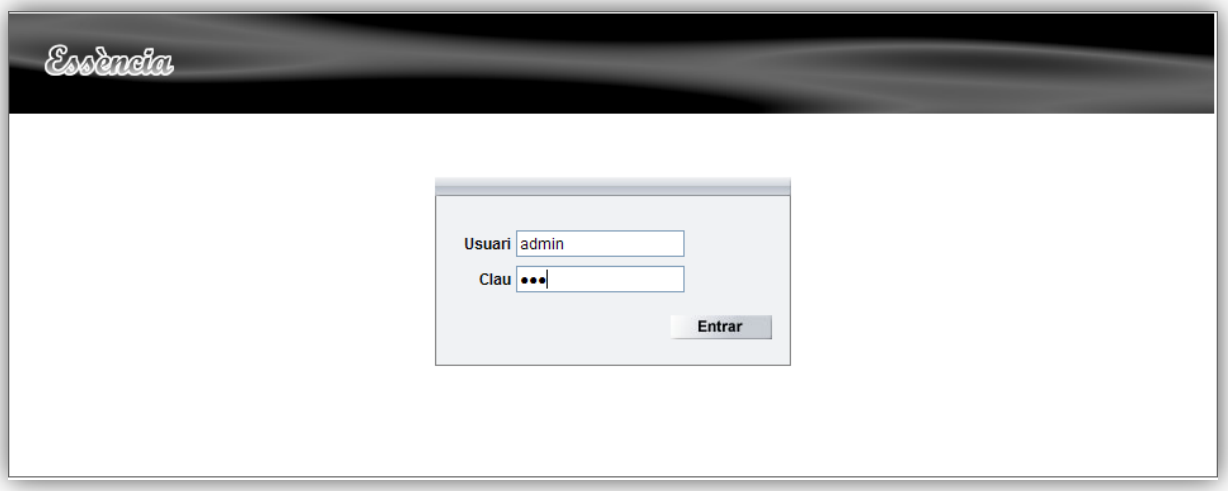


Figura 37: Página de login

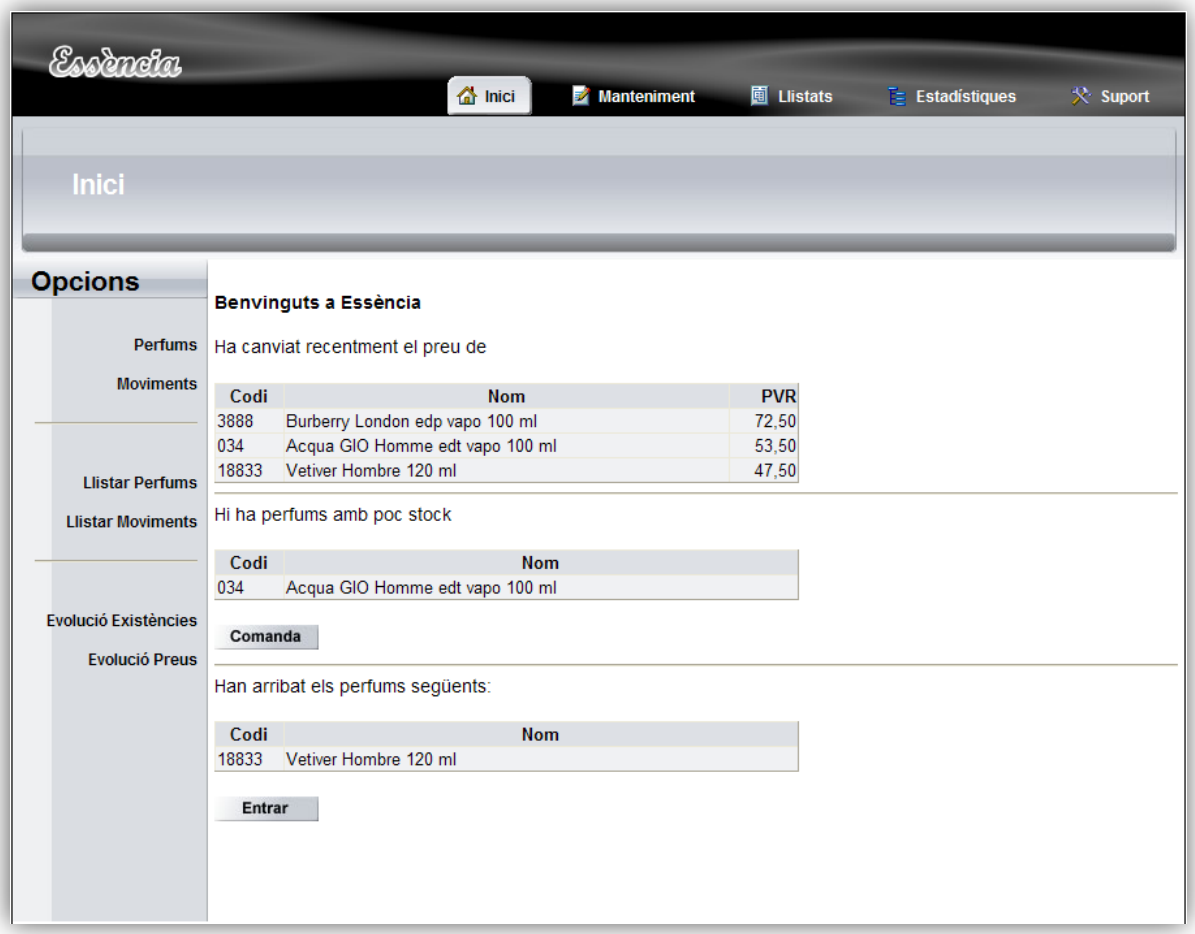


Figura 38: Página de inicio

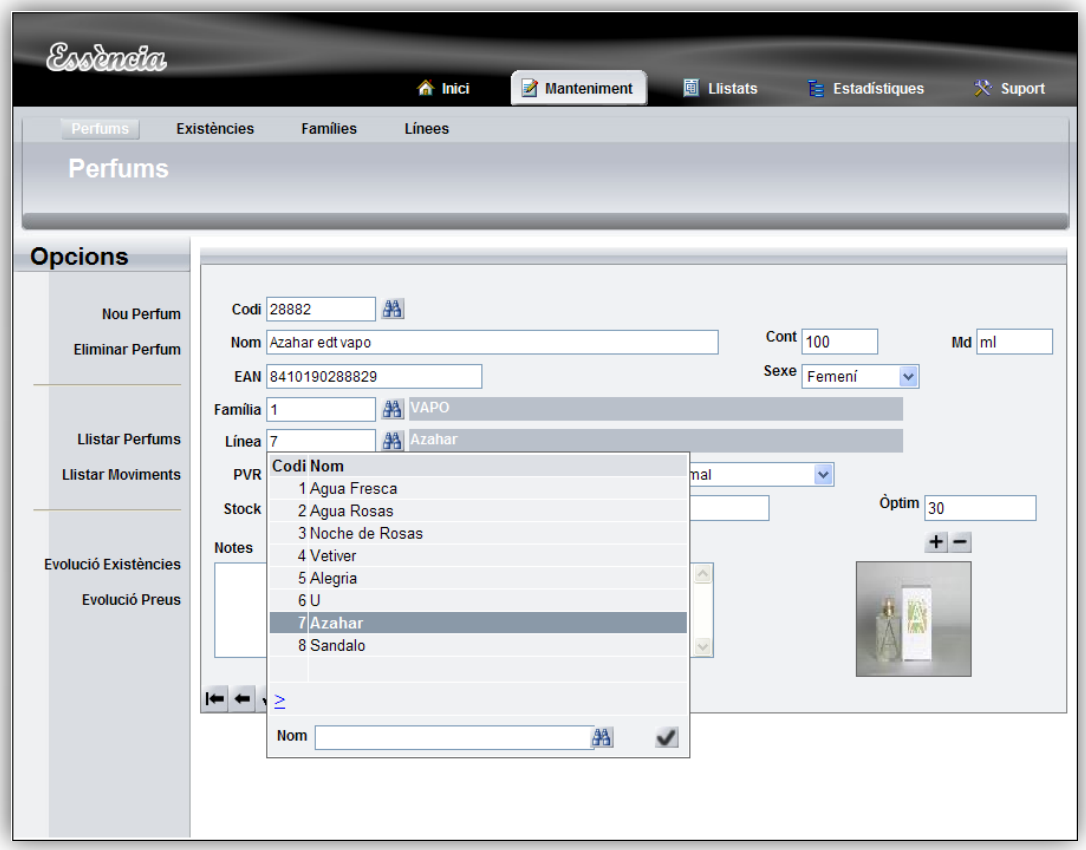


Figura 39: Página mantenimiento de perfumes

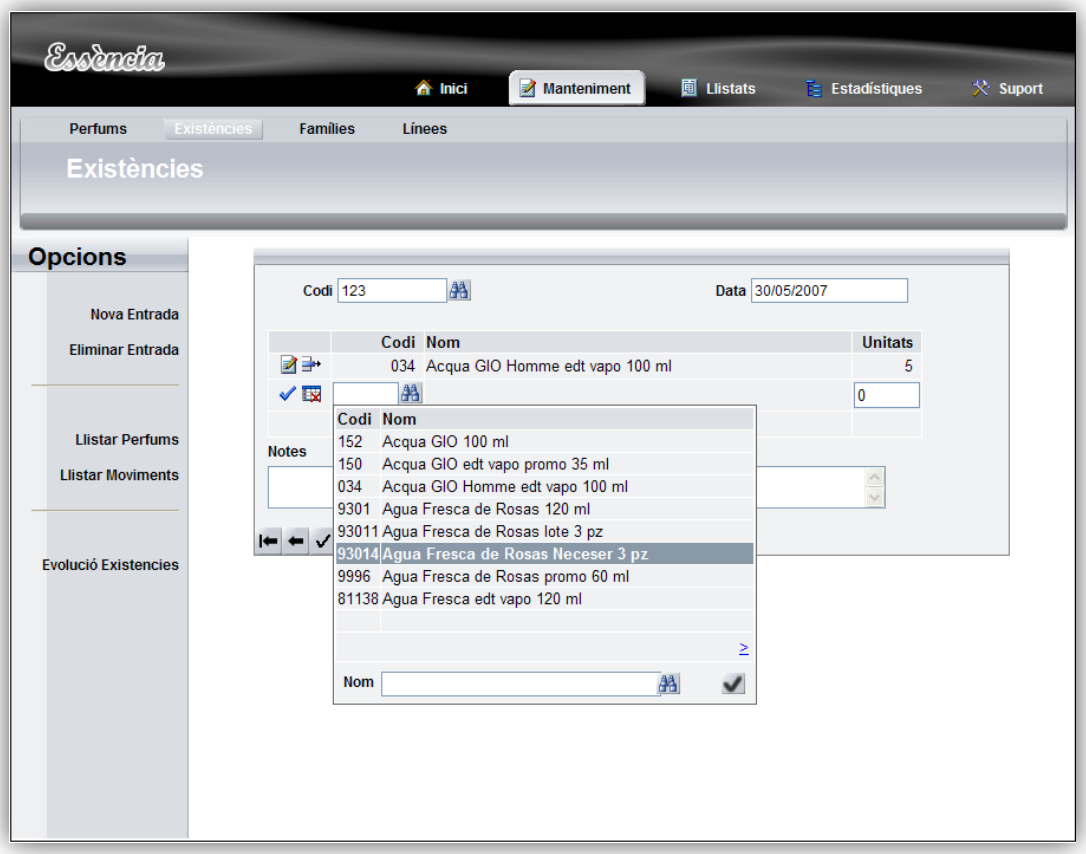


Figura 40: Página variación del stock

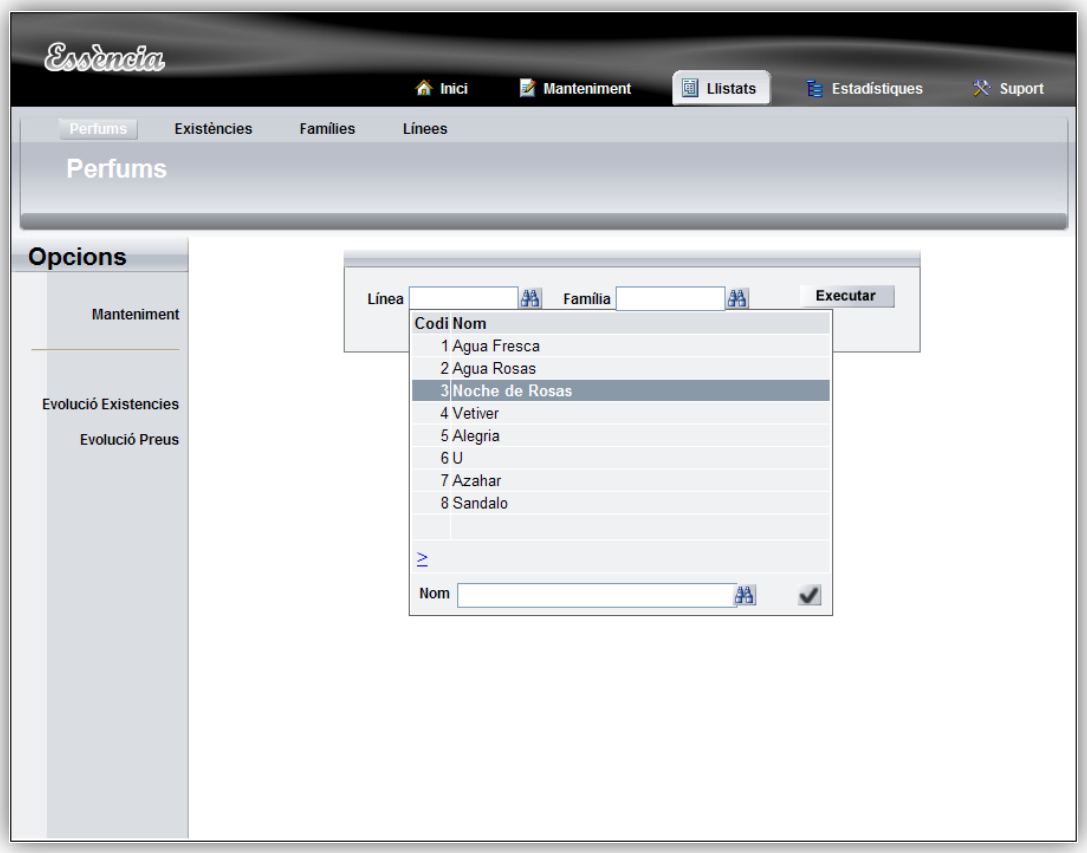
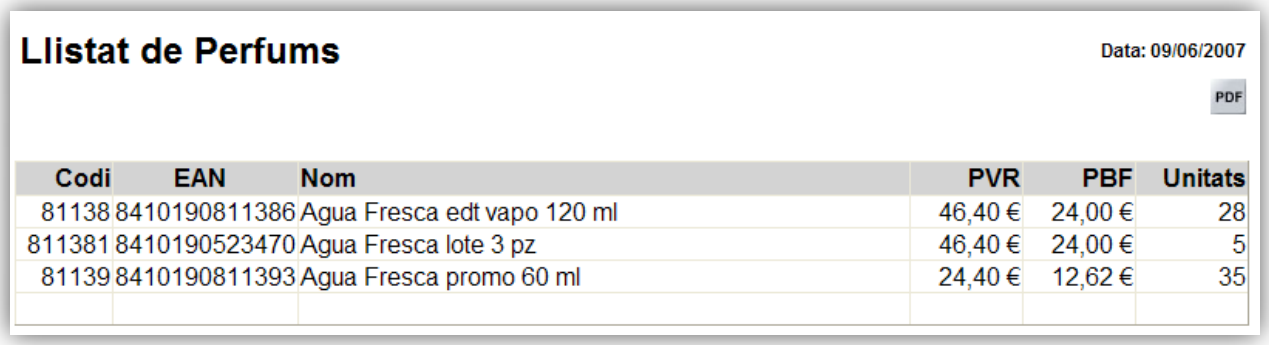


Figura 41: Página selección listado perfumes



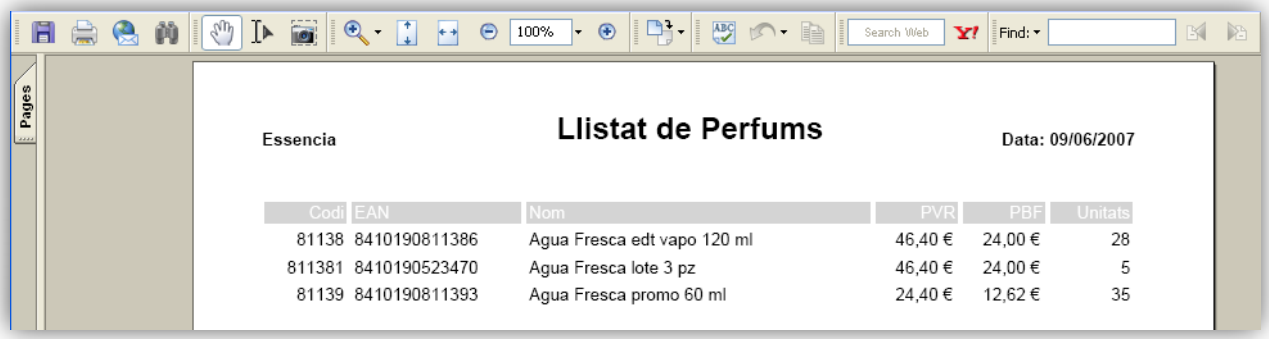
Llistat de Perfums

Data: 09/06/2007

PDF

Codi	EAN	Nom	PVR	PBF	Unitats
81138	8410190811386	Agua Fresca edt vapo 120 ml	46,40 €	24,00 €	28
811381	8410190523470	Agua Fresca lote 3 pz	46,40 €	24,00 €	5
81139	8410190811393	Agua Fresca promo 60 ml	24,40 €	12,62 €	35

Figura 42: Detalle listado de perfumes



Pages

Essencia

Llistat de Perfums

Data: 09/06/2007

Codi	EAN	Nom	PVR	PBF	Unitats
81138	8410190811386	Agua Fresca edt vapo 120 ml	46,40 €	24,00 €	28
811381	8410190523470	Agua Fresca lote 3 pz	46,40 €	24,00 €	5
81139	8410190811393	Agua Fresca promo 60 ml	24,40 €	12,62 €	35

Figura 43: Detalle listado de perfumes (PDF)

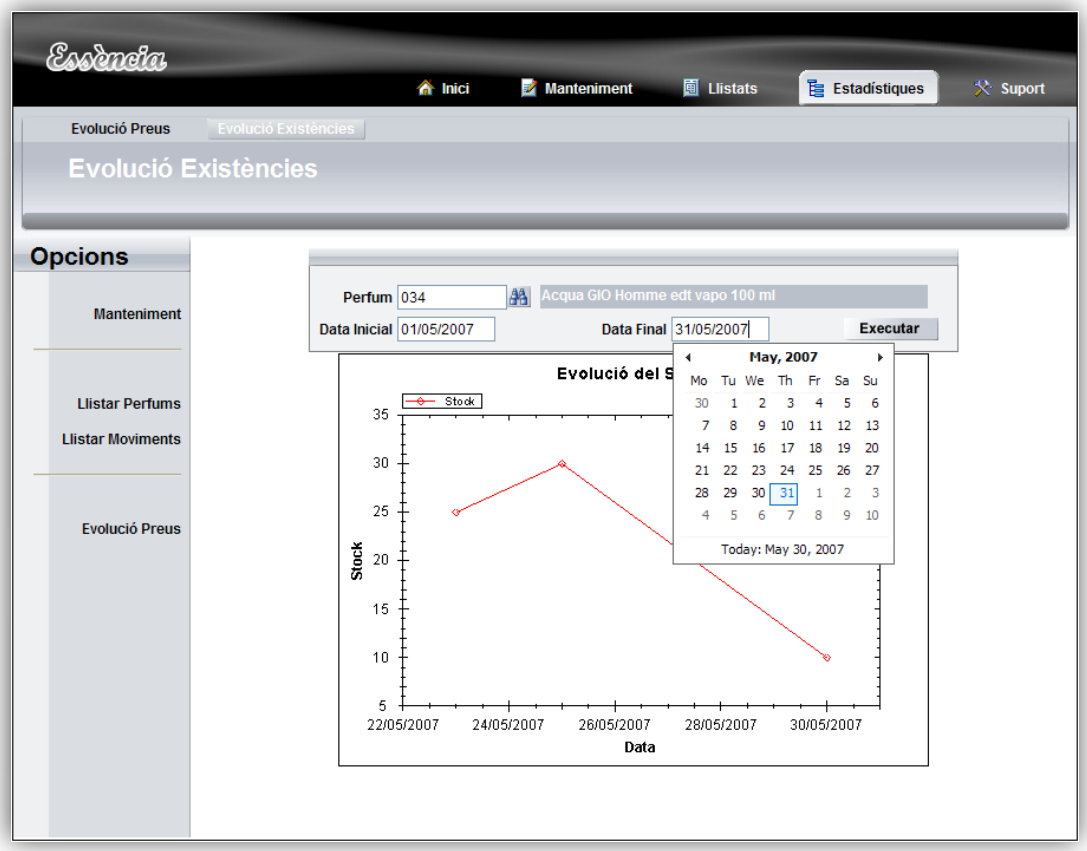


Figura 44: Página estadística de la evolución de existencias

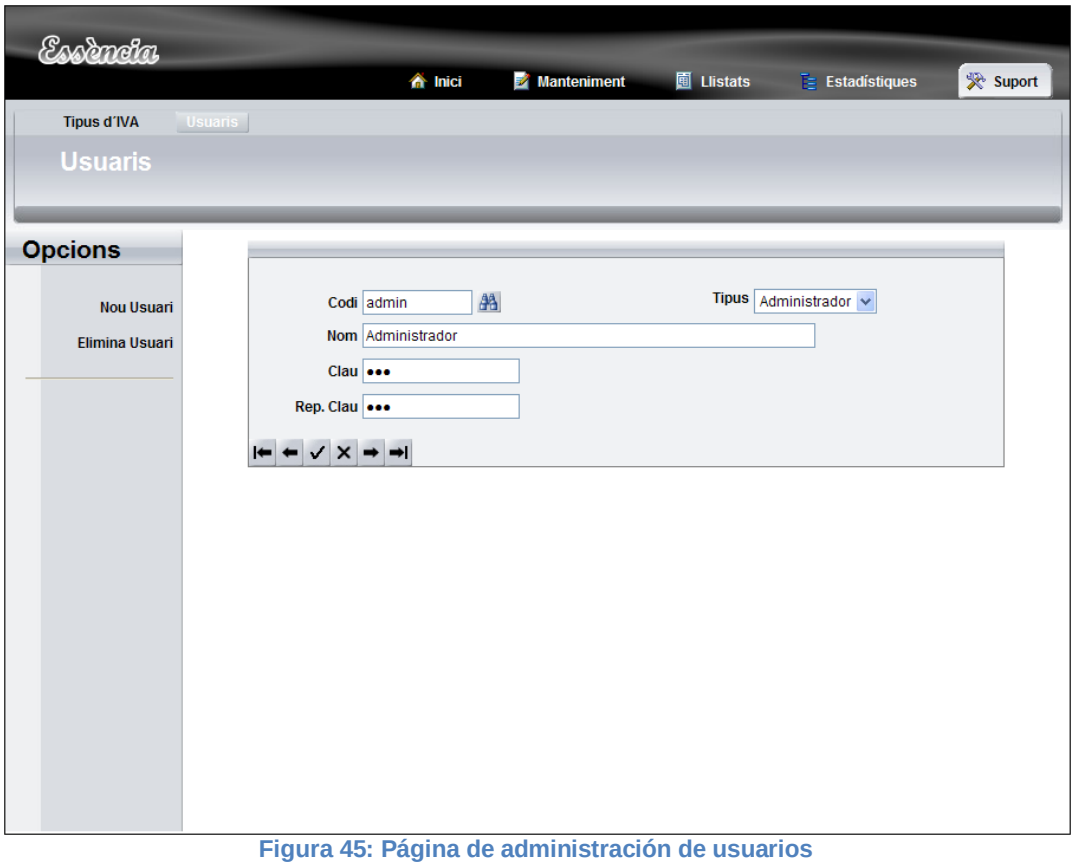


Figura 45: Página de administración de usuarios

3.5. Diseño de la base de datos

En la Figura 46 podemos ver el diagrama ER con el modelo lógico de la base de datos utilizada en el proyecto y el la Figura 47 el diagrama físico.

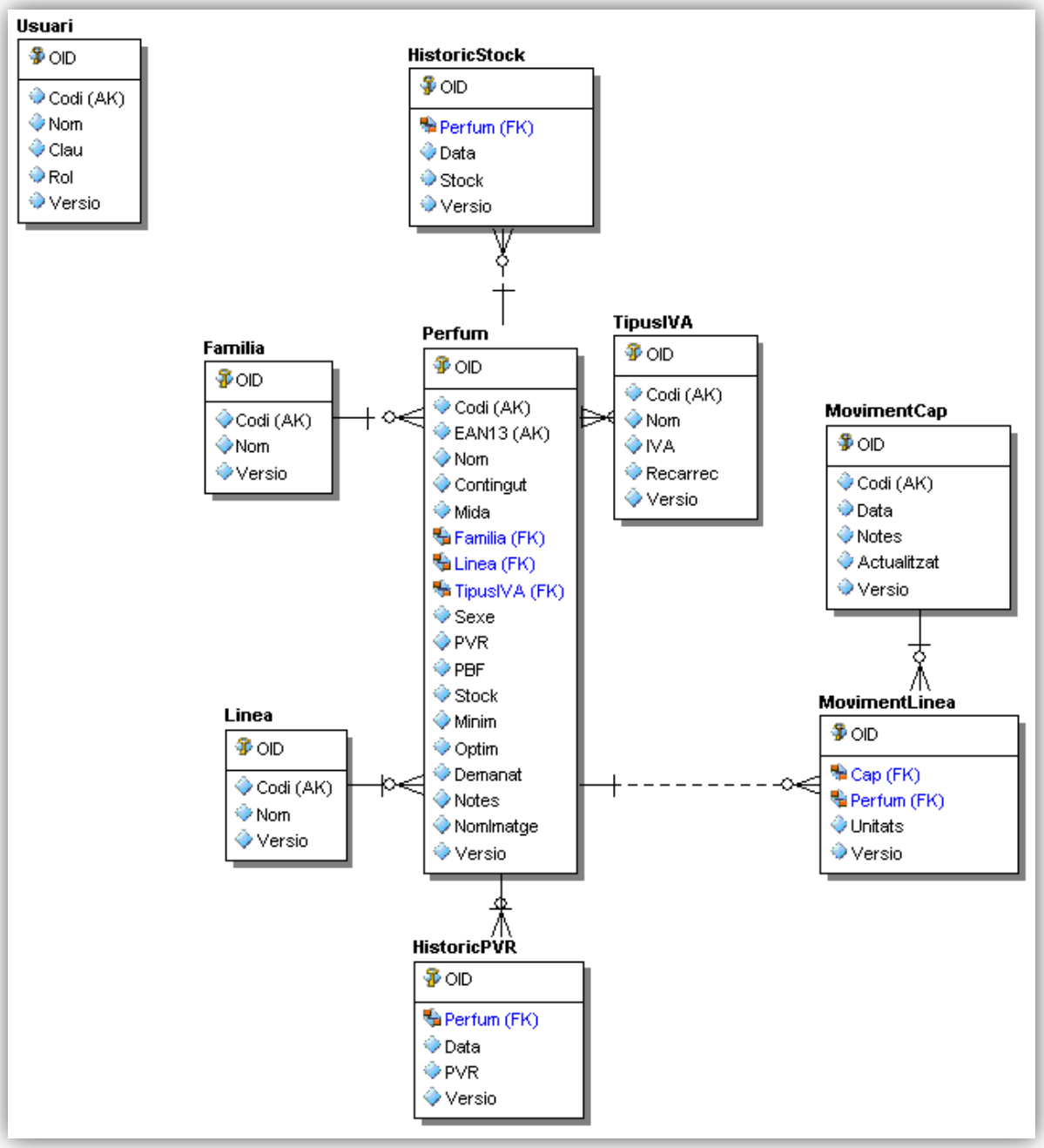


Figura 46: Diagrama lógico (ER) de la base de datos

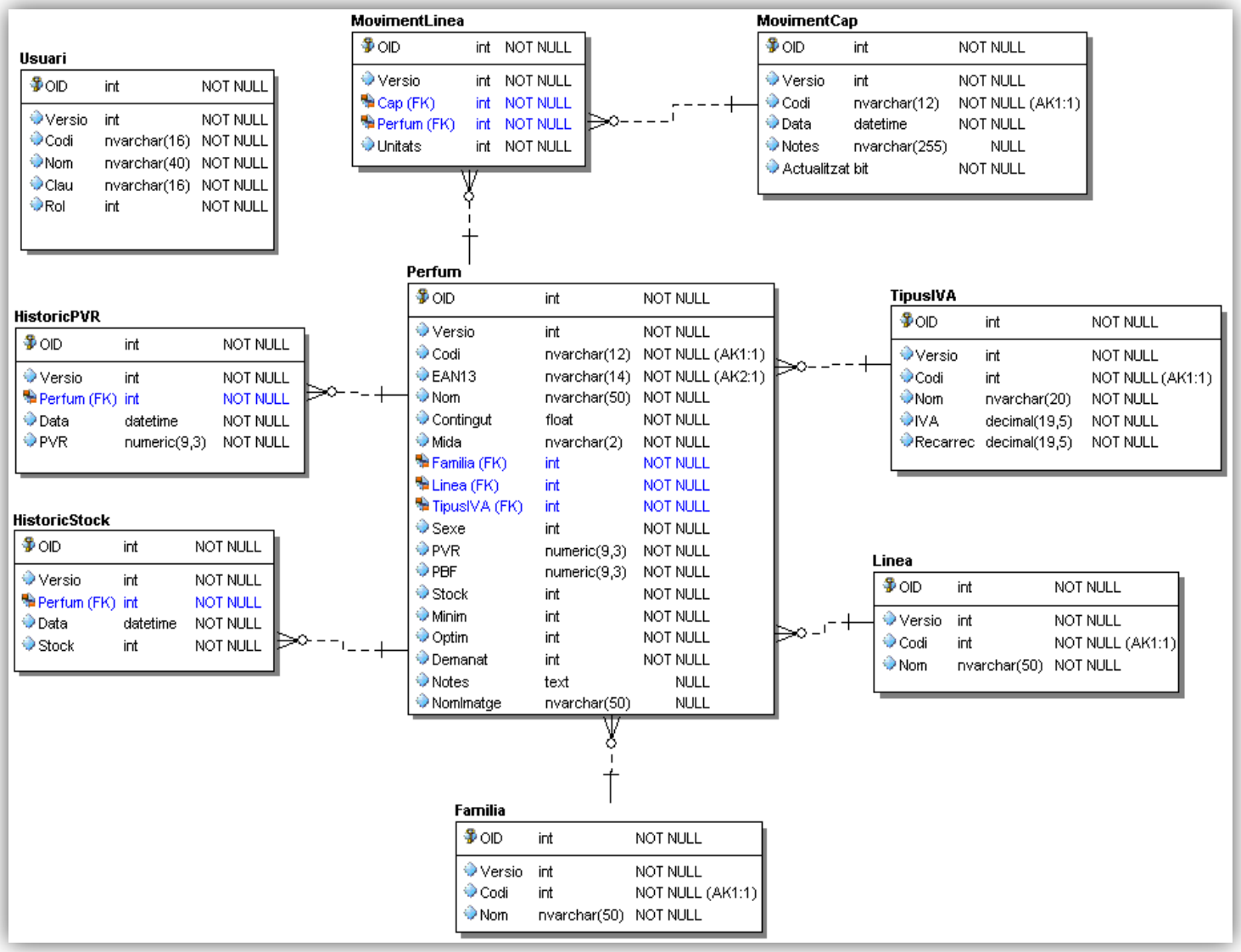


Figura 47: Diagrama físico de la base de datos

4. Conclusiones

El desarrollo de este proyecto me ha enseñado que hoy por hoy es posible crear interfaces de usuario con páginas Web que tienen poco que envidiar a los sofisticados interfaces que se pueden encontrar en Windows. Una tecnología clave en esta realidad es sin duda el AJAX que ha cambiado la forma en la que actúan las páginas Web. Un punto negativo es la excesiva dependencia del AJAX de la velocidad de conexión y de la velocidad del servidor Web utilizado ya que si el servidor responde lentamente los interfaces creados con AJAX pueden resentirse mucho.

El esfuerzo para programar los interfaces web es mucho mayor que a la hora de hacerlo para Windows y desde mi punto de vista es debido al gran número de tecnologías que hay que utilizar para conseguir páginas con un gran grado de interactividad y sobre todo a las diferencias existentes entre los diversos navegadores que hay disponibles. Estas diferencias obligan a ir probando las páginas con varios navegadores para estar seguros que todo funciona razonablemente bien en todos ellos con lo que los tiempos de desarrollo y depuración aumentan considerablemente.

Me he encontrado muy pocas limitaciones a la hora de crear la infraestructura necesaria para este proyecto, hay una gran cantidad de proyectos de código libre que pueden utilizarse para ello. Personalmente me decante por NHibernate por su madurez en Java, pero hubiera podido escoger otro de los múltiples ORM disponibles para Microsoft .NET. Algo similar pasa con el resto de librerías utilizadas en el proyecto, tuve el lujo de poder elegir cada una de ellas entre varias opciones. Desde mi punto de vista creo que todo esto significa que la plataforma .NET ya ha alcanzado suficiente notoriedad como para poder considerarse como consolidada en el mercado.

Mi opinión sobre las librerías utilizadas es muy positiva y la detallo a continuación:

- **ASP.NET AJAX**: Es un producto muy completo y sólido a pesar de ser la primera versión sacada al mercado. Su utilización ha sido sencilla y no me ha causado problemas salvo en el caso del control FileUpload que no es compatible. La utilización de AJAX permite crear páginas Web muy dinámicas y completas sin necesidad de utilizar JavaScript, pero un abuso de llamadas al servidor para ejecutar código que sería fácilmente programable en JavaScript puede causar problemas de rendimiento especialmente una vez puesto en producción. Desde mi punto de vista los controles que se usen en los UpdatePanel deberían producir las mismas llamadas al servidor que cuando no se usa el AJAX para evitar saturarlo, si necesitamos información constante del servidor es preferible utilizar el AJAX para comunicar el JavaScript directamente con un servicio Web o un método de página y finalmente seguir utilizando el JavaScript para aquello que realmente no precise del servidor.

- **NHibernate**: Es un ORM muy completo que nos da una libertad a la hora de definir nuestro modelo de datos prácticamente absoluta. El mapeo de nuestros objetos con la base de datos se realiza mediante un fichero de configuración XML, este fichero es crítico ya que desde el indicaremos no solo donde se guardaran nuestros objetos sino como los tratará NHibernate. El rendimiento es excelente y las posibilidades consultar a la base de datos magnificas. Personalmente es una librería que me gusta mucho y la verdad es que no le encuentro inconvenientes remarcables.

- **Spring.Net**: La primera vez que utilicé esta librería reconozco que lo hice más por la curiosidad que por que viera algo realmente útil en ella. Con el tiempo me ido dando cuenta que al crear los objetos sin dependencias internas y utilizando Spring.Net para resolverlas me estaba ayudando a crear objetos mucho más reutilizables y con ellos aplicaciones mucho más fáciles de extender y mantener. No todo es sencillo, Spring.Net requiere del uso de a veces largos y complejos ficheros de configuración XML que en el caso de contener algún error puede ser difícil de solucionar. La utilización de Spring.Net en este proyecto es simple y no muestra todo el potencial de que dispone, pero sí puede hacernos una idea de su capacidad.

- **ZedGraph**: Es una librería gráfica en 2D realmente interesante. Dispone de multitud de opciones a la hora de sacar las gráficas y quizás el único problema que le veo es debido precisamente a esas múltiples opciones que a veces hace que sea difícil encontrar lo que se está buscando. El uso de la librería en este proyecto sólo es superficial ya que no se precisaba de gráficas muy complejas.

- **Gios PDF.NET**: Es una librería para crear documentos PDF con múltiples opciones, pero que destaca de sobre manera por su sencillez a la hora de crear documentos basados en tablas. Me decante por esta librería en vez de otras quizás más completas y potentes precisamente por esa sencillez a la hora de trabajar con tablas ya que mi intención era simplemente la de crear listados y las tablas fueron el método más simple para hacerlo. Su utilización ha sido muy sencilla y su funcionamiento muy correcto por lo que creo que fue una buena elección.

Como final indicar que se han podido cumplir todos los objetivos fijados al principio del proyecto, resultándome especialmente satisfactorios todo el tema de desarrollo web que desconocía en gran medida antes de empezar. Me hubiera gustado disponer de algo más de tiempo para desarrollar algunas ideas, algunas de ellas expuestas en el punto de Líneas de desarrollo futuro, pero aún así estoy contento con el resultado final.

5. Líneas de desarrollo futuro

Un catálogo de productos no aporta un gran beneficio por sí solo a una empresa, pero sí que es una buena base para creación de una gestión empresarial mucho más completa. Las posibilidades son muy grandes y todo dependería del tipo de negocio (pequeña perfumería, cadena de perfumerías, mayorista, vendedor online, una fusión de ambos, etc.) en el que se quisiera poner en marcha esta solución.

En el caso de una pequeña perfumería sería importante crear un módulo de ventas al cliente final que funcionara en un TPV. Este módulo debería ser capaz de hacer tickets de venta y restar del stock la mercancía vendida.

Si la empresa en la que montar la aplicación es una cadena de perfumerías también se debería hacer un módulo de ventas similar al de la pequeña perfumería. El stock debería controlarse por tienda y también llevar el stock de un almacén central. Un punto importante sería crear un módulo para reponer la mercancía a las tiendas para evitar que se quedaran sin stock de algún producto. Deberían tenerse en cuenta temas como conectarse con la central, si se trabaja online o bien offline enviando posteriormente los datos de fin de día.

Un mayorista de perfumes necesita mantener fichas de clientes, así como la posibilidad de hacerles ofertas, gestionar pedidos, albaranes y hacer facturas. Posiblemente también sería muy útil una gestión de los pagos y sin duda algún método para enviar los datos de venta a la contabilidad para así evitar hacerlo a mano. Hoy en día es muy común trabajar vía EDI con lo que no sería muy útil preparar la aplicación para poder recibir pedidos vía EDI y enviar por el mismo método las facturas.

En el caso que se quiera vender online, el primer paso es la creación de una nueva web específica para los potenciales clientes. Se debería crear un módulo para el control de los clientes con el fin de facturarles y enviarles correctamente la mercancía.

En resumen, las posibilidades son muchas y variadas dependiendo lo que se vaya a hacer. La infraestructura creada con este proyecto debería ser una sólida base para estas posibles ampliaciones.

6. Glosario

.NET: es un proyecto de Microsoft para crear una nueva plataforma de desarrollo de software con énfasis en transparencia de redes, con independencia de plataforma y que permita un rápido desarrollo de aplicaciones. Basado en esta plataforma, Microsoft intenta desarrollar una estrategia horizontal que integre todos sus productos, desde el Sistema Operativo hasta las herramientas de mercado.

AJAX: acrónimo de Asynchronous JavaScript And XML (JavaScript y XML asíncronos), es una técnica de desarrollo web para crear aplicaciones interactivas. Éstas se ejecutan en el cliente, es decir, en el navegador del usuario, y mantiene comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre la misma página sin necesidad de recargarla. Esto significa aumentar la interactividad, velocidad y usabilidad en la misma.

ASP.NET: es un conjunto de tecnologías de desarrollo de aplicaciones web comercializado por Microsoft. Es usado por programadores para construir sitios web domésticos, aplicaciones web y servicios XML. Forma parte de la plataforma .NET de Microsoft y es la tecnología sucesora de la tecnología Active Server Pages (ASP).

Assembly: en la plataforma de Microsoft .NET un Assembly es una librería de código parcialmente compilada usada en la instalación de programas.

C#: es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET, que después fue aprobado como un estándar por la ECMA e ISO. Su sintaxis básica deriva de C/C++ y utiliza el modelo de objetos de la plataforma .NET el cual es similar al de Java aunque incluye mejoras derivadas de otros lenguajes (más notablemente de Delphi y Java).

C++: es un lenguaje de programación, diseñado a mediados de los años 1980 como extensión del lenguaje de programación C. Abarca tres paradigmas de la programación: la programación estructurada, la programación genérica y la programación orientada a objetos

Delphi: es un entorno de desarrollo de software diseñado para la programación de propósito general con énfasis en la programación visual. En Delphi se utiliza como lenguaje de programación una versión moderna de Pascal llamada Object Pascal.

Domain Model: es el modelo conceptual de un sistema el cual describe las entidades involucradas en ese sistema y la relación entre ellas.

EDI: del inglés, Electronic Data Interchange, intercambio electrónico de datos, es un software Middleware que permite la conexión a distintos sistemas empresariales.

Framework: es una estructura de soporte definida en la cual un proyecto de software puede ser organizado y desarrollado.

HTTP: hace referencia a protocolo de transferencia de hipertexto y es el protocolo usado en cada transacción de la Web.

IIS: hace referencia a Internet Information Server y es el programa estándar de Windows que permite publicar páginas web en este entorno.

IOC: acrónimo ingles de Inversion of Control, la inversión del control explica la manera con la cual un objeto obtiene las referencias a sus dependencias

Java: Es un lenguaje orientado a objetos diseñado a principios de los 90 y cuya sintaxis proviene en gran medida del C++ pero con un modelo de objeto más sencillo y eliminando herramientas de bajo nivel como los punteros.

JavaScript: Es un lenguaje interpretado utilizado principalmente en el desarrollo de páginas Web. Su sintaxis es similar a la del Java y el C.

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.

Nhibernate: es una herramienta de Mapeo objeto-relacional para la plataforma.Net que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML) que permiten establecer estas relaciones.

ORM: del inglés, Object-Relational mapping, mapeo Objeto-Relacional es una técnica que permite guardar objetos en una base de datos relacional.

PDF: (del inglés Portable Document Format, Formato de Documento Portátil) es un formato de almacenamiento de documentos especialmente ideado para documentos susceptibles de ser impresos, ya que especifica toda la información necesaria para la presentación final del documento, determinando todos los detalles de cómo va a quedar.

POCO: (del inglés Plain Old CLR Objects) se utiliza para describir aquellos objetos de algún lenguaje de .NET que no son de ningún tipo especial ni cumplen ningún rol establecido por algún interface externo al proyecto en el que se usan.

Proxy: En programación se refiere a un interface a otro objeto.

Spring.NET: Librería de código libre que facilita la utilización del IOC en los programas entre muchas otras cosas.

TPV: es un acrónimo para Terminal Punto Venta y hace referencia a los programas y tecnologías que ayudan en las tareas de gestión de un negocio de venta al público.

7. Bibliografía

Erich Gamma, Richard Help, Ralph Johnon, John Vlissides. 1995. *Design Patterns.* s.l. : Addison Wesley, 1995.

Esposito, Dino. 2005. *Programming Microsoft ASP.NET 2.0. Core Reference.* s.l. : Mocrosoft Press, 2005.
—. 2006. *Programming Microsoft ASP:NET 2.0 Applications. Advanced Topics.* s.l. : Microsoft Press, 2006.

Evans, Eric. 2005. *Domain-Driven Design.* s.l. : Addison Wesley, 2005.

Fowler, Martin. 2004. http://www.martinfowler.com/articles/injection.html.
—. 2003. *Patterns of enterprise application Architecture.* s.l. : Addison Wesley, 2003.
__. **2005.** UML Distilled Third Edition. Addison-Wesley.

Larman, Craig. 2005. *Applying UML and Patterns.* s.l. : Prentice Hall, 2005.

Rockford Lhotka. 2004. Expert C# Business Objects. Apress 2004.

Microsoft. Ajax Control Toolkit. [En línia]
http://www.codeplex.com/Release/ProjectReleases.aspx?ProjectName=AtlasControlToolkit.
—. ASP.NET AJAX. [En línia] http://ajax.asp.net/.
—. Página oficial de Microsoft .NET. http://msdn2.microsoft.com/en-us/netframework/default.aspx.

Nikhil Kothari, Vandana Datye. 2002. *ASP.NET Server Controls and Components.* s.l. : Microsoft Press, 2002.

Nilsson, Jimmy. 2006. *Applying Domain-Driven Design and Patterns.* s.l. : Addison Wesley, 2006.

Troelsen, Andrew. 2001. *C# and the .NET Platform.* s.l. : Apress, 2001.

ZedGraph. ZedGraph. http://zedgraph.org/wiki/index.php?title=Main_Page.

Hibernate/NHibernate http://www.hibernate.org/.

Gios, Paolo. Gios.NET: http://www.paologios.com/default.aspx?page=pdf.

Spring.NET. http://www.springframework.net/.

Wikipedia: http://es.wikipedia.org/wiki/