Títol:Portafolio de finanzas implementado en Joomla!

Volum:1

Alumne:Antoni Aguiló Tarré

Director/Ponent:Argimiro Arratia/Ricard Gavaldà

Departament: Llenguatges i Sistemes Informàtics.

Data: 01/07/2010

DADES DEL PROJECTE

Títol del Projecte: Portafolio de finanzas implementado en joomla!

Nom de l'estudiant: Antoni aguiló Tarré
Titulació: Enginyeria Informàtica
Crèdits: 37,5
Director/Ponent: Argimiro A. Arratia Quesada
Departament: LSI

MEMBRES DEL TRIBUNAL *(nom i signatura)*

President: Lluís Vila Grabulosa

Vocal:Monica M.Becue Bertaut

Secretari:Ricard Gavaldà Mestre

QUALIFICACIÓ

Qualificació numèrica:
Qualificació descriptiva:

Data:

Facultat d'Informàtica de Barcelona
Universitat Politècnica de Catalunya

PROYECTO FINAL DE CARRERA

Portafolio de finanzas implementado en Joomla!

Alumno:
Antoni Aguiló Tarré

Director:
Argimiro A.Arratia Quesada

GUÍA DE LECTURA

Este documento, memoria de un proyecto final de carrera, se estructura en distintas partes descritas a continuación:

La primera parte, de introducción, donde se trata la motivación del proyecto, el entorno de desarrollo y la planificación inicial y final. El objetivo es dar una breve introducción al ámbito del proyecto para que el lector entienda la especificación de éste.

Las dos siguientes secciones definen el contexto de la aplicación. En la primera se describen varios conceptos financieros, y en concreto, el IBEX 35. En la segunda, se introducen los CMS, particularmente Joomla!, que es la aplicación usada en la implementación. De esta forma el lector tendrá unos conocimientos básicos para poder entender los términos que se usan en las siguientes explicaciones.

El cuerpo del proyecto se encuentra en el contenido de las siguientes secciones: Análisis del proyecto, diseño e implementación. La primera, da una especificación más detallada de toda la funcionalidad del proyecto. Se explica a quién va dirigido la aplicación y cuáles son todos los casos de uso que engloban todo el sistema. En general, responde a QUÉ hace el sistema construido. El diseño, explica CÓMO es el sistema construido. En nuestro caso, define cómo son las tablas usadas en la base de datos y define algunos diagramas de actividades del sistema. Finalmente, la descripción de la implementación ayuda al lector a entender cómo se implementa un componente en Joomla! desde el inicio hasta el final pasando por las fases de instalación, creación e implementación.

Las dos últimas secciones concluyen la lectura. Indican cuáles han sido las pruebas usadas para que el sistema funcione correctamente, y cuáles son las conclusiones, mi experiencia personal y las mejoras futuras que se pueden aplicar.

ÍNDICE DEL CONTENIDO

INDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS:

INDICE DE CÓDIGOS:

INDICE DE ECUACIONES:

Capítulo 1

Introducción

1.1 Orígenes y antecedentes

Desde finales del año 2008, estoy investigando lenguajes de programación que permitan la creación de páginas web. Empecé aprendiendo el lenguaje HTML y CSS para crear páginas estáticas. Una vez obtenida la base, a través de unos libros de PHP y JavaScript, diseñé páginas con mayor dinamismo.

Fue entonces cuando descubrí los sistemas gestores de contenidos CMS que facilitaban la implementación y sobretodo el mantenimiento de los sitios web. En realidad, son el futuro de los sistemas de información web.

De allí surge el deseo de hacer uso de los CMS para el desarrollo de una plataforma. En particular, elegí Joomla!, uno de los CMS de código abierto más potentes y mejor aceptados en el mercado.

Además, el director del proyecto, Argimiro Arratia, me ha introducido en el mundo de los mercados de valores. Tras entrar en contacto con él buscando dirección del proyecto, me expresó su interés por las finanzas. Como punto de encuentro de nuestros respectivos intereses, se planteó desarrollar una plataforma electrónica de análisis y gestión de los mercados bursátiles. Se pretende crear un servicio de seguimiento de carteras bursátiles con posibilidades de desarrollo en una plataforma comercial. Concretamente, en el proyecto se quiere mostrar la información y gestionar las variaciones que se producen en los componentes del IBEX 35. Para ello, se permitiría a un usuario rellenar un formulario de compra y venta de acciones, para que el sistema le muestre información financiera de sus títulos: la ganancia, la variancia, el stop-loss y otras estadísticas.

Este ha sido un muy buen ejercicio de aprendizaje de herramientas para realizar desarrollos comerciales.

1.2 Objetivos del proyecto

El objetivo principal del proyecto, es crear un portafolio de finanzas para que pueda ser utilizado en cualquier página web gestionada con Joomla!. Concretamente, se implementará un componente que permitirá al usuario ver su historial de compras y ventas de acciones junto con información actual de los mercados financieros. Para llevar a cabo esto, se aprovecha la gran cantidad de información publicada en la página oficial de finanzas de Yahoo y otras páginas web públicas de información financiera.

Este proyecto puede ser el inicio de una aplicación que en un futuro, si se ampliara, realizaría más estudios del mercado, para tener un mejor acierto del comportamiento de éste. Así podría aconsejar al usuario cuándo sería un buen momento de realizar una compra o una venta para obtener un mayor beneficio en las ganancias. Además, se podría ampliar el portafolio para que abarcara un número mayor de mercados.

Actualmente, ya existen herramientas que permiten la gestión de los mercados bursátiles. Éstas son privadas y de gran coste. El proyecto, al implementarse en Joomla! sigue las ideologías de código abierto y podrá ser usado en cualquier sitio Joomla!. A este proyecto se le da mucha importancia a la creación de un pequeño manual para futuros implementadores de Joomla! En la red hay varias explicaciones del código Joomla! y de cómo se debe implementar, pero o bien no están en castellano, o les faltan detalles importantes.

Otro gran objetivo del proyecto es aprender a realizar un proyecto software de tamaño medio aplicando todas sus fases: análisis, diseño, implementación, pruebas y cálculo de costes. En el mundo laboral informático se generan a diario muchos proyectos de este estilo que fracasan por su mala planificación. Realizar un proyecto software supone primero, crear una buena planificación temporal, y después, aprender los roles que participan: dirigir el proyecto, hacer el análisis, crear un diseño e implementarlo.

Un pequeño objetivo en este proyecto es aprender a realizar una redacción de un documento técnico grande con todo lo que conlleva. Se planteó que la memoria debe servir de guía para un usuario con conocimientos de Joomla! dando mucha importancia a la sección de implementación. Esta sección tiene que servir de manual para futuros implementadores de componentes Joomla!.

1.3 Requisitos de la aplicación

El portafolio de finanzas está compuesto por:

Gestión usuarios:

El componente Joomla! permitirá la gestión de usuarios en la aplicación web. Para ello, se crea una tabla en la base de datos que contenga la información privada del usuario. El usuario, podrá darse de alta a la aplicación gracias a un formulario de registro. Además, al identificarse, tendrá un portal personalizado, donde se indicará su situación en el mercado financiero.

Información financiera actualizada:

Habrá una base de datos que contendrá toda la información histórica de los valores de los componentes del Ibex 35. El usuario podrá consultar dicha información para saber cómo va variando el mercado. Además, las transacciones diarias se van actualizando al mismo momento que la página de Yahoo finanzas.

Gracias a ello, la aplicación tiene toda la información del mercado actualizada.

Formularios de compras y ventas de acciones

Un usuario identificado podrá rellenar unos formularios de compra y venta de acciones, para que el sistema los almacene en la base de datos. A partir de estos datos, el sistema proporciona información de la ganancia, la pérdida, el stop-loss y la volatilidad de la cartera del usuario.

Noticias financieras

Se añadirán a la aplicación unos RSS de noticias de las páginas más importantes del mercado español. Noticias de páginas como el Economista o YahooFinance serán mostradas en un apartado de la aplicación.

Secciones de información y perfil de las compañías

Habrá una sección de perfiles de compañías. Esto permitirá al usuario conocer más la compañía del IBEX 35. A su vez se mostrarán datos como el volumen de la empresa en el mercado, el número de acciones, sus valores actuales, etc.

Sección de administración

Al administrador que use la aplicación se le permitirá añadir, eliminar y modificar los componentes del Ibex 35. Cuando un componente del Ibex 35 cambie valores como su volumen o su número de acciones, el administrador podrá modificarlos en su zona privada.

1.4 Metodología de trabajo y herramientas usadas

Partiendo de la base que el presente proyecto está catalogado como de ingeniería del software y de base de datos, se ha seguido la metodología de trabajo UML (*Unified Modelling Language*, en español Lenguaje Unificado de Modelado). A lo largo de la lectura se mostrará el diagrama de clases y relacional, los casos de uso y algún diagrama de actividades.

Siguiendo la metodología UML el proyecto se ha separado en cinco fases:

- Primero, a través de entrevistas con el director, se ha definido el objetivo principal del proyecto junto con el coste y la planificación inicial.
- A continuación, se ha concretado más la funcionalidad del proyecto con la especificación de los casos de usos principales y los que se pueden implementar en el futuro. Los casos de uso, explican la interacción que tiene el usuario con el sistema viendo cual es el flujo de comunicación principal y alternativos, si los hay.
- El siguiente paso, es generar un diseño del proyecto indicando cómo será la base de datos y algunos de los diagramas de actividades principales.
- Una vez, definido todo el proyecto, se implementa el sistema partiendo de la base que se tienen que cumplir todos los requisitos. En este punto se produce una iteración con los puntos anteriores donde se van modificando o añadiendo casos de uso según las necesidades del sistema.
- Finalmente, se van haciendo distintos tipos de pruebas para que el proyecto sea robusto y no tenga fallos.

Ilustración 1-1: fases de un proyecto software

Las herramientas utilizadas para el desarrollo de todo el proyecto se listan a continuación:

- **Microsoft Office 2007:** Es el conjunto esencial de software que permite crear de un modo rápido y sencillo hojas de cálculo, documentos y presentaciones. Con esta herramienta se ha elaborado toda la documentación del proyecto, con Microsoft Office Word 2007, la presentación, con Microsoft Office PowerPoint 2007 y la planificación temporal con Microsoft Project 2010.
- **WAMP 2.0:** Sistema que incluye las aplicaciones libres para el desarrollo de un sistema web: el software para el servidor web*, Apache 2.2.11,* el gestor de la base de datos, *MySQL 5.1.36* y el software de programación script web *PHP 5.3.0.* Con esta herramienta se ha podido implementar el sistema.
- **Macromedia Dreamweaver 8:** Es un editor de HTML visual, diseñado para desarrolladores web. Es compatible con las tecnologías web más usadas tales como JavaScript, CSS; AJAX, XHTML, subversión (SVN) y una infinidad más de tecnologías. Gracias a este programa, se pueden diseñar páginas con el editor visual WYSIWYG (*What You See Is What You Get*). Con este editor se han creado todos los archivos PHP y JavaScript.
- **Visual Paradigm for UML 7.2 Enterprise Edition:** Es una herramienta UML profesional que permite generar el ciclo de vida de un proyecto de ingeniería del software: Especificación, análisis, diseño, construcción y pruebas. El software modelado ayuda a construir proyectos software de mejor calidad y con un menor coste. Se incluyen numerosos tutoriales de UML y ejemplos de proyectos. Gracias a este programa se han elaborado los diagramas de caso de uso, de actividades, el modelo de dominio y las clases de diseño.
- **PhpMyAdmin:** Programa que se incluye dentro de la aplicación WAMP que permite la gestión de la base datos mediante una interfaz simple y clara.

1.5 Planificación del proyecto

La planificación de un proyecto no es algo inalterable en el futuro. Es algo que hay que ir revisando y modificando a medida que un proyecto avanza. Hay que tener en cuenta que la ejecución final de un proyecto puede cambiar, hasta el punto de modificar la fecha final de entrega, por lo que afecta la planificación de todo el proyecto. Se puede decir, que la planificación inicial del proyecto, es una previsión del coste temporal de cada tarea. Lo ideal, es que la estructura inicial planificada se vea lo menos alterada posible. Esto se logra con la experiencia como administrador de proyectos.

En este caso, al tratarse de un proyecto de tamaño medio, hacer una previsión temporal puede resultar poco complicado.

El proyecto se planteó a mitades de enero del 2010. La idea inicial era invertir unas 700 horas en el proyecto estructuradas de la siguiente manera:

- Las primeras 200 horas dedicadas al estudio del entorno de programación Joomla! y a la lectura de documentación sobre los mercados financieros. La búsqueda de información es una actividad continua durante todo el proyecto. Se han mencionado las primeras cien horas para dar importancia a todo lo que ha permitido tener una base de conocimientos.
- Las siguientes 150 horas usadas para a la especificación y el análisis del proyecto.
- La parte más importante del proyecto, la implementación, tenía una duración prevista de unas 250 horas.
- Finalmente, las pruebas y la documentación de la memoria ocuparían unas 150 horas. Destacar que las pruebas unitarias de cada funcionalidad del sistema se incluyen en la etapa de implementación y que solamente se tienen en cuenta las pruebas de integración.

En la sección de costes del proyecto se detalla cada una de las tareas y las horas dedicadas a ello.

La siguiente tabla, muestra un diagrama de Gantt simple con la división de las tareas ejecutadas en el proyecto inicialmente. Nótese que se ha seguido una secuencia de especificación, análisis, diseño e implementación partiendo de la base que se trata de un proyecto de ingeniería del software:

	ene-10		feb-10		mar-10		abr-10		may-10		jun-10	
	1ª quin	2ª quin	1ª quin	2ª quin	1ª quin	2ª quin	1ª quin	2ª quin	1ª quin	2ª quin	1ª quin	2ª quin
Definición del proyecto												
Búsqueda de la información del problema												
Especificación del problema y metodología												
Análisis del proyecto												
Implementación y testeo												
Búsqueda de la información												
Instalación y aprendizaje de Joomla!												
Implementación del componente												
Testeo de partes en servidor local												
Testeo de partes en servidor externo												
Testeo de todas las partes												
Redacción de la memoria del proyecto												
Presentación del proyecto												

Tabla 1-1: Diagrama de Gantt inicial

Han aparecido muchos diagramas temporales intermedios a causa de imprevistos y problemas encontrados durante el transcurso del proyecto.

Uno de los cambios más importantes en la planificación inicial es que se ha invertido aproximadamente 250 horas en la investigación, cuando inicialmente se calculaban 200 horas.

Además, en la implementación, se dedicó unas 100 horas más. Esto es debido a que hay muy poca información del código en Internet y en libros. Los libros *Learning Joomla 1.5 Extension Development* [L07] y *Joomla! 1.5: A User's Guide: Building a Successful Joomla!* [N08] me permitieron encontrar los puntos claves para implementar un componente siguiendo el método MVC (Model, View, Controller) [Ver sección 5.3.1-El patrón MVC].

Esto, supuso tener que comprimir el testeo de la aplicación y posponer la fecha de la defensa del proyecto a 1 de Julio.

La ilustración 1-1, muestra el diagrama de Gantt final que engloba las 6 partes del proyecto: estudio, ámbito, análisis, diseño, implementación, testeo y documentación.

Nótese que el proyecto al final ha durado 790 horas. Se ha tenido que aumentar el tiempo de trabajo a causa de una planificación inicial errónea.

Como se comprueba en la imagen varias tareas del proyecto están solapadas. En la segunda parte de la implementación, se empezaron a hacer pruebas al mismo tiempo. Esto es debido a que cuando se implementaba un caso de uso, automáticamente se comprobaba que funcionase correctamente.

A partir de finales de marzo, una vez definido el ámbito y la especificación del proyecto, se empezó a documentar la memoria de éste.

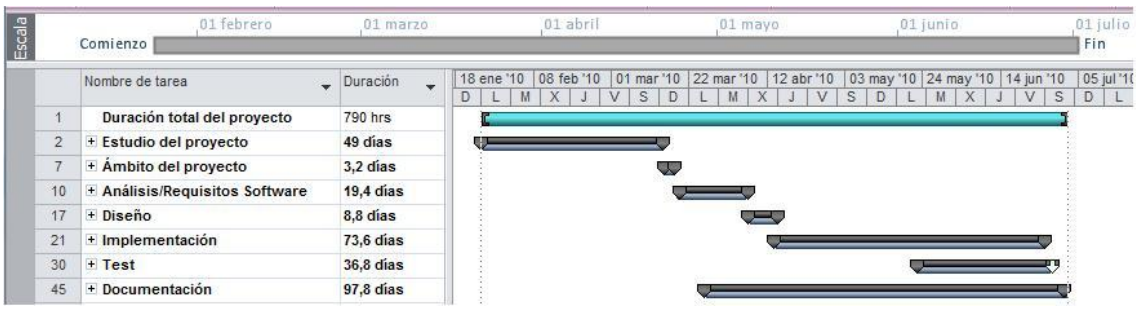


Ilustración 1-2: Diagrama Gantt - Tareas

La ilustración 1-3 muestra detalladamente cómo se han invertido las 250 horas de estudio inicial para llegar finalmente al ámbito del proyecto. En este punto ya se saben cuáles son los objetivos que se pueden abarcar en el proyecto y se tiene una buena formación sobre Joomla! y su entorno de programación. Además, se han investigado páginas web de mercado de valores para tener conocimientos básicos financieros.

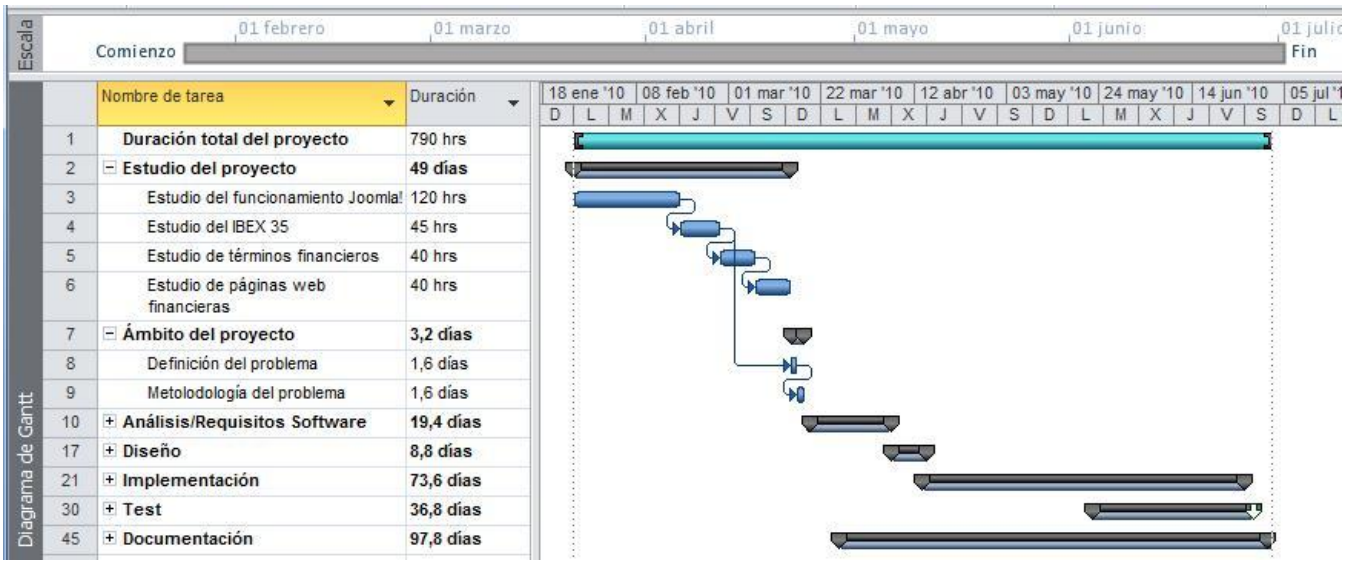


Ilustración 1-3: Diagrama Gantt - Estudio inicial

La ilustración 1-4 despliega el análisis y el diseño para indicar cuáles han sido las subtareas asignadas. Cualquier proyecto de ingeniería de software requiere que inicialmente se haga un estudio preliminar sobre las necesidades y los objetivos del proyecto. El analista debe reunirse constantemente con el cliente, en mi caso, el director del proyecto final de carrera, para detallar todos los requisitos del sistema a partir del documento preliminar. Una vez, llegado a un acuerdo, se redefinen los casos de uso para finalizar con el análisis. El diseño se basa en el resultado del análisis realizado. A partir del diagrama conceptual del proyecto, se genera un modelo relacional para la base de datos y los diseños de los casos de uso. En el diseño de este proyecto se ha elegido un caso de uso para mostrar su Vista y diagrama de actividades.

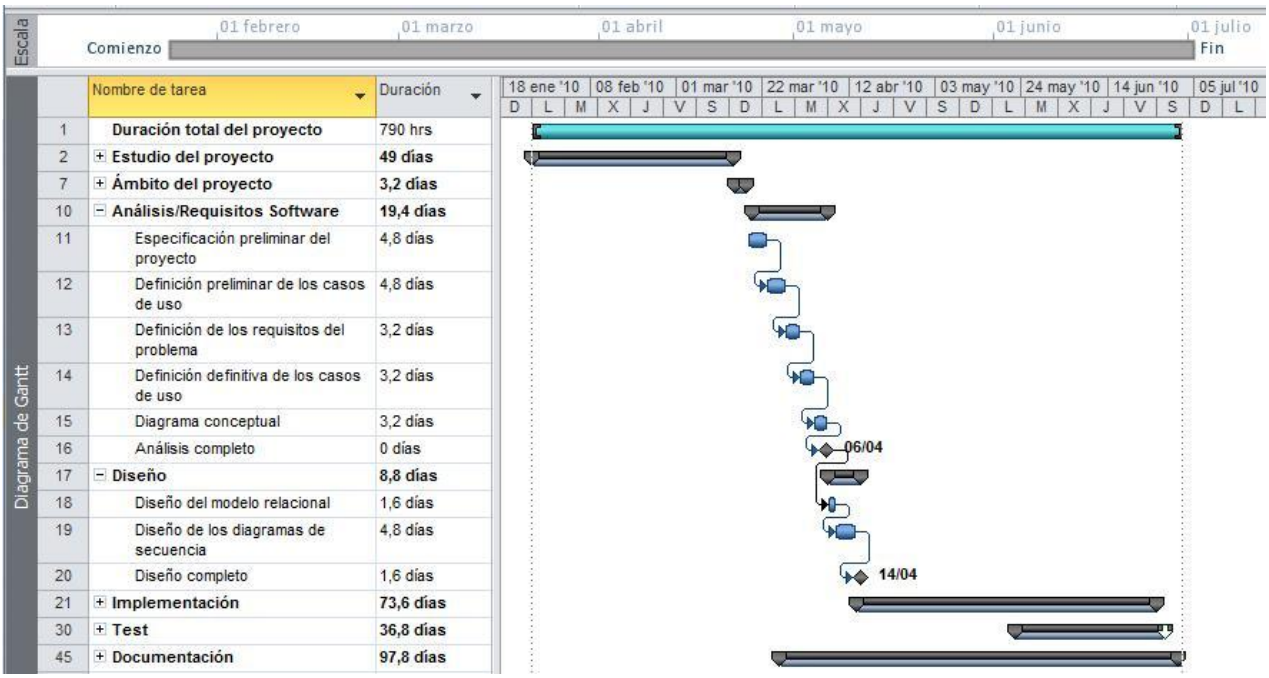


Ilustración 1-4: Diagrama Gantt - Análisis

La implementación está compuesta por tres partes bien diferenciadas. La primera, consiste en un estudio del código Joomla! para saber cuáles son los patrones de programación y qué tipo de diseño utilizan. Esto se consiguió gracias al libro *Learning Joomla 1.5 Extension Development* [L07] y a la página web oficial de Joomla! [JOOMLA] y supuso unas 40-50 horas de estudio. En la segunda parte, se instala Joomla! junto a algunos módulos ya implementados, se crea la estructura del componente y se introducen datos en la BD. El coste temporal total fue de unas 40 horas. Y finalmente, la tercera parte, la más importante del proyecto, consiste en la implementación de todos los casos de uso. Se implementan todas las funcionalidades y el paquete instalador del componente con un coste de 240 horas.

La ilustración 1-5 recoge toda la información detallada:

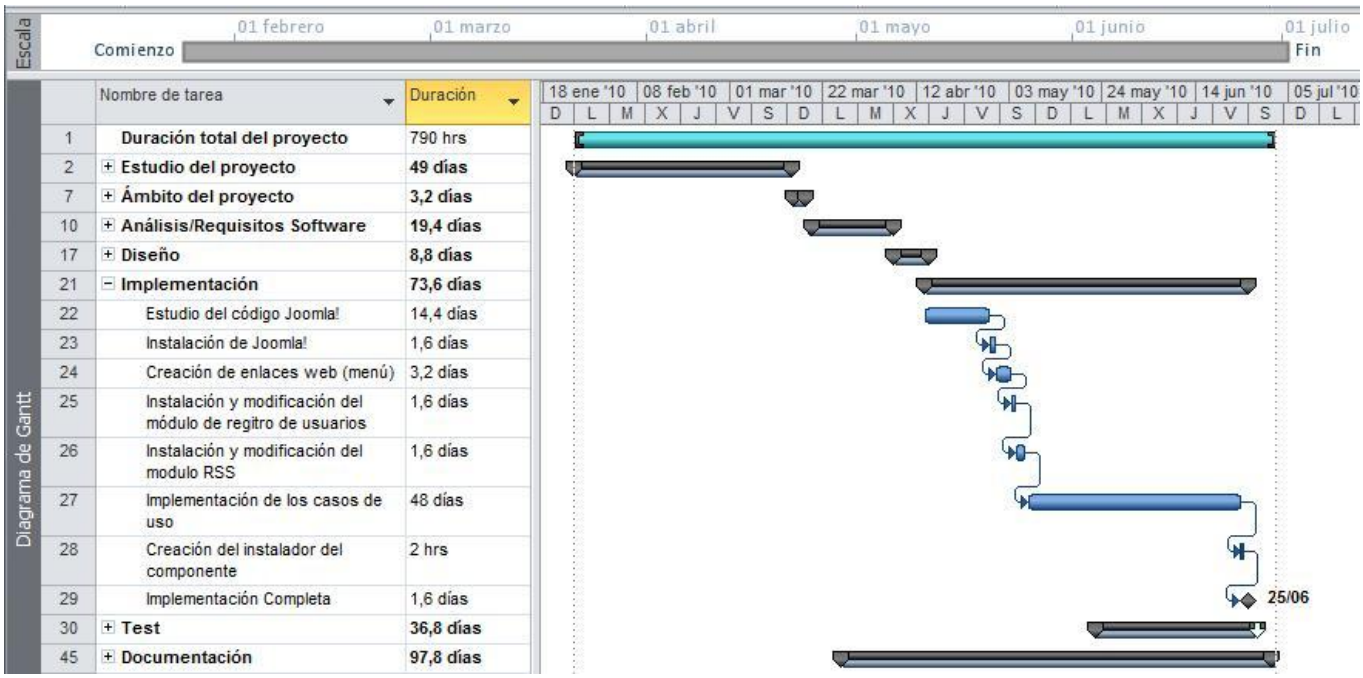


Ilustración 1-5: Diagrama de Gantt - Implementación

La ilustración 1 -6 indica cómo se ha extendido el test y la documentación. Estas dos tareas se solapan con la implementación por varias razones: Se consideró oportuno ir documentando la sección de implementación en la memoria e ir haciendo pruebas a medida que se iban añadiendo nuevas funcionalidades en el sistema. Por eso, las horas dedicadas a ello son más difíciles de ajustar. Un cálculo dedicado a ello ha sido aproximadamente de 150 horas para las pruebas de integración.

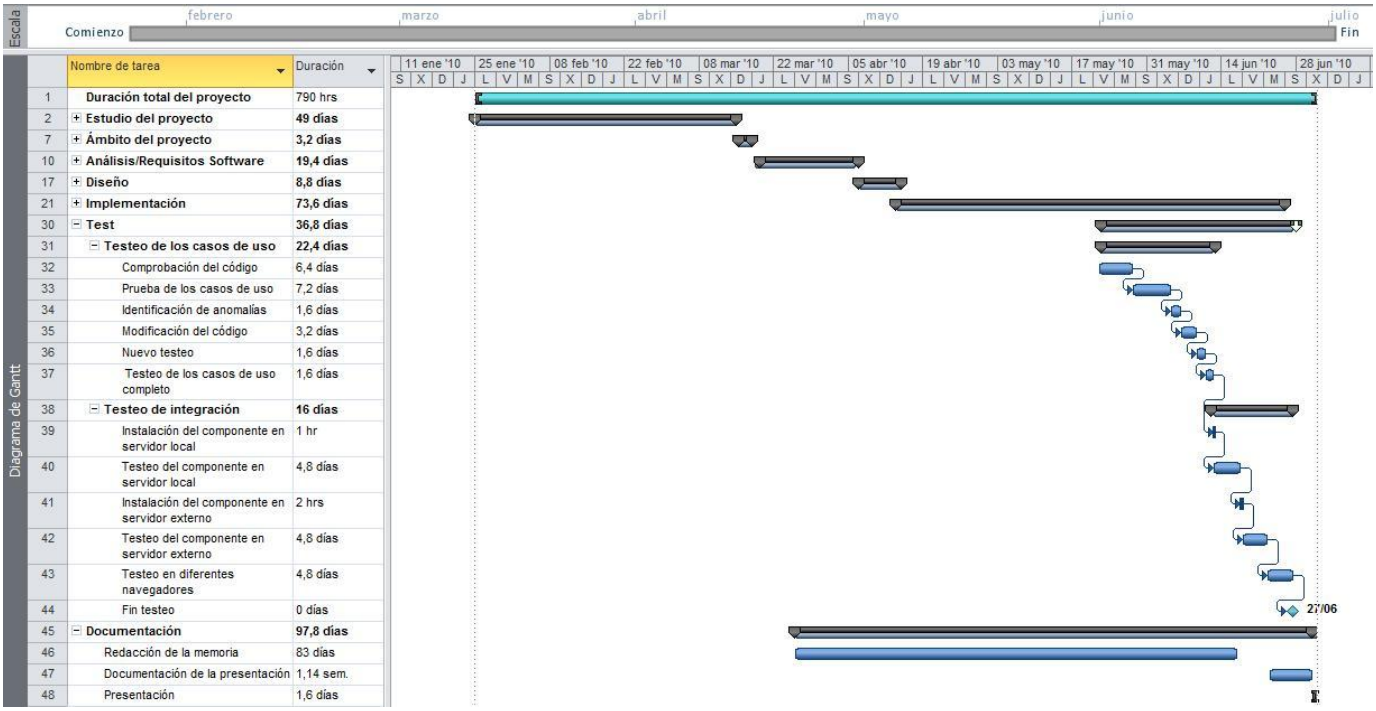


Ilustración 1-6: Diagrama de Gantt - Test y Documentación

1.6 Coste del proyecto

Como sucede en la planificación temporal, es muy importante realizar un presupuesto inicial del proyecto para ver el grosor del proyecto a nivel económico. Esto permite hacer una predicción económica del proyecto y saber si se dispone de recursos suficientes para su realización. A pesar de todo, el coste real del proyecto se calcula al finalizar éste ya que puede haber alteraciones en su cálculo inicial.

Hay dos grandes grupos dentro de los costes temporales: El coste del personal encargado en llevar a cabo todas las fases del proyecto, y el coste del material usado para ello.

Cuando se calcula el coste total de un proyecto aparecen también factores de desplazamiento, imprevistos, sistema eléctrico y otros. Éstos aumentan la complejidad del cálculo del coste final, por lo que no se han considerado.

1.6.1 Coste del personal

Este bloque es el que más recursos económicos consume. En este proyecto, al ser un proyecto final de carrera, no hay un personal especializado en cada una de las tareas, sino que solamente hay una persona, el proyectista que se encarga de toda la realización de éste.

Para mostrar el cálculo que supondría la realización de este proyecto se ha supuesto lo siguiente:

Habrán distintos roles/recursos encargados de realizar las tareas que se han especificado en el diagrama de Gantt. En un proyecto informático intervienen los roles de director de proyecto, analista, diseñador y programador. En este caso, el director del proyecto será a su vez el analista y se encargará de toda la parte de especificación y análisis del problema. El diseñador se encarga de realizar el diseño del sistema y el programador de codificarlo. En la siguiente tabla se muestra el precio por hora de cada rol:

Recurso/Rol	*Coste*
Analista o director del proyecto	30 €/hora
Diseñador	25 €/hora
Programador	15 €/hora

Tabla 1-2: Precios €/hora de los trabajadores

A partir del diagrama de Gantt final y del coste de cada uno de los recursos/roles del proyecto se puede realizar una estimación del coste del personal del proyecto. Esto se muestra en la siguiente tabla:

Nombre de la tarea	Recurso/Rol	Duración en horas	Coste
Estudio del proyecto			
-Estudio del funcionamiento Joomla!	Analista Programador	120 horas	2.700,00 €
-Estudio del IBEX 35	Analista Diseñador	45 horas	1.237,50 €
-Estudio de términos financieros	Analista Diseñador	40 horas	1.100,00 €
-Estudio de páginas web financieras	Analista Programador	40 horas	900,00 €
Ámbito del proyecto			
-Definición del problema	Analista	8 horas	240,00 €
-Metodología del problema	Analista	8 horas	240,00 €
Análisis/Requisitos Software			
-Especificación preliminar del proyecto	Analista	20 horas	600,00 €
-Definición preliminar de los casos de uso	Analista	20 horas	600,00 €
-Definición de los requisitos del problema	Analista	12 horas	360,00 €
-Definición definitiva de los casos de uso	Analista	12 horas	360,00 €
-Diagrama conceptual	Analista	12 horas	360,00 €
Diseño			
-Diseño del modelo relacional	Diseñador	4 horas	100,00 €
-Diseño de los diagramas de actividades	Diseñador	16 horas	400,00 €
-Diseño completo	Diseñador	4 horas	100,00 €
Implementación			
-Estudio del código Joomla!	Programador	40 horas	600,00 €
-Instalación de Joomla!	Programador	4 horas	60,00 €
-Creación de enlaces web (menú)	Programador	8 horas	120,00 €
-Instalación y modificación del módulo de registro de usuarios	Programador	4 horas	60,00 €
-Instalación y modificación del modulo RSS	Programador	4 horas	60,00 €
-Implementación de los casos de uso	Programador	200 horas	3.000,00 €
-Creación del instalador del componente	Programador	2 horas	30,00 €
Test			
-Testeo de los casos de uso			
-Comprobación del código	Programador	8 horas	120,00 €
-Prueba de los casos de uso	Analista Diseñador	8 horas	

-Identificación de anomalías	Programador	8 horas	120,00 €
-Modificación del código	Programador	8 horas	120,00 €
-Nuevo testeo	Analista Diseñador	8 horas	
-Testeo de integración			
-Instalación del componente en servidor local	Programador	1 horas	15,00 €
-Testeo del componente en servidor local	Analista Diseñador	8 horas	
-Instalación en servidor externo	Programador	1 horas	15,00 €
-Testeo del componente en servidor externo	Analista Diseñador	8 horas	220,00 €
-Testeo en diferentes navegadores	Analista Diseñador	4 horas	110,00 €
Documentación			
-Redacción de la memoria	Analista Diseñador Programador	100 horas	2.333,33 €
-Documentación de la presentación	Analista	10 horas	300,00 €
-Presentación	Analista	1 horas	30,00 €
TOTAL		796 horas	16.610,8€

Tabla 1-3: Coste total en función del personal

Se observa que el desarrollo total del proyecto ha necesitado 796 horas con un coste total de 16.610,83 €.

1.6.2 Coste del material

En el coste material se cuenta el software y el hardware utilizado. Se ha trabajado con el sistema operativo Windows Vista utilizando programas que se incluían en el sistema o que se han conseguido de prueba: Se ha usado el editor Dreamweaver, los programas de Microsoft Office, y los navegadores Firefox y Explorer. Todo el código y servidores usados son de código libre, por lo que no conllevan un coste adicional.

Además se ha usado un portátil de coste 800 € incluyendo el mencionado sistema operativo. Añadiendo los programas Dreamweaver y Microsoft Office el coste total del material sube a unos 1000€.

1.6.3 Coste total

El coste total no es más que la suma de los dos costes anteriores:

Coste total = 16.610,83 € + 1000 € = 17.610,83 €

Se puede considerar un proyecto de un tamaño económico medio/bajo.

Capítulo 2

Contexto de aplicación

2.1 Introducción

En el presente capítulo se expondrán las características principales del Mercado Financiero Español, con énfasis en su índice principal el IBEX 35, y además de varios elementos de finanzas relevantes en este trabajo.

La exposición se fundamenta en la información recogida en diversos portales digitales como Yahoo Finanzas [YAHOOFINANZAS], El Economista [ELECONOMISTA] y documentos web para el cálculo de elementos financieros [B10] y [K09].

2.2 Qué es el IBEX?

El **IBEX 35** (*Iberia Index*) nació a finales de 1989 y es considerado el principal índice de referencia de la bolsa española elaborado por Bolsas y Mercados Españoles (BME).

Está compuesto por las 35 empresas españolas con más liquidez que cotizan en el Sistema Interconexión Bursátil Electrónico (SIBE) en la cuatro Bolsas Españolas (Madrid, Barcelona, Bilbao y Valencia). Se trata de un índice ponderado por capitalización bursátil; es decir, no todas las empresas tienen el mismo peso, al contrario que sucede con los índices de Dow Jones.

Todas las grandes compañías están interesadas en pertenecer al grupo de 35 empresas que integran el IBEX 35, debido a que estar en esta élite empresarial supone gozar de un gran prestigio a nivel nacional e internacional. De ahí, que también se lo conozca como grupo *selectivo*.

La composición del IBEX 35 es decidida por el Comité Asesor Técnico (CAT) de la Bolsa según una serie de criterios que permiten establecer el peso de las compañías en el mercado bursátil como pueden ser volumen de acciones, grado de liquidez, nivel de rotación de las acciones, montante de la capitalización y nivel de capital flotante. Dicho comité se reúne cada seis meses y siempre que sea necesario por los movimientos del mercado para revisar la composición del índice y decidir si hay que excluir algún miembro en beneficio de otras compañías. Para que un valor forme parte del IBEX 35 se requiere principalmente que:

- Su capitalización media sea superior al 0,30 por ciento a la del IBEX 35 al periodo analizado.
- Que haya sido contratado al menos en la tercera parte de las sesiones de ese periodo.

Una empresa también podría ser elegida para entrar si estuviese entre los 15 valores con mayor capitalización.

2.2.1 Historia

El IBEX 35 se inició el 14 de enero de 1992. Gracias a una estimación del índice que se hizo posteriormente, existen valores históricos desde 1989.

Históricamente, la mayor caída sufrida por el IBEX 35 se produjo el viernes 10 de octubre del 2008 cuando se perdió un 9,14% llegando a los 8.997,70 puntos. Además ha sido la mayor caída de la bolsa española desde el noviembre de 1987. Durante el mismo año 2008 el IBEX sufrió una depreciación progresiva de 39,4%.

En cambio, la mayor subida fue el lunes 10 de mayo de 2010 con un incremento de un 14,43%. Se pasó de 9.065,10 a 10.351,90 puntos. La subida se produjo gracias a la aprobación del plan de rescate europeo, justo después de la segunda peor semana del índice en su historia. El anterior mayor aumento fue el 13 de octubre de 2009, cuando EEUU y Europa anunciaron medidas de apoyo a la banca después de la crisis desatada por la quiebra de Lehman Brothers.

2.2.2 Cálculo

El cálculo del valor del índice se hace según la fórmula [B10]:

$$IBEX\,35(t) = IBEX\,35(t-1) \times \frac{\sum_{i=1}^{35} Cap_i(t)}{[\sum_{i=1}^{35} Cap_i(t-1) \pm J]}$$

Ecuación 2-1: Cálculo del valor del índice IBEX 35

Donde *I(t)* es el valor del índice en el momento t, *Cap* es la capitalización bursátil del *free float*[1] de las compañías que integran el índice y *J* es un coeficiente usado para ajustar el índice para que no se vea afectado por ampliaciones de capital.

[1] Término inglés que significa Capital Flotante. Se trata de la parte del Capital Social de una empresa o sociedad que cotiza libremente en bolsa y que no está controlado por accionistas de forma estable.

2.2.3 Composición del IBEX 35

A lo largo de su existencia, el IBEX 35 ha ido cambiando de componentes.

2.2.3.1 Componentes actuales

Actualmente el IBEX 35 está compuesto por:

Ticker	Empresa	ISIN		Ticker	Empresa	ISIN
ABE	Abertis	ES0111845014		GRF	Grifols	ES0171996012
ABG	Abegoa	ES0105200416		IBE	Iberdrola	ES0144580Y14
ACS	Grupo ACS	ES0167050915		IBLA	Iberia	ES0147200036
ACX	Acerinox	ES0132105018		IBR	Iberdrola Renovables	ES0147645016
ANA	Acciona	ES0125220311		IDR	Indra	ES0118594417
MTS	ArcelorMittal	LU0323134006		ITX	Inditex	ES0148396015
BBVA	Banco Bilbao Vizcaya Argentaria	ES0113211835		MAP	Corporación Mapfre	ES0124244E34
BKT	Bankinter	ES0113679I37		OHL	Obrasción Huarte Lain	ES0142090317
BME	Bolsas y Mercados Españoles	ES0115056139		POP	Banco Popular	ES0113790531
BTO	Banest	ES0113440038		REE	Red Eléctrica España	ES0173093115
CRI	Criteria CaixaCorp	ES0140609019		REP	Repsol	ES0173516115
ELE	Endesa	ES0130670112		SAB	Banco Sabadell	ES0113860A34
ENG	Enagás	ES0130960018		SAN	Banco Santander Central Hispano	ES0113900J37
EVA	Ebro Puleva	ES0112501012		SYV	Sacyr Vallehermoso	ES0182870214
FCC	Fomento de Contrucciones y Contratas	ES0122060314		TEF	Movistar	ES0178430E18
FER	Grupo Ferrovial	ES0162601019		TRE	Técnicas Reunidas	ES0178165017
GAM	Gamesa	ES0143416115		TL5	Telecinco	ES0152503035
GAS	Gas Natural	ES0116870314				

Tabla 2-1: Compañías que forman el IBEX 35

2.2.3.2 Componentes históricos

Los componentes que han sido parte del IBEX 35 en el pasado son los siguientes:

- **Aceralia**: ACE. Dejó de cotizar en 2004.
- **Agromán**: AGR. Salió en 1993 y se fusionó con el Grupo Ferrovial en 1999.
- **Aguas de Barcelona**: AGS. Salió el 4 de febrero de 2008 tras la OPA de La Caixa y SUEZ.
- **Altadis**: ALT. Salió el 4 de febrero de 2008 tras fructificar la OPA de Imperial Tobacco al 100% cotizado.
- **Antena 3 TV**: A3TV.salió el 2 de enero de 2008.
- **Arcelor**: LOR
- **Corporación Financiera Alba**: ALB.
- **Amper**: AMP.
- **Amadeus**: AMS.
- **Energía e Industrias Aragonesas**: ARA. Dejó de cotizar en 2003.
- **Asland**: ASL. Dejó de cotizar en Bolsa.
- **Autopistas del Mare Nostrum**: AUM. En 2000 se transformó en Aurea, que se fusionó en 2003 con ACESA para dar nacimiento a Abertis.
- **Asturiana de Zinc**: AZC. Dejó de cotizar en 2001.
- **Carrefour**: CAR
- **Banco Central**: CEN.se fusionó con Hispanoamericano y luego con Santander.
- **CEPSA**: CEP.
- **Cintra**: Absorbida por Ferrovial.
- **Cristalería Española**: CRI. Dejó de cotizar en Bolsa.
- **Continente**: CTE. Se fusionó con Pryca en 2000 para crear Carrefour.
- **Cortefiel**: CTF
- **Dragados**: DRC. Dejó de cotizar en 2003. Forma parte del Grupo ACS.
- **Ebro Industrias Agrícolas**: EBA. En 2001 se fusionó en Ebro Puleva
- **Ercros**: ECR.
- **Empresa Nacional de Celulosas de España**: ENCE.
- **Fadesa**: FAD
- **FECSA**: FEC. Dejó de cotizar en 1999. Forma parte del Grupo Endesa.
- **GESA (Gas y Electricidad)**: GES. Forma parte del Grupo Endesa.
- **Hidrocantábrico**: CAN. Dejó de cotizar en 2002.
- **Banco Hispano Americano**: HIS. Integrado en el Banco Santander Central Hispano.
- **Huarte:** HHU. Integrado en el Grupo Obrascón Huarte Lain.
- **Movistar Móviles (anteriormente Telefónica Móviles)**: TEM.
- **NH Hoteles**: NHH.
- **Picking Pack.**
- **Pryca**: PRY. En 2000 se fusionó con Continente dando lugar a Carrefour.
- **Puleva**: PUL. Llamada antes de 1992 UNIASA - UNI). En 2001 se fusionó en Ebro Puleva.
- **Radiotrónica**: RAD. Convertido en 1999 en Avanzit – AVZ.
- **Sarrió**: SAR. Dejó de cotizar en Bolsa.
- **Sevillana de Electricidad**: SEV. Dejó de cotizar en Bolsa en 1999. Forma parte del Grupo Endesa.
- **Sol Meliá**: SOL.
- **Telepizza**: TPZ.

- **Telefónica Móviles**: TEM. Recomprada por Telefónica
- **Telefónica Publicidad e Información**: TPI. Vendida al grupo británico Yell y excluida de la bolsa.
- **Terra Networks**: TRR. Dejó de cotizar en 2005. Forma parte del Grupo Telefónica.
- **Tubacex**: TUB.
- **Uralita**: URA.
- **Urbis**: URB.
- **Portland Valderrivas**: VDR.
- **Prisa**.
- **Unión Fenosa.** Salió del Ibex el 15 de abril de 2009 al finalizar la OPA de Gas Natural.
- **Viscofán**: VIS.
- **Vocento**.
- **Yell Publicidad (anteriormente Telefónica Publicidad e Información)**: TPI.
- **Zeltia.**
- **Zardoya Otis**: ZOT.

2.2.4 Índices sectoriales

2.2.4.1 Actuales

Además del índice IBEX 35, en España existen otros indicadores bursátiles que van ganando peso como referencia financiera [B10].

Éste es el ejemplo del índice **IBEX Medium Cap** (anteriormente conocido como IBEX Complementario) y del índice **IBEX Small Cap** creados el 1 de julio de 2005. Sirven de referente para observar la evolución de las empresas con mediana o pequeña capitalización bursátil de las cuatro Bolsas Españolas que cotizan en el Sistema de Interconexión Bursátil.

Estos índices ofrecen una información añadida sobre la evolución y el estado real de los mercados españoles. Pertenecer a estos índices supone cumplir una serie de requisitos tales como no formar parte del IBEX 35, tener un porcentaje de capital flotante superior al 15% y una rotación de capital flotante anual superior al 15 %.

Los valores que cumplen estos requisitos son ordenados por su capitalización bursátil, incluyéndose a los 20 primeros en el IBEX Medium Cap y a los 30 siguientes en el IBEX Small Cap.

Cada vez existen más productos de inversión y ahorro basados en alguno de estos dos índices debido a que las entidades bancarias y financieras tratan de ofrecer una mayor oferta de productos.

2.2.4.2 Históricos

El siguiente listado muestra los índices históricos del IBEX 35 que se crearon en 1998 y que desaparecieron el 1 de Julio de 2005 con la entrada de IBEX MediumCap y SmallCap [B10].

- **IBEX "Utilities".** Incluía compañías del sector de los servicios públicos cotizadas en el SIBE.
- **IBEX Financiero**. Compuesto por los valores de finanzas, banca y seguros cotizados en el SIBE.
- **IBEX Industria y Varios**. Reflejaba la evolución de las compañías del sector industrial y de servicios.
- **IBEX COMPLEMENTARIO®**. Incluía los valores que formaban parte de los índices sectoriales de la Sociedad de Bolsas pero que no estaban incluidos en el IBEX 35.

Además en abril de 2000 se creó el IBEX Nuevo Mercado® que incluía empresas de nuevas tecnologías que se negociaban en el Nuevo Mercado. Finalmente, desapareció el 3 de diciembre de 2007.

2.3 Gestor de portafolios financieros

Un gestor de portafolios financieros se encarga de informar al usuario cómo se encuentra el mercado financiero actual, la historia de la Bolsa y todos los movimientos que ha realizado el usuario.

Para ello, hay que distinguir cuatro grandes grupos dentro del portafolio:

1. Consulta de valores: El portafolio debe permitir al usuario ver a tiempo real los valores actuales del IBEX 35. Se muestra el contenido a través de tablas o gráficas.

   Permite consultar todos los datos históricos de los componentes que forman el IBEX 35 junto a sus gráficas de valores y de volumen.

   Además, se describe detalladamente cada compañía del IBEX 35.

2. Consulta de noticias actualizadas. Se debe mostrar un listado con todas las noticias financieras actuales. Así, el usuario sabe en todo momento cómo se encuentra el mercado.

3. Formularios de entrada de datos: Cuando un usuario ha realizado una compra o venta de una acción, debe poder introducirlo en su portafolio. Para ello, es necesario que el portafolio disponga de dichos formularios. Lógicamente, todos los valores introducidos son comprobados antes de la ejecución del formulario.

4. Estado actual del usuario en la Bolsa: El portafolio debe mostrar una sección con todo el historial del usuario. Dentro se incluyen los valores comprados, los valores vendidos, las ganancias o pérdidas y otros aspectos financieros como el stop-loss, la volatilidad y la rentabilidad.

2.3.1 Elementos financieros a considerar

En la presente sección se explican los elementos de finanzas que se han incorporado en el gestor de portafolios para ayudar al usuario a tomar decisiones informadas sobre la gestión de su cartera.

Éstos son los siguientes:

- Stop-loss
- Rentabilidad del portafolio
- Rentabilidad simple acumulada
- Ganancia de cada lote vendido
- Ganancia total
- Volatilidad

Además de los cuadros de evolución de precios y volúmenes de transacciones para cada título bursátil.

2.3.1.1 Stop-loss

El *stop-loss* permite limitar las pérdidas en Bolsa. Son órdenes condicionadas de venta que permiten al usuario limitar sus pérdidas cuando el valor contratado experimenta un retroceso importante en su cotización. Esto permite al inversor evitar que dicho retroceso pueda llevarse gran parte del ahorro invertido.

Por lo tanto, "s*top-loss"* es una orden de venta de acciones cuyo envío a bolsa se supedita a que se cumpla una condición en el precio fijado por el usuario. Cuando se cumple esta condición, se envía al mercado una orden limitada a un precio del que también se ha informado. Así, por ejemplo, cuando se compran determinados títulos, a la vez se puede ordenar que se vendan automáticamente si su cotización cae en un 5%. La condición de activación podrá ser "menor o igual" a un precio.

En el caso del portafolio, el usuario introducirá un **porcentaje** que se aplicará sobre el valor del título comprado y el sistema indicará a través de un correo electrónico, cuándo se ha llegado al valor mínimo. La fórmula a seguir es:

$$\text{Precio mínimo (Stop - loss)} = \text{PrecioCompra} - \text{Porcentaje} \times \text{PrecioCompra}$$

Ecuación 2-2: Stop-loss

2.3.1.2 Rentabilidad del portafolio

La rentabilidad porcentual (R) del portafolio en un instante de tiempo T se define como:

$$R = \frac{100}{M} \times \sum_{i=1}^{k} \left[\left(\mathrm{Pr}ecioEnT_{(i)} - \mathrm{Pr}ecioCompra_{(i)} \right) \times CantidadComprada_{(i)} \right]$$

Ecuación 2-3: Rentabilidad del portafolio

Donde:

- PrecioEnT es el valor del título i-ésimo en la fecha *t*.
- PrecioCompra es el valor por el que se compró el título i-ésimo.
- CantidadComprada es la Cantidad de títulos *i* comprados
- K es el número de títulos distintos adquiridos
- M es el valor neto acumulado de la cartera, calculado como:

$$M = \sum_{i=1}^{k} (\mathrm{Pr}ecioCompra_{(i)} \times CantidadComprada_{(i)})$$

Ecuación 2-4: Valor neto acumulado de la cartera (M)

La *Ecuación 2* se puede simplificar considerablemente al tener como denominador M. Una forma más rápida de calcular R es:

$$R' = 100 \times \left[\frac{\sum_{i=1}^{k} \left(\mathrm{Pr}ecioEnT_{(i)} \times CantidadComprada_{(i)} \right)}{M} - 1 \right]$$

Ecuación 2-5: Cálculo simplificado de la rentabilidad del portafolio

Un inversor usa la rentabilidad del portafolio para saber si es buen momento de vender algunos de sus títulos. Cuando el porcentaje es positivo y alto, significa que los valores de los títulos a la fecha de la petición son más altos que el día que se compraron. Por lo tanto, sería un buen momento para venderlos. A su vez, el inversor puede ir viendo que sucedería con la rentabilidad si vendiera solamente alguno de sus títulos. Así se mostrarían cuáles son los títulos activos que le dan más o menos ganancia.

2.3.1.3 Rentabilidad simple acumulada

La rentabilidad simple acumulada (RSA) de cada título a la fecha consultada se define como:

$$RSA = \frac{\mathrm{Pr}ecioEnT - \mathrm{Pr}ecioCompra}{\mathrm{Pr}ecioCompra} \times 100$$

Ecuación 2-6: rentabilidad simple acumulada

Donde:

- PrecioEnT es el valor del título consultado en la fecha *t*.
- PrecioCompra es el valor por el que se compró el título consultado.

El valor de RSA permite saber al inversor si el título que ha comprado le puede ofrecer beneficio o pérdida.

2.3.1.4 Ganancia de cada lote vendido

La ganancia de cada lote vendido es el producto del número de títulos vendidos por la diferencia entre el precio de la venta y el precio de la compra. La formula es:

$$GananciaIndividual = (\mathrm{Pr}ecioVenta - \mathrm{Pr}ecioCompra) \times CantidadVendida$$

Ecuación 2-7: Ganancia de cada lote vendido

Lógicamente, si el valor obtenido es negativo se han producido pérdidas.

2.3.1.5 Ganancia total

La ganancia total es la suma de todas las ganancias individuales.

$$GananciaTotal = \sum_{i=1}^{títulosvendidos} \left[\left(\mathrm{Pr}ecioVenta_{(i)} - \mathrm{Pr}ecioCompra_{(i)} \right) \times CantidadVendida_{(i)} \right]$$

Ecuación 2-8: Ganancia total

2.3.1.6 La volatilidad

En finanzas, la volatilidad tiene una gran influencia a la hora de valorar activos financieros. Mide la velocidad con la que varía el precio del activo subyacente, al alza o a la baja.

Supongamos 2 acciones:

- La acción A durante los 4 últimos años se mueve en un rango de un 5% en un mes como media.
- La acción B durante los 4 últimos años se mueve en un rango de un 20% en un mes como media.

Entre el máximo y el mínimo del mes se podrían obtener los siguientes valores:

Acción A		Acción B	
Mínimo	**Máximo**	**Mínimo**	**Máximo**
10	**10,5**	**10**	**12**
9	**9,45**	**9**	**10,8**
8	**8,4**	**8**	**9,6**
7	**7,35**	**7**	**8,4**

Tabla 2-2: Comparación de valores entre dos acciones

Se puede comprobar que la acción B tiene un intervalo de valores mucho más amplio que la acción A. Se dice que tiene mayor volatilidad.

Existen tres tipos de volatilidades:

1. **Volatilidad histórica:** Es conocida y se puede medir al tener el activo subyacente en el pasado. Un mismo activo subyacente suele tener volatilidades similares para diferentes períodos de tiempo, aunque no siempre es así. Por ejemplo, puede tener una volatilidad en los últimos 3 meses distinta de la que tuvo en el último año.
2. **Volatilidad futura:** Es la que tendrá el activo subyacente en el futuro. Es imposible conocer de antemano aunque existen algoritmos que calculan aproximaciones. Es la volatilidad más importante ya que se conocería qué va a hacer la cotización de ese activo subyacente.
3. **Volatilidad implícita:** Es la que se cotiza en el mercado actualmente. Es una opinión media de los intervinientes en el mercado de acciones sobre cuál será la volatilidad futura. La volatilidad implícita varía constantemente, igual que la cotización de las acciones.

2.3.1.6.1 Cálculo de la volatilidad

La volatilidad es un término tan abierto, que hace que se pueda calcular de muchas maneras. En el portafolio se ha usado la volatilidad histórica no centralizada.

A continuación se explica cómo se calcula la volatilidad histórica centralizada y la no centralizada y se comparan sus valores:

Volatilidad centralizada:

Sea S_i el precio de una acción al final del día *i* y sean $u_i = \ln \frac{S_i}{S_{i-1}}$ los precios relativos de *i*=1, 2, 3,..., n siendo *n* el número de valores dentro del ejemplo histórico.

La volatilidad histórica centralizada se calcula a partir de:

$$\sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}\left(u_i - \bar{u}\right)^2}$$

Ecuación 2-9: Volatilidad histórica centralizada

Donde $\bar{u}$ es la media definida como

$$\bar{u} = \frac{1}{n}\sum_{j=1}^{n} u_j$$

Ecuación 2-10: Media usada para la volatilidad

El valor de σ de la ecuación 2-9 da la volatilidad estimada para un intervalo. Para extender la volatilidad a otros periodos, hay que escalar el resultado con un factor *h*, que indica el número de intervalos para el periodo.

Por ejemplo, si se quiere extender al año la volatilidad sería:

$$\sigma_{an} = \sigma \times \sqrt{h}$$

Ecuación 2-11: Volatilidad escalada

Si se usan datos diarios, el intervalo es una operación diaria y se usa h=252, si el intervalo es semanal, h=52 y si es mensual, h=12.

La ecuación 2-9 es simplemente una desviación estándar de la serie de ejemplos u_i.

La tabla 2-3 muestra un ejemplo de volatilidad histórica. Se encuentran los 22 precios diarios de ABE.MC del mes de julio del 2001. Esto ha devuelto 21 salidas con una media de los valores uj de 0,0023797, una desviación estándar de 0,0124468 y una volatilidad histórica anual del 19,76%.

Volatilidad no centralizada:

En la tabla 2-3 se ve que la media de las salidas uj es un valor muy cercano a cero. Esto suele ocurrir en los mercados de valores. Si se eliminara la media de la ecuación 2-9, la volatilidad calculada pasaría a ser no centralizada y sería:

$$\sigma' = \sqrt{\frac{1}{n}\sum_{j=1}^{n} u_j^2}$$

Tabla 2-3: Volatilidad histórica no centralizada

Aplicando dicha fórmula se ha obtenido una desviación estándar de 0,0126834 y una volatilidad del 20,13%.

Fecha	Precio	u_j	u_j^2
02/07/2001	20,49		
03/07/2001	20,72	0,0111625	0,0001246
04/07/2001	21,01	0,0138991	0,0001932
05/07/2001	21,55	0,0253773	0,0006440
06/07/2001	21,9	0,0161108	0,0002596
09/07/2001	21,65	-0,0114812	0,0001318
10/07/2001	21,5	-0,0069525	0,0000483
11/07/2001	21,15	-0,0164130	0,0002694
12/07/2001	20,95	-0,0095013	0,0000903
13/07/2001	20,95	0,0000000	0,0000000
16/07/2001	20,38	-0,0275846	0,0007609
17/07/2001	20,4	0,0009809	0,0000010
18/07/2001	20,71	0,0150818	0,0002275
19/07/2001	20,9	0,0091325	0,0000834
20/07/2001	21,13	0,0109447	0,0001198
23/07/2001	21,15	0,0009461	0,0000009
24/07/2001	21,05	-0,0047393	0,0000225
25/07/2001	21	-0,0023781	0,0000057
26/07/2001	21	0,0000000	0,0000000
27/07/2001	21,24	0,0113638	0,0001291
30/07/2001	21,43	0,0089056	0,0000793
31/07/2001	21,54	0,0051199	0,0000262
	Media	0,0023797	
	Desviación Est.	0,0124468	
	Volatilidad anual σ	19,758603%	
		σ'	0,0126834
		Volatilidad anual σ'	20,134230%

Tabla 2-4: Volatilidades de ABE.MC

A priori, los valores son muy parecidos, pero para los analistas financieros es mejor usar la volatilidad no centralizada por las siguientes razones:

1. Hay un parámetro menos a estimar al evitar la media.
2. Hace que la estimación de la volatilidad sea más cercana a lo que afecta a los P/L (Ganancias/Pérdidas) de los comerciantes.

   Imaginemos que se usa los retornos de una semana para calcular la volatilidad y que el valor de cada día es un 2% menor. Si la media se resta de cada día, según la fórmula, la volatilidad de la semana sería de cero. Esto no concuerda con las expectativas de los inversores durante esta semana, ya que la volatilidad real debería ser mucho mayor que cero.
3. Las volatilidades sin media son mejores para el pronóstico de futuras volatilidades.
4. Según el artículo de Kotzé [K09], si se usa la desviación estándar con la media, un impreciso cálculo de la media reduciría la precisión de la volatilidad. Esto es cierto para periodos cortos del orden de 1 a 3 meses.

Capítulo 3

Plataforma de desarrollo Joomla!

3.1 Introducción

En el presente capítulo se definirá qué es un sistema de gestión de contenidos (CMS) y cuál es su estructura en general.

Concretamente, se expondrá la plataforma de desarrollo Joomla!, usada para la implementación del proyecto. Se analizarán las características principales y los elementos que componen un sitio creado con Joomla!.

La exposición se fundamenta en la información recogida en el portal oficial de Joomla! [JOOMLA], y en los libros de introducción a Joomla! [G08], [N08] y [T07].

3.2 Los CMS

Un sistema de gestión de contenidos o *Content Management System* es una colección de scripts que permiten crear una estructura de soporte para la creación y administración de contenidos. Además, separan el contenido de la presentación de un sitio web. Los CMS son usualmente muy sofisticados y pueden tener contenidos complejos como foros, tiendas online y *RSS new feeds*[2]. Cada vez más, las páginas web están gestionando su contenido a través de CMS.

La mayoría de CMS comerciales son caros, con valores que oscilan entre 40.000 – 300.000 euros, pero en los últimos años están apareciendo muchas alternativas *Open Source* bastante competitivas. Los CMS de código abierto son cada vez más fiables y son muy usados en importantes proyectos de empresas, organizaciones sin ánimo de lucro y otras compañías.

La gran ventaja de un CMS es que separa las responsabilidades involucradas en el desarrollo de una página web: Una página web puede ser creada por un diseñador, mientras que el contenido puede ser gestionado por personal no técnico.

Un CMS moderno es definido por su capacidad de administración y publicación de contenido. Además, muchos CMS tienen más funcionalidades gracias a la cantidad de extensiones y aplicaciones complementarias.

[2] RSS es una familia de formatos de fuentes web codificados en XML. Se utiliza para difundir información actualizada frecuentemente a usuarios que se han suscrito a la fuente de contenidos. El formato permite distribuir contenidos sin necesidad de un navegador, utilizando un software diseñado para leer estos contenidos RSS (agregador).

Pero, ¿cómo funcionan los CMS exactamente? Para entender el poder de un CMS, primero hay que entender unos cuantos conceptos de las páginas web tradicionales. Conceptualmente, hay dos aspectos principales en una página: La presentación y el contenido. En las últimas décadas, ha habido una evolución en cómo interactúan estas dos partes:

(1) **Páginas Web Estáticas:** El contenido y la presentación se encuentran en el mismo fichero. En la ilustración 3-1, se comprueba que todo el contenido que hay en el archivo de la izquierda solamente genera una línea de texto en el navegador.

A pesar de que se considera un sistema muy antiguo, muchos diseñadores aún lo usan para crear páginas web. Tiene dos desventajas que lo caracterizan:

- Es muy difícil de editar y de mantener. Por ejemplo, si se quiere cambiar el color de todos los títulos, hay que hacerlo uno por uno.
- El tamaño de los archivos puede ser excesivamente grande a causa de tener que unir el contenido con el estilo y diseño de la página.

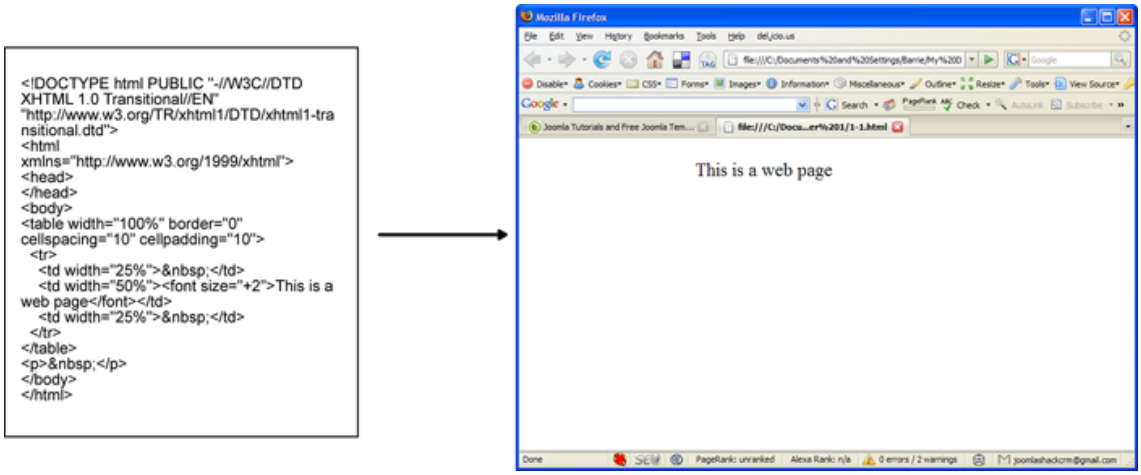


Ilustración 3-1: Página web estática

(2) **Página web con hojas de estilo (*Cascading Style Sheet* – CSS)**: El contenido y la presentación están separados.

Los archivos CSS son un mecanismo sencillo de gestión del estilo de la página como el color de letra, la fuente, el estilo, el espaciado... Ahora el fichero que contiene el contenido es mucho más pequeño, lo que permite una carga más rápida de la página web. Además, el mantenimiento y la revisión de la página son más fáciles. Para cambiar el color de todos los títulos, solamente hay que ir al fichero CSS y cambiar una línea de código. Tal como se muestra en la ilustración 3-2, el contenido se edita en el archivo superior, mientras que el estilo en el archivo inferior.

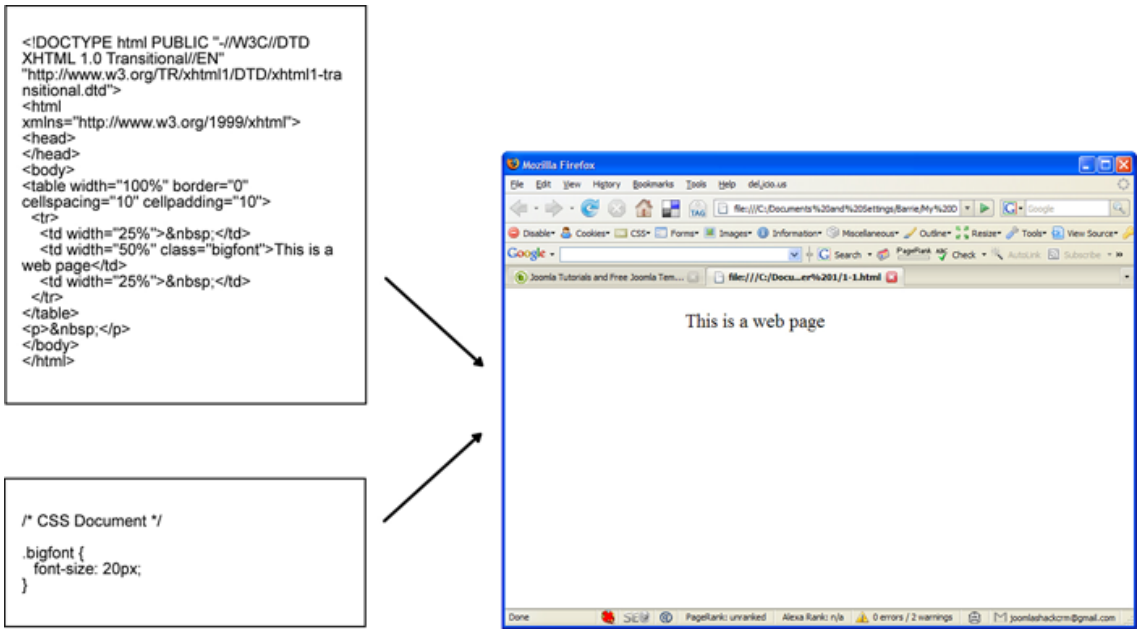


Ilustración 3-2: Página web con hojas de estilo

(3) **Páginas web dinámicas:** Tanto el contenido como la presentación están separados por la misma página web.

Un CMS simplifica la gestión de sitios web con la creación de páginas web dinámicas. Mientras que un CSS separa la presentación del contenido, un CMS separa el contenido de la página. Por lo tanto, *un CMS hace para el contenido lo que un CSS hace para la presentación.*

La ilustración 3-3, muestra la separación entre el contenido, el diseño y la página para obtener el mismo resultado que en los anteriores casos.

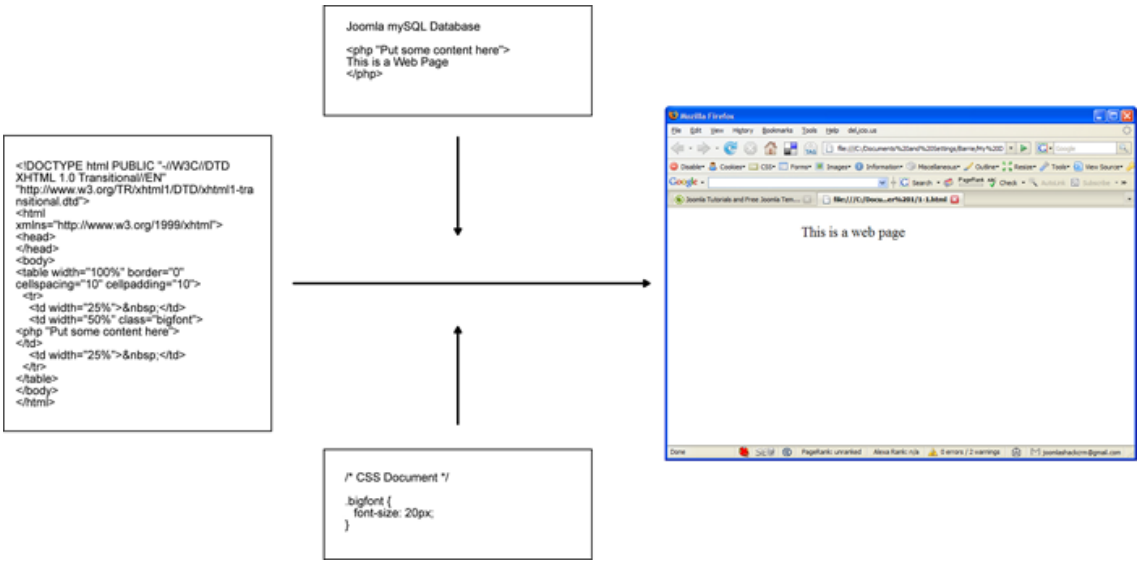


Ilustración 3-3: Página web dinámica

La instrucción “poner el contenido aquí” explica al CMS que tiene que coger el contenido de la Base de Datos, purificarlo y ponerlo en un sitio designado de la página. En principio, parece que eso no sea tan especial, pero en realidad es muy poderoso.

Permite separar la responsabilidad del desarrollo de la página. Un diseñador se encarga de la presentación y del diseño, mientras que los “procuradores” no técnicos se pueden encargar del contenido. Muchos CMS han construido herramientas para manejar la publicación del contenido.

Por lo tanto, un CMS crea dinamismo en una página. La página no existe hasta que se clica a un enlace para verlo. Esto implica que el contenido puede ser distinto cada vez que se visita. Esto significa que el contenido puede ser actualizado y basado en la interacción de los usuarios con la página.

La siguiente tabla muestra una comparativa entre páginas estáticas y CMS donde se resume el concepto de *difícil instalación pero fácil crecimiento.*

Página web estática	*Content Management System*
Creación inicial de página muy fácil	*Consume mucho tiempo en la creación inicial de la página porque hay que instalar una infraestructura grande de ficheros, la Base de Datos se tiene que configurar y la plantilla tiene que estar diseñada antes de creación de la primera página.*
Contenido estático; Los cambios requieren técnicos expertos y múltiples instancias de contenido tienen que ser editadas individualmente en cada página.	*El contenido es dinámico; puede ser cambiado sin experiencia técnica y un simple cambio puede tener efecto en todas las páginas.*
Añadir nuevas funcionalidades es difícil y normalmente requiere un código muy concreto.	*La mayoría de CMS tienen extensiones que se "conectan" muy fácilmente.*

Tabla 3-1: Comparativa entre páginas web estáticas y CMS

3.3 ¿Por qué Joomla?

Para la elección de un buen CMS se deben tener en cuenta cinco puntos básicos, de acuerdo con el informe "*How to Choose a CMS in 5 easy steps!*" de *CMS Critic* (http://www.cmscritic.com/) que se puede consultar en el CD:

1. Si cumple los objetivos para los cuales se desea utilizar: No todos los CMS son iguales. Algunos están exclusivamente diseñados para la creación de blogs, otros para crear una red social, o incluso para la publicación.
2. Si posee las funcionalidades requeridas para el proyecto: Muchos CMS tienen pocas extensiones o *plugins* (por ejemplo: registro y administración de usuarios, foros, manejo de estadísticas y demás), pero en cambio son más rápidos de aprender.
3. Compatibilidad del CMS con el sistema operativo de tu servidor: Hay plataformas CMS basadas en PHP, otras en cambio en Java, C o Ruby Rails. Es aconsejable saber también el tipo de base de datos soporta el servidor.
4. Facilidad de diseño y presentaciones de página: Es interesante saber cómo son las plantillas o los temas de las páginas implementadas con el CMS para asegurarse trabajar con algo que se adecúe a las necesidades.
5. El tamaño y la actividad de la comunidad de ese CMS: Este punto es muy importante. Un CMS con mucha documentación y con una comunidad muy amplia ayuda a resolver las dudas más rápidamente.

Joomla cumple ampliamente la mayoría de estos puntos básicos. Además usa código abierto, en concreto PHP, que permite englobar un número mayor de servidores para ser instalado. Adicionalmente, se ubica entre los primeros CMS más populares y mejor valorados según estadísticas recogidas en sitios como *The Content Management Comparison Tool* del sitio web http://www.cmsmatrix.org.

Joomla! es un CMS de código abierto muy potente que ha crecido en popularidad desde su renombramiento de Mambo[3] en 2006. Sus dos características clave, facilidad de administración y flexibilidad en el uso de plantilla, lo ha hecho muy útil para el potenciamiento de cualquier sitio en internet, desde intranets de empresas hasta páginas de distritos escolares.

A finales de 2007, apareció la versión de Joomla 1.5, que significaba una mayor reescritura del software. Los cambios incluían una simplificación y agilización de los procesos de los usuarios encargados de escribir contenido, añadir extensiones y administrar el sitio. Fue tan significativo el cambio, que puede que una extensión de la versión Joomla! 1.0 no funcione en la versión actual. A pesar de ello, ya se han implementado programas que reformatean el código para que sean compatibles.

Los expertos recomiendan siempre la instalación de la última versión de Joomla! y la creación o instalación de extensiones con el formato de la versión 1.5.x.

[3] **Mambo** es un sistema de portales CMS basado en el lenguaje de programación PHP y base de datos SQL de código abierto. Basa todo su aspecto en *templates*.

3.3.1 Comunidad Joomla!

La existencia de una comunidad grande y activa es el factor importante para que un proyecto de código abierto tenga éxito. La comunidad Joomla! es ambas, simultáneamente, muy grande y muy activa. El foro oficial en http://forum.joomla.org/ tiene más de 250.000 elementos, haciéndolo uno de los foros más grandes en las comunidades web.

Adicionalmente, hay muchos foros en páginas internacionales Joomla! y en las respectivas páginas de sus desarrolladores. Hay, también, otros foros de terceras personas como www.joomlashack.com que contiene 120.000 usuarios.

Esto da una idea de lo grande que es la comunidad Joomla! y de lo mucho se puede extender.

3.3.2 Características de Joomla!

Joomla! tiene muchas características que no se aprecian en un principio. Cuando uno se descarga el fichero de Joomla! en www.joomlacode.org , se obtiene un fichero de unos 5MB que necesita ser instalado en el servidor web [Ver apéndice A]. Al ejecutar la instalación se extraen los ficheros y se introducen los contenidos de “relleno” dentro de la BD. Dichos contenidos son:

- Editores de texto para la simple creación y revisión de artículos desde el *frontend* o el *backend*.
- Registros de usuarios y restricción de la visualización de la página según el nivel de usuario.
- Control de la edición y la publicación del contenido basado en varios niveles de usuarios de administración.
- Encuestas.
- Formularios simples de contacto.
- Estadísticas públicas del sitio.
- Estadísticas detalladas del tráfico del sitio.
- Funcionalidad de búsqueda de contenido.
- Capacidad de enviar por email, convertir en PDF e impresión de los artículos.
- RSS.
- Sistema de calificación de contenido.
- Subministro de noticias de otros sitios *Newsfeeds.*

La creación de todo este contenido en una página estática tendría un coste económico muy elevado. Además, Joomla! tiene más de 4.000 extensiones, las más populares son:

- Foros.
- Calendarios.
- Carritos de la compra o *Shopping Carts.*
- Galerías de fotos.
- Formularios.
- Descarga de documentos y multimedia.

Cada extensión puede ser instalada dentro de la página Joomla! para extender la funcionalidad del sitio. En el apéndice B se explica cómo instalar una extensión. Concretamente, se instala el componente implementado para el proyecto.

3.3.3 El *Frontend* y El *Backend*

La versión 1.5 de Joomla! separa la zona de administración o *backend* de la zona de la página web o *frontend*.

Después de la instalación de Joomla! en realidad hay dos sitios:

- El sitio público (llamado *frontend*) donde puede acceder cualquier persona al introducir www.tusitio.com en el navegador.
- El sitio de administración (llamado *backend*) donde acceden los administradores del sitio al introducir www.tusitio.com/administrator.

Mientras que alguna administración se puede hacer vía el *frontend* del sitio, es mucho más eficiente gestionar el sitio a través del *backend*.

3.3.4 Elementos de una página web Joomla!

Un sitio web Joomla! contiene varios elementos que, interactuando juntos, producen la página web. Los tres elementos principales son el **contenido**, la **plantilla** y los **módulos**. El contenido es el núcleo de la página, la plantilla controla cómo el contenido es presentado y los módulos añaden funcionalidades dinámicas alrededor de los márgenes del contenido principal de la página.

En la ilustración se muestran los tres elementos de una página Joomla!. El contenido principal o *main body* es una columna larga con una lista de enlaces a tutoriales. Varios módulos son mostrados en las columnas de la derecha, de arriba y de abajo. La distribución y a posición del contenido está manejado por la plantilla junto al fichero CSS.

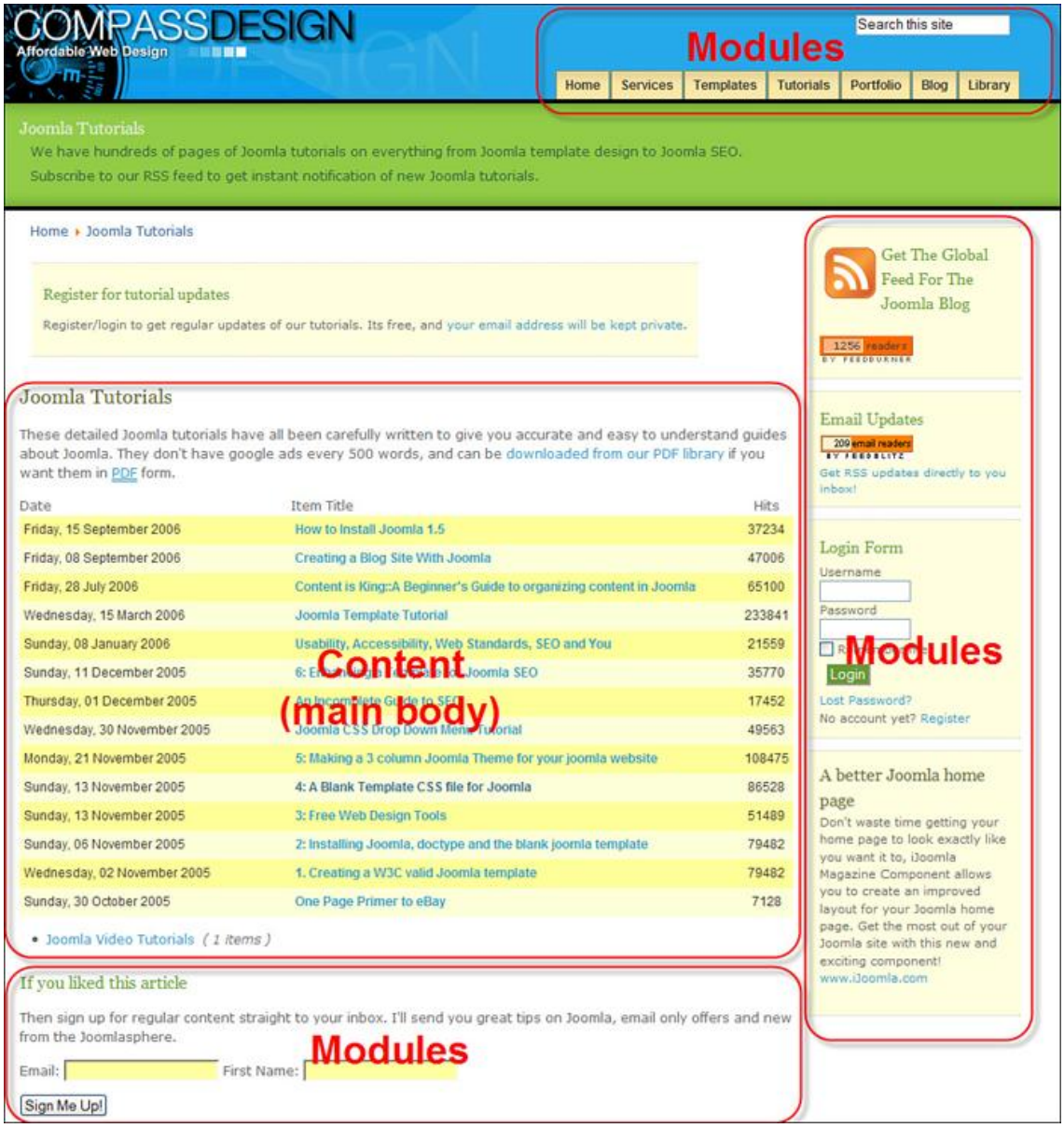


Ilustración 3-4: Elementos de una página web Joomla!

3.3.4.1 El contenido

La parte más importante de una página web es el contenido. Joomla!, como buen CMS, nos ayuda eficientemente a crear, publicar y administrar el contenido. A pesar de esto, para aquéllos que son nuevos en Joomla! la organización del contenido puede resultar un poco confuso.

La primera idea principal es pensar que *no hay páginas*. Hay que recordar que el contenido se guarda en una BD y Joomla! sólo sabe qué contenido tiene que ser usado cuando se clica a un enlace. Por lo tanto, en un sitio Joomla! la página es creada justo en el momento que se clica en el enlace.

Hay dos formas de generación de contenido:

- Con componentes
    - Artículos (organizados en secciones y categorías)
    - Otros componentes, como enlaces a páginas o de contacto.
- Con módulos

3.3.4.1.1 Cómo se organiza el contenido de artículos

Joomla! presenta dos opciones en la organización según el tamaño y la complejidad del sitio web:

- **Artículos no categorizados:** esta opción es la forma más simple de organizar el contenido. Consideremos la siguiente analogía. Estamos organizando una montaña de papeles del escritorio. Cada papel representa un simple artículo del contenido y el sitio web está representado por un archivador del escritorio. Entonces, si no hubieran muchos papeles, simplemente se cogerían todos y sin asignarles ninguna categoría se archivarían directamente. En cambio si hubiera miles de papeles, no funcionaría el uso de artículos no categorizados porque sería muy difícil encontrar uno en concreto posteriormente.
- **Secciones y Categorías:** Como la mayoría de CMS, Joomla! proporciona una jerarquía para la organización de un gran número de artículos. Ofrece dos niveles: el más alto llamado *secciones* y el situado debajo, las *categorías*. Por lo tanto, en la mayoría de veces se encontraría una estructura como la siguiente:

Sección 1:

Categoría A

Artículo i

Artículo ii

Categoría B

Artículo iii

Sección 2:

Categoría C

Artículo iv

Artículo v

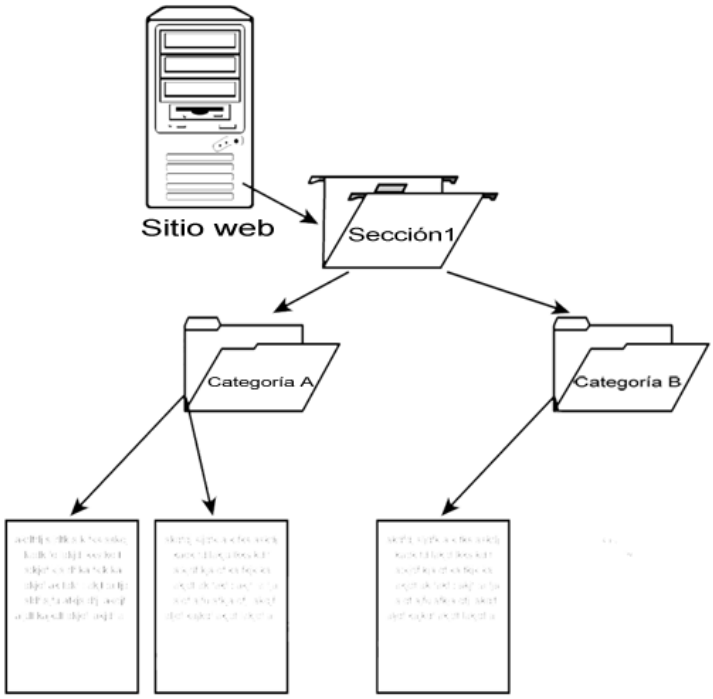


Ilustración 3-5: Estructura con secciones y categorías

Siguiendo con la analogía anterior. Ahora habría una carpeta con el nombre *Sección 1* y dentro de ella, dos carpetas más pequeñas llamadas *Categoría A* y *Categoría B*. Dentro de la primera carpeta se encontrarían los artículos i, ii y dentro de la segunda el artículo iii.

Por lo tanto, las secciones pueden tener 1 o más categorías como hijos, mientras que las categorías son los padres de los artículos. Un artículo siempre tiene que tener una categoría asignada para ser mostrado. Toda esta organización se consigue gracias a un componente llamado *Content article*, que se muestra los artículos en el cuerpo principal, *main body*.

En el capítulo 4 del libro *Joomla! 1.5. A User's Guide* [N08] se muestra cómo se crean secciones, categorías y artículos en el *backend*.

3.3.4.2 La plantilla

La plantilla es un conjunto de reglas sobre la presentación y el posicionamiento de componentes y módulos de la página web. Es decir, actúa como filtro para controlar la organización del sitio web. Una plantilla puede contener logotipos pero nunca contenido.

Este concepto de plantilla se muestra en la siguiente ilustración.

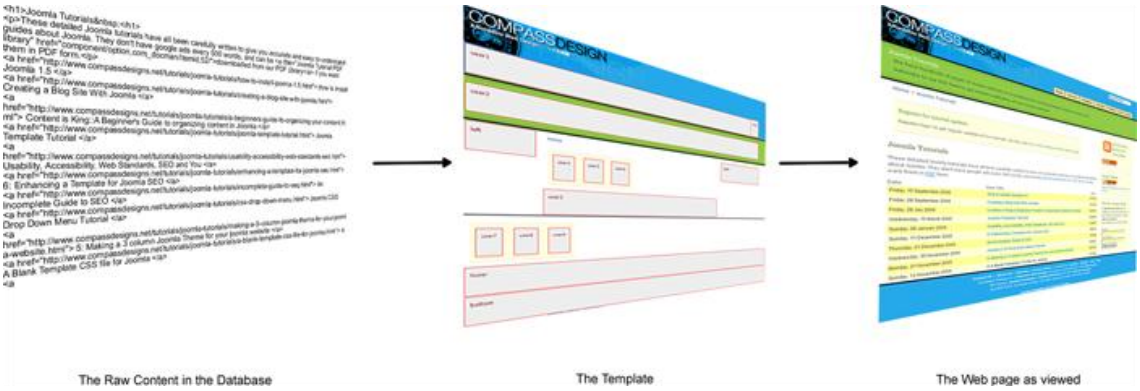


Ilustración 3-6: Uso de plantilla en un sitio web Joomla!

3.3.4.3 Los módulos

Los módulos son pequeños bloques funcionales que son mostrados alrededor de la parte principal de la página tales como encuestas o formularios de registro. Los módulos suelen mostrar otro contenido de la Base de Datos no relacionado con el contenido principal.

Los módulos, juntamente a los *plugin* y a los componentes, son referidos como *extensiones* de Joomla! porque extienden la funcionalidad principal del sitio web. En la siguiente sección se presentan algunas de estas extensiones.

3.3.5 Las Extensiones

Las extensiones son paquetes instalables que extienden la funcionalidad principal de Joomla.

Se pueden considerar cinco extensiones accesibles:

- Los componentes
- Los módulos
- Los *plugins*
- Las plantillas
- Los idiomas

3.3.5.1 Componentes

El componente es la extensión más compleja. Es como una mini aplicación que realiza alguna tarea sobre el contenido y se muestra en el cuerpo principal de la página. Por ejemplo, el componente *Content (com_content)*, es una mini aplicación que administra y muestra todos los artículos de alguna manera. Otro ejemplo es el componente foro que muestra temas, respuestas, etc. Pero no todos los componentes están relacionados con el contenido, sino que pueden tener funciones más complejas. Éste es el caso del componente *Registration(com_registration)* que gestiona el registro de usuarios.

El proyecto está basado en la creación del componente *com_finanzas* que realiza todas las funcionalidades definidas en el análisis del proyecto [Ver Sección Capítulo 4-

Análisis del proyecto].

Para manipular los componentes en el sitio Joomla! hay un menú llamado *Componentes* en la zona de administración, *backend*.

3.3.5.2 Módulos

Los módulos son más pequeños y menos complejos que los componentes. Normalmente aparecen a los extremos del sitio web rodeando el cuerpo principal. Suelen hacer tareas específicas relacionadas o no con el contenido. Por ejemplo, el módulo de últimas noticias *(lastest news module)* muestra enlaces con las últimas noticias añadidas en la página. Normalmente un módulo trabaja con un componente en particular. Otro ejemplo es el módulo de identificación (*login form module)* que permite a los usuarios registrados acceder a sus datos privados.

En el proyecto se ha añadido el módulo que muestra noticias relacionadas con finanzas obtenidas de las principales páginas financieras [YAHOOFINANZAS] y [ELECONOMISTA].

Para manipular los módulos en el sitio Joomla! hay un menú llamado *Gestor de módulos* dentro de *Extensiones* en la zona de administración.

3.3.5.3 Plugins

Un *plugin* o *mambot* es una parte especial de código que puede ser usado a través del sitio web y se ejecuta cuando se carga una página. Un ejemplo es el *email cloaking plugin*, que esconde las direcciones de correo con JavaScript para evitar que los robots *Spam* accedan a ellas. Los *plugins* son normalmente eventos manipuladores que funcionan en segundo plano para añadir funcionalidades.

Para manipular los *plugins* en el sitio Joomla! hay un menú llamado *Gestor de plugins* dentro de *Extensiones* en la zona de administración.

3.3.5.4 Plantillas

Como ya se ha comentado, las plantillas controlan el aspecto gráfico del sitio.

Para manipular las plantillas en el sitio Joomla! hay un menú llamado *Gestor de plantillas* dentro de *Extensiones* en la zona de administración.

3.3.5.5 Idiomas

Instalando el paquete de idiomas es posible internacionalizar el sitio Joomla! con distintos idiomas. En el proyecto se ha creado un diccionario para que el componente aparezca en inglés, castellano o catalán. En la sección donde explica como implementar un componente se explica cómo crear idiomas [Ver sección 6.3.8-Creación de traducciones].

Para manipular los idiomas en el sitio Joomla! hay un menú llamado *Gestor de idiomas* dentro de *Extensiones* en la zona de administración.

3.4 Estructura de ficheros Joomla!

Adentrándonos más en el código y en la implementación de Joomla! encontramos la siguiente estructura de ficheros cuando se instala el paquete en la carpeta raíz del sitio web:

***Administrator*:** Contiene todos los archivos relacionados con la zona de administración:

- *Backups*: Zona donde se encuentran los back-ups. Sirve para guardar versiones del sitio estables.

- *Cache*: Sitio donde se almacenan archivos temporales.

- *Components*: Código fuente de todos los componentes usados en la zona de administración. En esta carpeta se instalan los componentes con el nombre com-xxx

- *Help*: Manual de ayuda de Joomla! para el uso del *backend.*

- *Images*: Contiene las imágenes usadas.

- *Includes*: Contiene librerías, archivos PHP y JavaScript usados.

- *Language*: Contiene subcarpetas con nombres *en–GB, es–ES…*donde se encuentra el diccionario de los idiomas del sitio web.

- *Modules*: Código fuente de todos los módulos instalados en el sitio web Joomla! para la zona del *backend*.

- *Templates*: Plantillas instaladas en el sitio web.

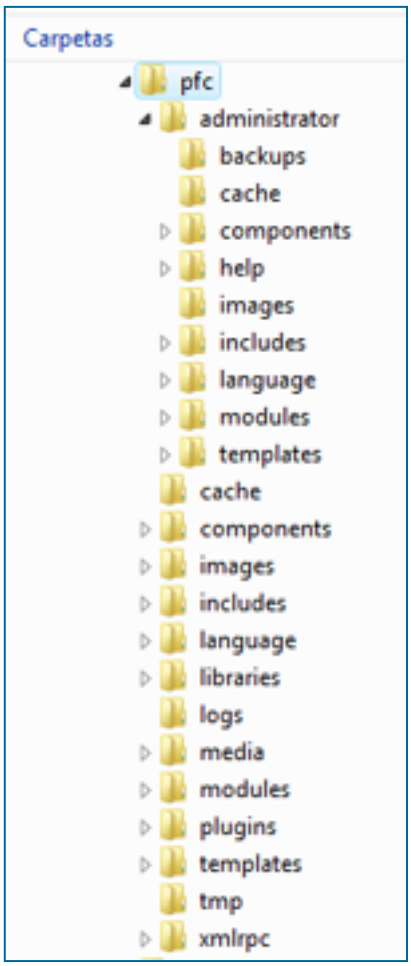


Ilustración 3-7: Estructura de ficheros Joomla!

***Cache*:** Sitio donde ese almacenan datos en cache para acceder más rápidamente a ellos.

Componentes: Código fuente de todos los componentes usados en el sitio web para el *frontend*. En esta carpeta se instalan los componentes con el nombre com-xxx

***Images*:** Contiene las imágenes usadas en el sitio web.

***Includes*:** Contiene cualquier archivo incluido en la implementación del sitio tales como JavaScript o PHP.

***Language*:** Contiene subcarpetas con nombres *en–GB, es–ES…* donde se encuentra el diccionario de los idiomas del sitio web.

***Libraries*:** Contiene las librerías usadas en la implementación del sitio.

***Logs*:** Contiene *Logs* que se ejecutan cuando hay algún error en el sitio web.

***Media*:** Contiene los archivos multimedia subidos al servidor.

***Modules*:** Código fuente de todos los módulos instalados en el sitio web Joomla! para la zona del *frontend*.

***Plugins*:** Código fuente de todos los *plugins* instalados en el sitio web Joomla!.

***Templates*:** Contiene todos los archivos implementados para cada plantilla usada.

***tmp*:** Archivos temporales del sitio.

***xmlrpc*:** *Extensible Markup Language-Remote Process Call*, traducido como llamada a procedimiento remoto por XML. Se trata de un protocolo utilizado para acceder a servicios de otras webs codificando los datos en XML y sirviéndose de HTTP como transmisor de mensajes

A parte de estas carpetas, hay ficheros de configuración, de instalación, de licencias, informativos, de créditos, copyright y el *htaccess*[4].

La carpeta *administrator* tiene una estructura muy similar a la carpeta *root*. Es importante diferenciar entre ellos dos ya que insertar el código al lugar incorrecto dará error en la ejecución.

[4] También conocido como *archivo de configuración distribuida*. Es un archivo que nos permite definir diferentes directivas de configuración para cada directorio (con sus respectivos subdirectorios) sin necesidad de editar el archivo de configuración principal de apache

Capítulo 4

Análisis del proyecto

4.1 Introducción

La mayor preocupación en la gente que trabaja en UML es saber cuáles son los artefactos mínimos indispensables para obtener beneficios tangibles en los proyectos software. Algunos puristas sugieren usar la mayoría de los artefactos en cada proyecto, lo que en muchas ocasiones resulta poco factible. Hay muchos proyectos en los que el analista no tiene suficiente tiempo para modelar todo el sistema, por lo que busca un mínimo indispensable en el uso de este estándar.

El uso de todos los artefactos ocasiona que al final se termine sin usar ninguno, usar el menos apropiado o usar algunos inadecuadamente. Así que es recomendable usar por lo menos el diagrama de casos de uso y el diagrama de clases. Con el primero, se obtienen mayores beneficios en cuanto a la calidad. Mientras que con el segundo, se diseña un sistema más orientados a objetos.

Se puede desarrollar un proyecto directamente haciendo el diseño de la aplicación, es decir, diseñar el diagrama de clases desde una sola perspectiva modelando las clases software a implementar. Esto puede suponer no sacar el máximo provecho a la aplicación y obtener unos resultados acabados. Lo ideal, es el desarrollo en dos ciclos: Uno de análisis y otro de diseño.

En un proyecto software, el análisis, y en concreto las obtención de los requisitos, es una de las fases más relevantes para el resultado final.

Extraer los requisitos de un producto de software es la primera etapa para crearlo. La captura, análisis y especificación de requisitos (incluso pruebas de ellos), es una parte crucial; de esta etapa depende en gran medida el logro de los objetivos finales.

Mientras que los clientes piensan que ellos saben lo que el software tiene que hacer, se requiere de habilidad y experiencia en la ingeniería de software para reconocer requisitos incompletos, ambiguos o contradictorios.

Se han ideado modelos y diversos procesos de trabajo para estos fines. Aunque aún no está formalizada, ya se habla de la Ingeniería de requisitos.

En este proyecto, la extracción de requisitos se ha obtenido a partir de varias entrevistas e intercambios de opinión por correo con el tutor del proyecto y el supuesto cliente de la aplicación, Argimiro Arratia.

Para llevar a cabo las entrevistas y recopilar los requisitos se siguieron los siguientes puntos [A06]:

- No invente una solución
- Formule preguntas abiertas
- Escuche
- No adivine los pensamientos
- Tenga paciencia

El resultado del análisis de requisitos con el cliente se plasma en un documento ERS, Especificación *de Requisitos del Sistema*, cuya estructura puede venir definida por varios estándares. En este caso, el resultado del análisis se muestra en el presente capítulo. Por este orden, se indican primero los requisitos funcionales y no funcionales, el modelo de caso de uso y el modelo de dominio o conceptual.

4.2 Requisitos del sistema

Los requisitos son capacidades y condiciones que el sistema debe cumplir. El primer gran reto de trabajo de los requisitos es encontrar, comunicar y recordar lo que realmente se necesita, de manera que tenga un significado claro para el cliente y los miembros del equipo desarrollador [L03].

Hay dos tipos de requisitos: los funcionales o de datos, los no funcionales y los de información.

4.2.1 Requisitos funcionales y de datos

Describen el funcionamiento del sistema: características y capacidades.

A continuación se muestran los requisitos funcionales del sistema:

Requisito	
Descripción	
El sistema deberá permitir a los administradores gestionar las compañías del IBEX 35, ya sea creando, modificando, eliminando o consultando sus perfiles.	
Justificación	
Los administradores deben poder modificar el listado de compañías que se muestran en el sistema para que éste sea ampliable.	

Tabla 4-1: RF - Gestión de compañías

Requisito	
Descripción	
El sistema deberá permitir a los usuarios registrados modificar y consultar su perfil o datos personales.	
Justificación	
Los usuarios registrados pueden cambiar su información personal en cualquier momento.	

Tabla 4-2: RF - Gestión del perfil

Requisito
Descripción
El sistema deberá permitir a usuarios registrados consultar noticias financieras actuales.
Justificación
Es interesante que los usuarios registrados estén informados de las últimas noticias del mercado financiero.

Tabla 4-3 : RF - Consulta de noticias

Requisito
Descripción
El sistema deberá permitir a los usuarios registrados consultar los valores del IBEX 35 actuales e históricos mediante tablas y gráficas.
Justificación
A los usuarios registrados les interesa saber el valor de las compañías del IBEX 35 en todo momento.

Tabla 4-4: RF - Consulta de valores

Requisito
Descripción
El sistema deberá permitir a los usuarios registrados gestionar sus acciones ya sea añadiendo, modificando o eliminando títulos mediante formularios de compra/venta.
Justificación
Los usuarios registrados quieren añadir datos en su portafolio para ver su estado actual en el mercado financiero.

Tabla 4-5: RF - Gestión de acciones

Requisito	
Descripción	
El sistema deberá permitir a los usuarios registrados consultar sus datos financieros tales como ganancias o pérdidas, rentabilidad, stop-loss y títulos comprados y vendidos.	
Justificación	
Los usuarios registrados quieren ver los resultados obtenidos de su portafolio.	

Tabla 4-6: RF - Consulta datos financieros personales

Requisito	
Descripción	
El sistema deberá permitir a los usuarios registrase e identificarse en el sistema.	
Justificación	
Los usuarios pueden tener un espacio registrado donde aparezca su portafolio personal.	

Tabla 4-7: RF - Registro de usuarios

4.2.2 Requisitos no funcionales

Describen las propiedades del sistema tales como el aspecto, la usabilidad, el rendimiento, el mantenimiento y la seguridad.

La mayoría de requisitos no funcionales de seguridad, fiabilidad y rendimiento son cubiertos por Joomla!. Por eso, a continuación se muestran los requisitos no funcionales cubiertos por el componente implementado:

4.2.2.1 Requisitos de aspecto (look & feel)

Requisito
Descripción
El sistema tendrá una interfaz atractiva y seria
Justificación
Es importante que el sistema llame la atención a tantos usuarios como sea posible para que se registren
Criterio de validación
El componente se adaptará a la plantilla que use el cliente.

Tabla 4-8: RNF - Interfaz atractiva

Requisito
Descripción
El sistema será fácil y sencillo de leer y usar
Justificación
El sistema será usado para distintos tipo de usuarios, de los que no se requiere ninguna formación previa y, por lo tanto, la interfaz tiene que ser fácil de utilizar.
Criterio de validación
Se realizaran consultarán varios sitios web y documentación sobre la sencillez de los sitios web.

Tabla 4-9: RF - Interfaz sencilla

Requisito	
Descripción	
El sistema tendrá que transmitir una imagen de seguridad.	
Justificación	
Para que los usuarios estén satisfechos con el sistema, es importante que se transmita seguridad.	
Criterio de validación	
Se consultarán varios sitios web de seguridad para adaptar una postura similar.	

Tabla 4-10: RNF – Imagen de seguridad

Requisito	
Descripción	
La moneda en la que ese expresarán todos los datos monetarios será el euro.	
Justificación	
El sistema se podrá usar desde cualquier parte del mundo y para facilitar el trabajo de convertir la moneda, se expresarán los datos con una única divisa.	
Criterio de validación	
Todos los datos monetarios se expresarán en euros.	

Tabla 4-11: RNF - Dato monetario

4.2.2.2 Requisitos de usabilidad

Requisito	
Descripción	El sistema permitirá al usuario elegir el idioma de la interfaz (español, catalán, inglés).
Justificación	Los usuarios se sentirán más cómodos usando la lengua más comuna para ellos.
Criterio de validación	Los usuarios podrán elegir el idioma.

Tabla 4-12: RNF - Idioma

Requisito	
Descripción	El usuario no requiere de ninguna información previa para usar el sistema.
Justificación	Cuanta menos complicación tenga el sistema, más usuarios la podrán usar.
Criterio de validación	El sistema está implantado en una página web Joomla!, por lo tanto, la única complicación será navegar a través de la red.

Tabla 4-13: RNF - Facilidad de uso

Requisito	
Descripción	El sistema tendrá una imagen común en toda su estructura.
Justificación	Para que un usuario se familiarice y recuerde con facilidad el sistema, es importante, tener un estilo constante para toda su estructura.
Criterio de validación	El estilo de la presentación será constante.

Tabla 4-14: RNF - Estructura

4.2.2.3 Requisitos de rendimiento

Requisito
Descripción
El sistema soportará múltiples conexiones de los usuarios.
Justificación
Es importante el hecho que el sistema está basado en una página web y, por lo tanto, tenga que soportar múltiples accesos.
Criterio de validación
El sistema permitirá el número de conexiones múltiples que Joomla! consienta.

Tabla 4-15: RNF - Múltiples usuarios

Requisito
Descripción
El sistema tendrá una alta disponibilidad.
Justificación
Al tratarse de una página web, el sistema siempre estará accesible a no ser que se caiga el servidor.
Criterio de validación
El sistema está disponible las 24 horas del día y los 365 días del año.

Tabla 4-16: RNF - Accesibilidad

Requisito
Descripción
El sistema tiene que actualizarse constantemente.
Justificación
Al tratarse de un sistema financiero, los datos se tienen que actualizar al momento para que el usuario tenga la información correcta.
Criterio de validación
Los datos financieros estarán actualizados según los datos obtenidos en Yahoo Finances.

Tabla 4-17: RNF - Actualizaciones

4.2.2.4 Requisitos operacionales y entorno

Requisito	
Descripción	El sistema funcionará con cualquier navegador web.
Justificación	Es importante que el sistema se pueda cargar desde cualquier navegador web para facilitar el acceso a todos los usuarios posibles.
Criterio de validación	La funcionalidad y la visualización del sistema será la misma independientemente del navegador web usado.

Tabla 4-18: RNF - Navegador

Requisito	
Descripción	Para usar el sistema no hace falta instalar ningún componente adicional al ordenador.
Justificación	Al tratarse de una página web, no se requiere instalaciones adicionales y, por lo tanto, será más fácil y accesible.
Criterio de validación	El sistema está basado en una página web, por eso no hace falta ningún programa adicional a excepción claro está del mismo sistema.

Tabla 4-19: RNF - No hay instalaciones adicionales

4.2.2.5 Requisitos de mantenimiento

Requisito	
Descripción	
El mantenimiento del sistema es transparente al usuario.	
Justificación	
El usuario final del sistema no se dará cuenta de los cambios que se hagan en el sistema.	
Criterio de validación	
Los usuarios pueden usar el sistema mientras se realiza el mantenimiento ya que esté se hará localmente.	

Tabla 4-20: RNF - Mantenimiento

4.2.2.6 Requisitos de seguridad

Requisito	
Descripción	
Los usuarios se podrán autentificar en el sistema de forma única y de esta forma tendrán total privacidad en sus datos.	
Justificación	
El usuario proporcionará datos privados que no querrá que se puedan ser vistos por terceras personas, por lo tanto, el sistema proporcionará las herramientas de seguridad de Joomla! para que el usuario tenga confianza.	
Criterio de validación	
El usuario tendrá un único identificador para autentificarse en el sistema.	

Tabla 4-21: RNF - Identificación

Requisito
Descripción
El sistema garantizará la integridad de los datos, es decir, que no hayan sido modificados por terceras personas.
Justificación
Es importante que los datos que se almacenan en el sistema sean correctos para el buen funcionamiento de éste.
Criterio de validación
El sistema garantizará que la modificación o actualización de los datos han sido realizadas por una persona autorizada en hacerlo.

Tabla 4-22: RNF - Datos de usuario

Requisito
Descripción
El sistema garantizará que un usuario pueda acceder solamente a las zonas donde tiene autorización.
Justificación
Es necesario que el sistema restrinja el acceso a las personas no autorizadas para que no se puedan falsear o modificar ningún tipo de dato.
Criterio de validación
El sistema tendrá un sistema para detectar el tipo de usuario que se conecta y así saber si les tiene que restringir el acceso o no.

Tabla 4-23: RNF - Restricción de acceso

4.2.2.7 Requisitos culturales y políticos

Requisito
Descripción
El sistema mostrará las fechas siguiendo el formato AAAA/MM/DD.
Justificación
Para tener un convenio con todos los usuarios, se establecerá este formato.
Criterio de validación
Todos los datos del sistema estarán en formato AAAA/MM/DD.

Tabla 4-24: RNF - Formato fecha

Requisito
Descripción
El sistema mostrará un calendario donde el primer día de la semana será lunes.
Justificación
Existen distintos tipo de calendarios y, para evitar confusiones, el sistema establecerá este convenio.
Criterio de validación
Todos los calendarios empezarán su primer día de la semana con lunes.

Tabla 4-25: RNF - Calendario

Requisito
Descripción
El sistema no hará distinción según raza, religión o pensamiento político.
Justificación
El sistema no atendrá a los usuarios según sus creencias, etnias....
Criterio de validación
El sistema no hará ninguna pregunta personal al usuario con el objetivo de saber sus creencias, pensamientos, políticos, etc.

Tabla 4-26: RNF - Cuestiones políticas

4.2.2.8 Requisitos legales

Requisito
Descripción
Se cumplirá con la normativa de la LOPDP.
Justificación
Cualquier sistema que almacena información respecto a personas tiene que seguir esta ley.
Criterio de validación
Se comprobará jurídicamente que se cumple la normativa.

Tabla 4-27: RNF - LOPDP

4.2.3 Requisitos de información

Describen que información debe almacenar el sistema para satisfacer las necesidades de los clientes.

Requisito – Compañías del IBEX 35
Descripción
El sistema deberá almacenar la información correspondiente a las compañías del IBEX 35.
Datos específicos
Ticker ISIN Nombre Descripción Fecha inicio Domicilio Teléfono URL Uptoday Publicado Capital permitido Número de acciones País Mercado

Tabla 4-28: RI - Compañías del IBEX 35

Requisito – Clientes
Descripción
El sistema deberá almacenar la información correspondiente a los clientes.
Datos específicos
Username Nombre Apellido 1 Apellido 2 DNI Dirección CP Población País Email Teléfono

Tabla 4-29: RI - Clientes

Requisito – Datos históricos
Descripción
El sistema deberá almacenar la información correspondiente a los datos históricos del mercado de valores
Datos específicos
Ticker Apertura Máximo Mínimo Cierre Volumen Volumen ajustado

Tabla 4-30: RI - Datos históricos

Requisito – Títulos activos
Descripción
El sistema deberá almacenar la información correspondiente los títulos activos.
Datos específicos
Username Ticker Fecha compra Precio compra Cantidad comprada

Tabla 4-31: RI - Títulos activos

Requisito – Títulos históricos
Descripción
El sistema deberá almacenar la información correspondiente a los títulos históricos.
Datos específicos
Username Ticker Fecha compra Precio compra Cantidad comprada Fecha venta Precio Venta Cantidad vendida

Tabla 4-32: RI - Títulos vendidos

Requisito – Repositorio de noticias
Descripción
El sistema deberá almacenar la información correspondiente a los sitios web donde recoge las noticias.
Datos específicos
Nombre Alias Enlace Publicado Número artículos

Tabla 4-33: RI - Repositorio de noticias

4.3 Modelo de casos de uso

El modelo de casos de uso muestra la funcionalidad total del sistema. En un proyecto software, después de la definición del objetivo del proyecto y lo requisitos que cumple éste, se debe especificar QUÉ hace el sistema. Muchas veces se confunde el QUÉ con el CÓMO. En el análisis se describe el comportamiento externo del sistema siendo responsabilidad de otras fases del proyecto la explicación del funcionamiento interno.

Concretamente, se indica el curso típico de acontecimientos y algunos alternativos que se producen entre el sistema y el usuario. Para ello hay que identificar primero a qué tipo de usuarios se aplica el sistema.

Por lo tanto, en la presente sección, se explica el modelo de casos de uso donde se consigue:

- Encontrar los actores.
- Encontrar los casos de uso.
- Encontrar los límites del sistema.
- Encontrar las relaciones entre actores y casos de uso.

4.3.1 Los actores

Los actores especifican un papel ejecutado por un usuario u otro sistema que interactúe con el sistema. El actor debe ser externo al sistema y debe tener asociaciones exclusivamente para casos de uso, componentes o clases.

Una aplicación Joomla! permite la gestión de varios roles de usuarios. Esta plataforma permite tener ocho niveles de usuarios con diferentes permisos en el *frontend* y el *backend* del sitio web. La siguiente tabla describe los distintos niveles [N08]:

Usuario	***Frontend***	***Funciones Backend/Menús***
Público	Solamente navegar	No tiene acceso
Registrados	Puede ver contenido restringido	No tiene acceso
Autores	Puede crear contenido	No tiene acceso
Editores	Puede editar contenido	No tiene acceso
Publicadores	Puede publicar contenido	No tiene acceso
Manager	Como publicador	Gestión de media
		Gestión de menú
		Gestión de contenido
		Gestión de la página principal
		Gestión de componentes
		Ayuda
Administradores	Como publicador	Como manager
		Gestión de usuarios
		Instalar/Eliminar extensiones
		Gestión de módulos
		Gestión de *plugins*
		Comprobación global
Súper administradores	Como publicador	Como administrador
		Configuración del sistema
		Gestión de idiomas
		Gestión de plantillas
		Instalación de plantillas
		Instalación de idiomas
		Gestión de emails

Tabla 4-34: Niveles de control de acceso de usuarios en Joomla!

Los cinco primeros niveles muestran a los usuarios llamados usuarios del *frontend (frontend users)* mientras que los tres siguientes son los usuarios *backend (backend users)*.

En nuestro caso, se ha considerado que hay tres tipos de usuarios que usan el componente.

- **Administrador**: Cumple el rol de súper administrador en Joomla!. Es decir puede gestionar todas las cosas tanto en el *frontend* como en el *backend.*
- **Usuario no registrado**: Cumple el rol de usuario público. Solamente puede ver el contenido abierto del componente.
- **Cliente**: Rol de usuario registrado y puede consultar su contenido privado.

En la siguiente ilustración se muestra la jerarquía de usuarios empleada.

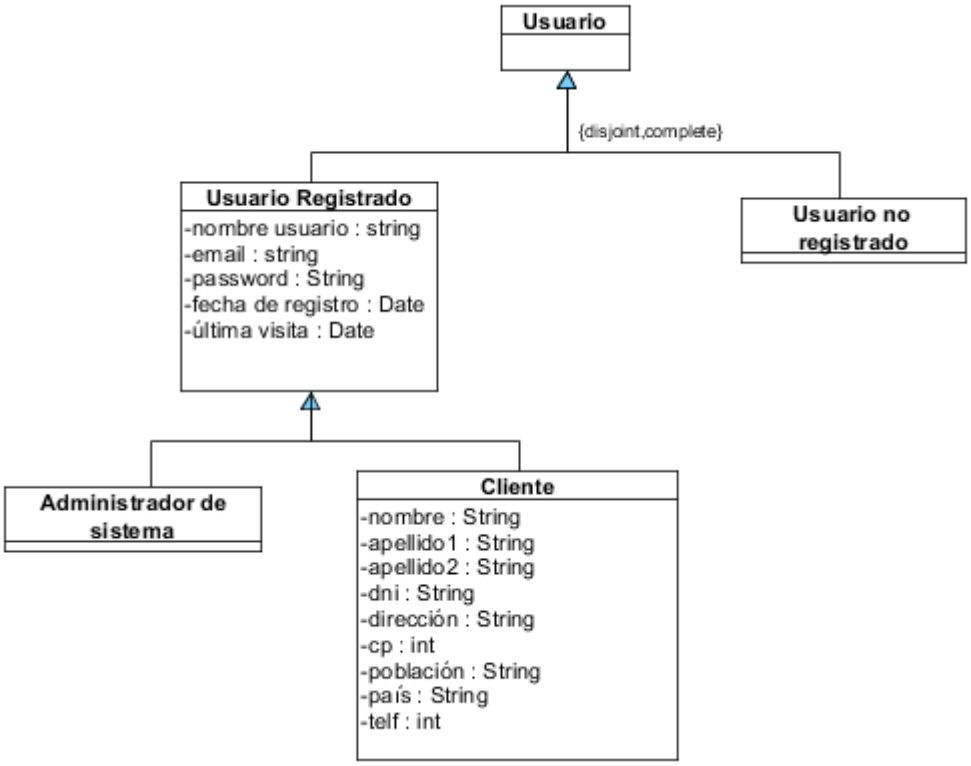


Ilustración 4-1: Jerarquía de usuarios

4.3.2 Diagrama y especificación de los casos de uso

Muchas veces, hay confusión entre el diagrama del caso de uso y el caso de uso en particular. Un diagrama muestra simplemente los nombres de los actores y de los casos de uso y las relaciones entre ambos. En cambio, una especificación o caso de uso, describe textualmente el curso de eventos que se producen para formar el caso de uso. Hay distintos tipos de formalidades para la especificación. En esta documentación se usa el grado de formalidad completo donde se muestra una descripción breve del caso de uso, el actor que actúa, un escenario principal y los alternativos, si los hay.

En la siguiente tabla se muestran todos los casos de uso surgidos en el análisis junto a sus actores:

Caso de uso	Actor
Consultar página de inicio	Usuario
Consultar compañía IBEX 35	Usuario
Consultar cotización del día	Usuario
Consultar cotizaciones históricas	Usuario
Consultar enlaces de interés	Usuario
Consultar noticias	Usuario
Registrarse	Usuario no registrado
Identificarse (Login)	Usuario registrado
Cerrar sesión (Logout)	Usuario registrado
Solicitar contraseña	Usuario registrado
Solicitar nombre de usuario	Usuario registrado
Darse de baja	Usuario registrado
Consultar datos de usuario	Usuario registrado
Modificar datos de usuario	Usuario registrado
Rellenar formulario acción comprada (añadir acción comprada)	Usuario registrado
Modificar acción comprada	Usuario registrado
Eliminar acción comprada	Usuario registrado
Rellenar formulario acción vendida (añadir acción vendida)	Usuario registrado
Modificar acción vendida	Usuario registrado
Eliminar acción vendida	Usuario registrado
Consultar portafolio	Usuario registrado
Consultar stop-loss	Usuario registrado

Modificar stop-loss	Usuario registrado
Consultar títulos activos	Usuario registrado
Consultar historial de ventas	Usuario registrado
Consultar rentabilidad porcentual acumulada	Usuario registrado
Consultar ganancias	Usuario registrado
Consultar lista compañías IBEX 35	Administrador
Añadir compañía IBEX 35	Administrador
Consultar compañía IBEX 35	Administrador
Modificar compañía IBEX 35	Administrador
Eliminar compañía IBEX 35	Administrador

Tabla 4-35: Casos de uso del sistema

Para tener una lectura más comprensible todos los diagramas junto con las especificaciones para cada usuario se muestran en el apéndice C. A continuación se ejemplifica cómo es el curso típico de un caso de uso; en concreto "rellenar formulario de compra", uno de los más relevantes del sistema:

Caso de uso	
Rellenar formulario de acción comprada	
Descripción	
Un usuario registrado quiere registrar acciones compradas en el sistema.	
Actor principal	
Usuario registrado (cliente)	
Precondición	
El usuario está registrado y ha iniciado sesión.	
Pos condición	
Se han registrado en el sistema las acciones de una compañía del IBEX 35 compradas por el usuario.	
Curso típico de acontecimientos	
Actor	**Sistema**
1. El usuario registrado pide rellenar el formulario de las acciones compradas.	

3. El usuario rellena dichos datos.	2. El sistema muestra un formulario para rellenar. En el formulario se pide: - La compañía IBEX - La fecha de compra - Precio de la compra - El número de acciones compradas. 4. El sistema comprueba que los datos estén correctos, es decir, que en la fecha introducida la compañía tenga acciones al precio indicado. Finalmente, registra la/s acciones comprada/s en la base de datos.

Cursos alternativos
3a.El usuario pide al sistema que le muestre el intervalo de precios de las acciones de una compañía para la fecha seleccionada. 3a1.El sistema muestra el máximo y mínimo de precios para ese día. 3a2.El caso de uso continúa en el paso 3. 4a. Los datos entrados son incorrectos. 4a1. El sistema notifica al usuario que los datos son incorrectos. 4a2. El usuario entra de nuevo los datos. 4a3. El caso de uso continúa en el paso 4.

Tabla 4-36: CU - Rellenar_formulario_acción_comprada

4.4 Modelo conceptual o modelo de dominio

La relevancia de un modelo conceptual radica en que si no se comprende el dominio del problema y las reglas de negocio de éste, hay pocas posibilidades de desarrollar un buen sistema. Cuando un cliente solicita el desarrollo de un sistema, el analizador debe plasmar todas las ideas propuestas en un documento. Muchas veces sucede que los documentos son tan largos que llegan a resultar ilegibles y a ser incomprensibles para el cliente. Existe una alternativa a tener que enlistar toda la información del dominio del problema en texto plano: El modelo conceptual. Éste está representación por un diagrama de clases que a su vez contiene una especificación.

Los elementos principales del diagrama de son:

- Conceptos. Se trata del elemento lógico o físico que ayuda a entender el problema. En el diagrama se representan con el símbolo de una clase. Ejemplo: Cliente y Compañías.
- Atributos. Es toda la información que caracteriza al concepto en el mundo real. Ejemplo: Nombre, apellidos y edad del cliente.
- Asociaciones. Son las relaciones lógicas o físicas que existen en el mundo real entre dos conceptos. En el diagrama se representa mediante la línea direccional que une a dos clases y que suele llevar el verbo de la relación. Ejemplo: El cliente vende un Ticker.
- Multiplicidad. Es el número de instancias de un concepto relacionados con otro concepto. Ejemplo: el IBEX 35 está formado-por 35 Compañías.
- Generalización. En lugar de poner una asociación para armar la frase “es-un-tipo-de” se puede poner una generalización. Ejemplo: El Cliente es un tipo de usuario registrado, al igual que sucede con el Administrador.
- Agregación y composición. Sirven para Indicar una relación donde uno de los conceptos es el contenedor del otro. Ejemplo: Un Foro contiene una lista de Temas.

Por lo tanto, un modelo de dominio es una representación de las clases conceptuales del mundo real. También se le puede denominar modelo conceptual, modelo de objetos de dominio o clases de análisis.

El siguiente diagrama de clases muestra el modelo conceptual del proyecto software implementado. Nótese que se han excluido los atributos de las clases para hacer más visible la ilustración. Todos los atributos se pueden consultar en la especificación de las clases.

4.4.1 Diagrama de clases.

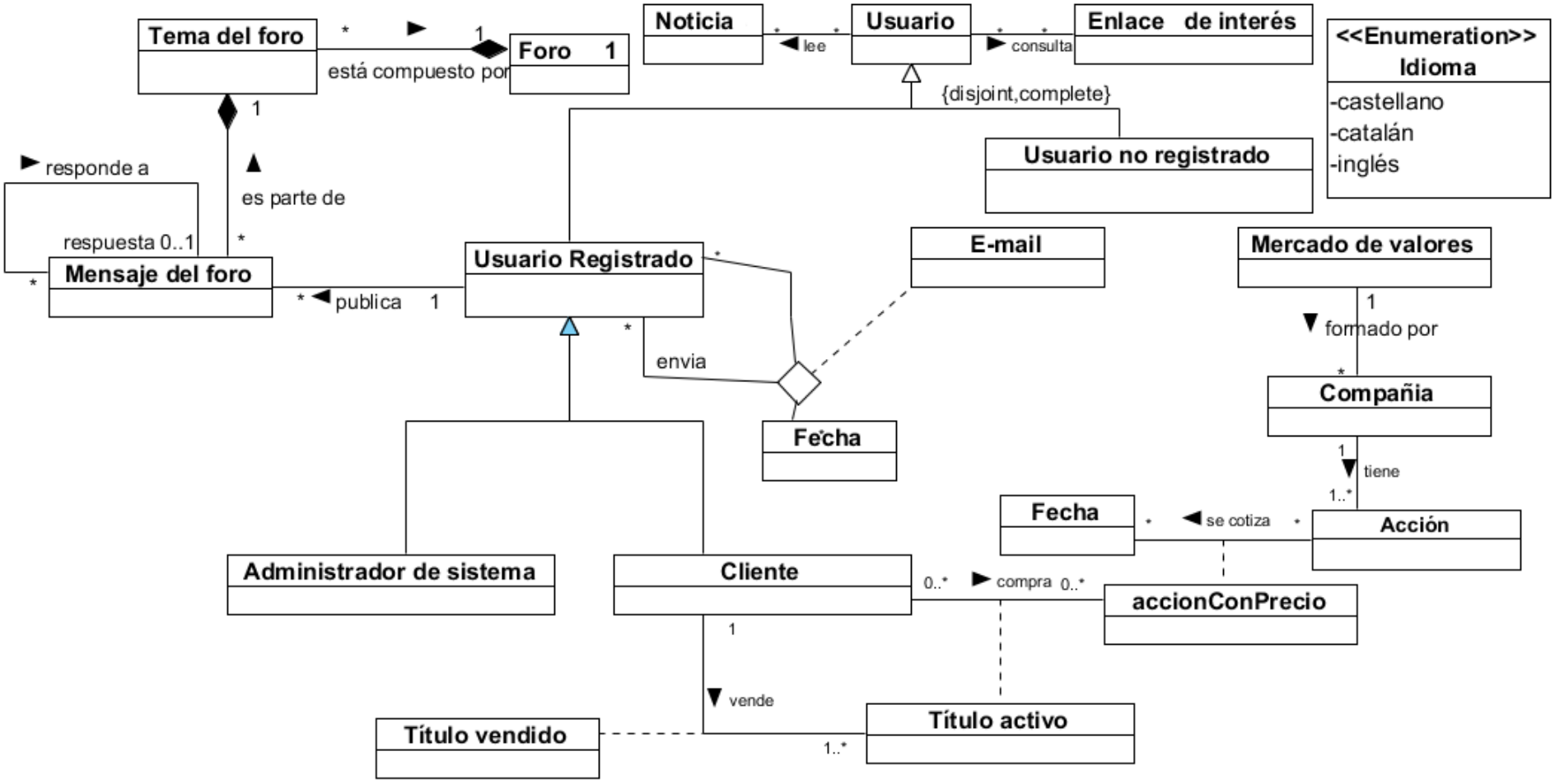


Ilustración 4-2: Modelo conceptual - Diagrama de clases

El modelo conceptual presenta las siguientes características:

Se genera una jerarquía de usuarios donde se diferencia primero entre los usuarios no registrados y los registrados. Los usuarios no registrados se consideran importantes porque en Joomla! se registra cuando un usuario en el *frontend* lee noticias o consulta enlaces de interés. De la clase usuario registrado derivan dos subclases: Cliente y Administrador. El administrador gestionará toda la información de las compañías y los mercados de valores ya sea eliminando, creando o modificando sus registros. No se han indicado estas operaciones en la clase administrador para hacer más comprensible el diagrama pero hay que tenerlas en cuenta.

Un mercado de valores está formado por varias compañías que a su vez tienen acciones. Estas acciones tienen un valor para cada fecha. De aquí surge la clase asociativa accionConPrecio (o dato histórico). Una acción para una fecha determinada tiene un precio.

El cliente se encarga de comprar acciones con precios para obtener un título activo. Esta nueva clase tiene los atributos obtenidos a partir de la asociación: nombre de usuario de Cliente y fecha, precio y ticker de accionConPrecio.

Dicho cliente, quiere a su vez, vender sus títulos activos. Aparece la restricción que la fecha de la venta tiene que ser posterior a la fecha de la compra. Además, cuando un título activo es vendido en su totalidad ya no se considera como tal.

Finalmente, existe la enumeración catalán, castellano e inglés para indicar que el sistema se puede mostrar en tres idiomas distintos.

Como parte de la ampliación del proyecto, y considerándolo oportuno mostrarlo en el diagrama, se han generado las clases que componen un foro. Un foro está compuesto por temas, y estos últimos por mensajes. La clase mensaje tiene una asociación a sí misma haciendo referencia a que un mensaje puede ser respuesta de otro mensaje. Un usuario registrado al sistema puede publicar mensajes en el foro. Además, los usuarios registrados tienen la opción de enviar emails a otros usuarios en una fecha determinada.

4.4.2 Especificación

En la especificación se muestran todas las clases del modelo conceptual junto a sus atributos.

Clase Usuario
Representa un usuario del sistema

Tabla 4-37: Clase usuario

Clase UsuarioNoRegistrado
Representa a un usuario no registrado al sistema

Tabla 4-38: Clase Usuario NoRegistrado

Clase UsuarioRegistrado	
Representa a un usuario registrado al sistema	
Atributos	**Descripción**
NombreUsuario	Nombre de usuario utilizado en la identificación
Email	Email de contacto del usuario
Password	Contraseña usada para registrarse en el sistema
FechaRegistro	Fecha en la que se registró el usuario *(Usado por Joomla!)*
FechaUltimaVisita	Fecha de la última visita del usuario *(Usado por Joomla!)*

Tabla 4-39: Clase Usuario Registrado

Clase Cliente	
Representa a un cliente que hace uso del sistema	
Atributos	**Descripción**
Nombre	Nombre real del cliente
Apellido1	Primer apellido
Apellido2	Segundo apellido
Dni	DNI o NIE que identifica al cliente
Dirección	Dirección de contacto
CP	Código Postal
Población	Población
País	País
Teléfono	Teléfono de contacto

Tabla 4-40: Clase Cliente

Clase Administrador
Representa a un administrador del sistema

Tabla 4-41: Clase Administrador

Clase MercadoDeValores	
Representa al mercado de valores	
Atributos	**Descripción**
Nombre	Nombre del Mercado de valores (Ejemplo IBEX35)
Descripción	Perfil, historia y compañías que forman el mercado de valores.

Tabla 4-42: Clase MercadoDeValores

Clase Compañía	
Representa a una Compañía de un mercado de valores	
Atributos	**Descripción**
Ticker	Identificador de la compañía para recoger los datos en el mercado de valores
ISIN	Número identificador de la compañía
Nombre	Nombre completo de la compañía
Descripción	Perfil, historia y otra información financiera de la compañía
Fecha_inicio	Fecha en la que se inició en el mercado de valor donde se encuentre
Domicilio	Dirección de contacto de la empresa
Teléfono	Teléfono de contacto
URL	Dirección web de la bolsa de Madrid donde se encuentra información de la compañía
UpToday	Indica si
Publicado	Indica si se puede consultar su información en el sitio web
Capital_permitido	Capital actual permitido en el mercado de valores
Número_acciones	Número de acciones que tiene en el mercado de valores.
País	País de origen de la compañía

Tabla 4-43: Clase Compañía

Clase Acción	
Representa a una acción	
Atributos	**Descripción**
Ticker	Identificador de la compañía

Tabla 4-44: Clase Acción

Clase Fecha	
Representa a un precio	
Atributos	**Descripción**
Fecha	Fecha

Tabla 4-45: Clase Precio

Clase AccionConPrecio	
Representa a una acción con un precio aplicado	
Atributos	**Descripción**
Ticker	Identificador de la compañía
Precio	Valor de la acción
Fecha	Fecha

Tabla 4-46: Clase AccionConPrecio

Clase TítuloActivo	
Representa a un título activo obtenido a partir de la compra de una acción	
Atributos	**Descripción**
NombreUsuario	Nombre de usuario del cliente que ha comprado el título
Ticker	Identificador de la compañía donde se ha obtenido el título
Fecha_compra	Fecha de la compra del título
Precio_compra	Precio de la compra del título
Cantidad_comprada	Cantidad de acciones comprada para este título.

Tabla 4-47: Clase TítuloActivo

Clase TítuloVendido	
Representa a un título vendido	
Atributos	**Descripción**
NombreUsuario	Nombre de usuario del cliente que ha vendido el título
Ticker	Identificador de la compañía del título
Fecha_compra	Fecha de la compra del título
Precio_compra	Precio de la compra del título
Fecha_venta	Fecha de la venta del título
Precio_venta	Precio de la venta del título
Cantidad_vendida	Cantidad de acciones vendidas para este título.

Tabla 4-48: Clase TítuloVendido

Clase Foro	
Representa a un Foro	
Atributos	**Descripción**
Nombre	Identificador del foro (Ejemplo: Foro financiero)

Tabla 4-49: Clase Foro

Clase Noticia	
Representa a una noticia de finanzas	
Atributos	**Descripción**
Título	Título de la noticia
Contenido	Contenido de la noticia
Fuente	Fuente de obtención de la noticia.

Tabla 4-50: Clase Noticia

Clase EnlaceDeInterés	
Representa a un enlace de interés a un sitio web	
Atributos	**Descripción**
Título	Nombre del lugar
Descripción	Breve descripción del sitio web
URL	Dirección URL

Tabla 4-51: Clase Enlace de interés

Clase TemaDelForo	
Representa a un tema del foro	
Atributos	**Descripción**
Título	Título que identifica el tema del foro

Tabla 4-52: Clase TemaDelForo

Clase MensajeDelForo	
Representa a un mensaje del foro	
Atributos	**Descripción**
Título	Título del mensaje escrito
Mensaje	Contenido del mensaje
Fecha	Fecha de publicación del mensaje

Tabla 4-53: Clase MensajeDelForo

Clase Email	
Representa un email	
Atributos	**Descripción**
Compositor	Usuario que envía el email
Destinatario	Usuario que recibe el email
Asunto	Asunto del email
Contenido	Contenido
Fecha	Fecha de envío del email

Tabla 4-54: Clase Email

Capítulo 5

Diseño

5.1 Introducción

La fase de diseño de sistemas software se define como el proceso de aplicar técnicas y principios con el propósito de definir un dispositivo, un proceso o un sistema, al nivel de detalle como para permitir su interpretación y realización física.

La fase de diseño del sistema se puede descomponer en cuatro etapas:

- El diseño de los datos: Se encarga de transformar el modelo de dominio, creado en el análisis, en la estructura de datos necesaria para implementar el software. En la mayoría de proyectos informáticos, esto implica crear un modelo relacional para la base de datos.
- El diseño arquitectónico: Define qué relación hay entre cada uno de los elementos estructurales. En esta etapa se muestra la diferenciación entre las capas de presentación, dominio y datos.
- El diseño estructural: Define la estructura del software utilizado y sobre que hardware se ejecuta.
- El diseño de la interfaz: Define la comunicación del sistema con los usuarios y su funcionamiento interno. Al tratarse de una página web, define qué hay internamente para poder satisfacer las peticiones de los usuarios Se muestra mediante diagramas de actividad o de secuencia.

La importancia del diseño del software se define en una sola palabra: calidad. El diseño es la única manera de materializar con precisión las necesidades del cliente. El diseño es un proceso y un modelado a la vez. Es decir, es el diseñador quien describe todos los aspectos del sistema mediante un conjunto de pasos repetitivos. Durante el transcurso del diseño se evalúa la calidad del desarrollo del sistema con varias revisiones técnicas. Inicialmente, se deben implementar todos los requisitos explícitos del modelo de análisis y se deben acumular los implícitos que el cliente desea. Además debe ser una guía para que los implementadores puedan leer y entender lo que construyen, revisan y mantienen. En general, debe proporcionar una idea completa de lo que es el software.

Todos estos criterios se consiguen aplicando los principios fundamentales de diseño, bajo una metodología sistemática y con revisiones exhaustivas. Cuando se va a diseñar un sistema software hay que tener en cuenta que en esta fase se incluye la elaboración de varios modelos o croquis eligiendo el más apropiado.

Para no extender la documentación del presente proyecto se han creado cuatro subapartados, haciendo referencia a las cuatro etapas del diseño.

- En la primera etapa, se mostrará el diagrama de clases o modelo relacional del sistema. Se indicarán las tablas que se han añadido en la base de datos para hacer referencia a los conceptos (clases) del modelo de dominio.
- En el diseño arquitectónico se explicará qué metodología de capas se ha elegido. Básicamente, se ha tenido que seguir el patrón de diseño de Joomla!.
- En el diseño estructural o diagrama de despliegue se muestra el despliegue de todo el software ejecutado dentro del hardware físico.
- Finalmente, en el diseño de la interfaz se mostrará el diseño de un caso de uso en particular. Lo conveniente en un proyecto software profesional sería diseñar todas las interfaces de los casos de uso. En este caso, al ser yo el diseñador y el implementador, no es necesario un documento que muestre todas las especificaciones técnicas de diseño pudiendo aplicar la implementación al análisis realizado. Además, al tratarse de un componente que se instala en Joomla!, el diseño que adopta el sistema se adapta a la plantilla que se aplique en el sitio web que lo usa.

5.2 Modelo relacional

La aplicación de Joomla! en un sitio web requiere del uso de una base de datos relacional. Cuando un diseñador instala Joomla! se encuentra que se ha creado una estructura de 34 tablas en la dase de datos formada por siete grandes grupos según el contenido de éstas:

- Para el contenido de un sitio web Joomla!
- Para las extensiones:
- Para componentes instalados por defecto
- Para las plantillas almacenadas
- Para el contenido del menú
- Para los usuarios y control de acceso
- Para los logs y estados

En el apéndice D se muestran todas las tablas y las relaciones que hay entre ellas. Es muy interesante, a la vez que importante, saber cómo funciona esta estructura. Cualquier persona que se adentre en el mundo Joomla! debería tener esta base de datos en mente.

Para el almacenamiento de los objetos mencionados en el análisis, hay que crear varias tablas en la base de datos. Toda la información que se presenta a continuación está ilustrada en la ilustración 5-1:

Nótese que en Joomla! todas las tablas empiezan con el prefijo jos_ y tienen la llave primaria llamada *id* para simplificar la implementación.

Primero, se debe almacenar toda la información de un perfil de usuario. Para ello se crea una tabla llamada *jos_cliente* con el contenido de la clase. El atributo *username* actúa de llave foránea a la tabla *jos_users* (ya implementada en Joomla!) para complementar toda la información de un usuario: tipo de usuario, contraseña, registro, etc. De esta forma, la tabla cliente sirve para añadir más propiedades a un usuario. Se podría haber añadido más columnas en la tabla de Joomla! *jos_users,* pero se optó por no darle tantos enlaces a esta tabla [Ver sección *2-Relación entre los bloques* del apéndice D]. A medida que un usuario va rellenando los formularios de compra-venta de acciones, estos datos se introducen en las tablas *jos_tituloactivo* y *jos_titulohistorico.*

Cuando un usuario ha rellenado el formulario de compra, éste título aparece como una entrada en *jos_tituloactivo* indicando el precio, la cantidad y el ticker de la compañía entre algunos valores. La conexión entre las dos tablas se produce mediante la llave foránea *username*. Este mismo usuario ha vendido parte de este título activo y se lo indica al sistema rellenando el formulario de ventas. El sistema almacena la entrada en la tabla *jos_titulohistorico* y actualiza el valor de *cantidad_comprada* de la tabla *jos_tituloactivo*. Si el título activo se ha vendido totalmente, el registro de la instancia queda eliminado de la tabla.

Además, el usuario quiere tener información acerca de los precios en el mercado. El sistema tiene una tabla llamada *jos_datoshistoricos* con todos los valores de las compañías que están almacenadas en el sistema. Para ello, debe tener otra tabla llamada jos_*compania* con los atributos de la compañía. Para relacionar estas dos

tablas se ha añadido el atributo *ticker* que indica el identificador de la compañía en el mercado de valores. Para que el proyecto sea ampliable a otros mercados, además del IBEX 35, se crea una tabla con el nombre *jos_mercadovalores* que da información de dicho mercado. Lógicamente, todas las compañías almacenadas en el sistema, forman parte de un mercado de valores, por lo que tiene un atributo que relaciona ambas tablas.

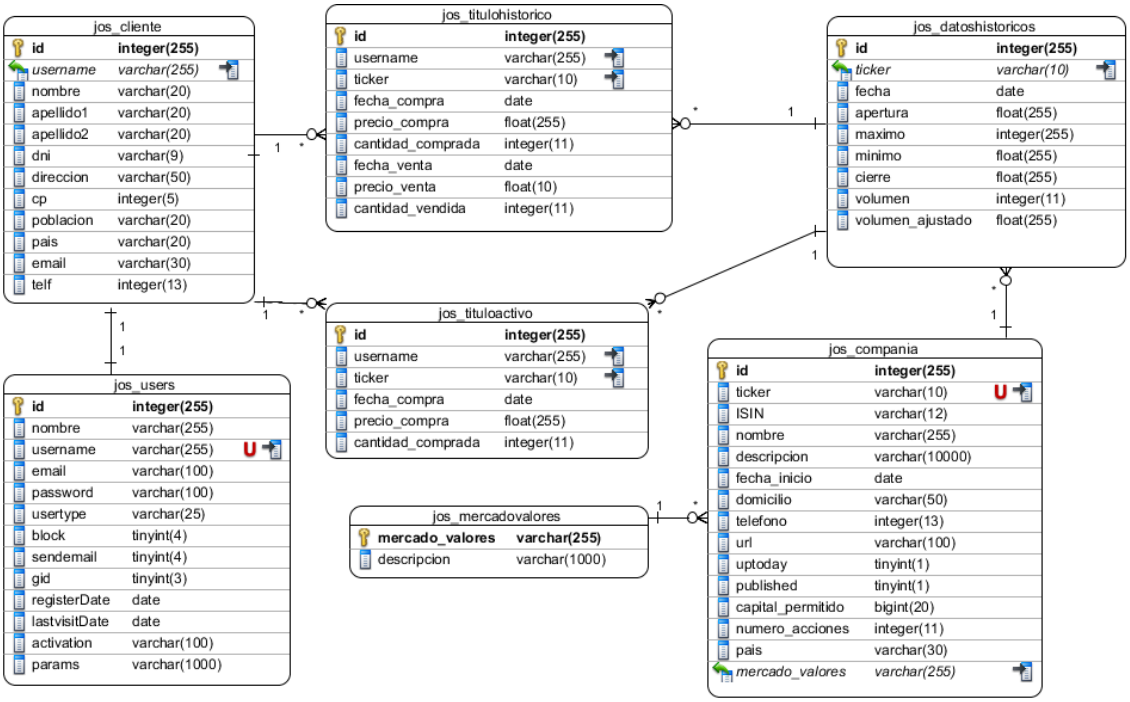


Ilustración 5-1: Modelo relacional

Además, la gestión de enlaces web y de noticias, se hace a través de unas tablas instaladas ya en Joomla!. Éstas son las tablas *jos_weblinks* y *jos_newsfeeds* cuyo funcionamiento es, respectivamente, como sigue:

Cada vez que se quiera añadir un enlace en la página web, se añade un registro en la en la tabla *jos_weblinks* con el nombre del sitio, la URL y una breve descripción. Todos los demás atributos de la tabla son añadidos automáticamente, por el programa. Un administrador puede añadir enlaces a través del *backend* del portal.

El comportamiento es similar para *jos_newsfeeds*. A través del *backend* se van añadiendo portales web de los que se quieren obtener noticias. Hay que indicar el nombre del enlace URL y el número de noticias que se quiere recibir. Automáticamente, Joomla! genera los demás atributos. La siguiente ilustración muestra las dos tablas:

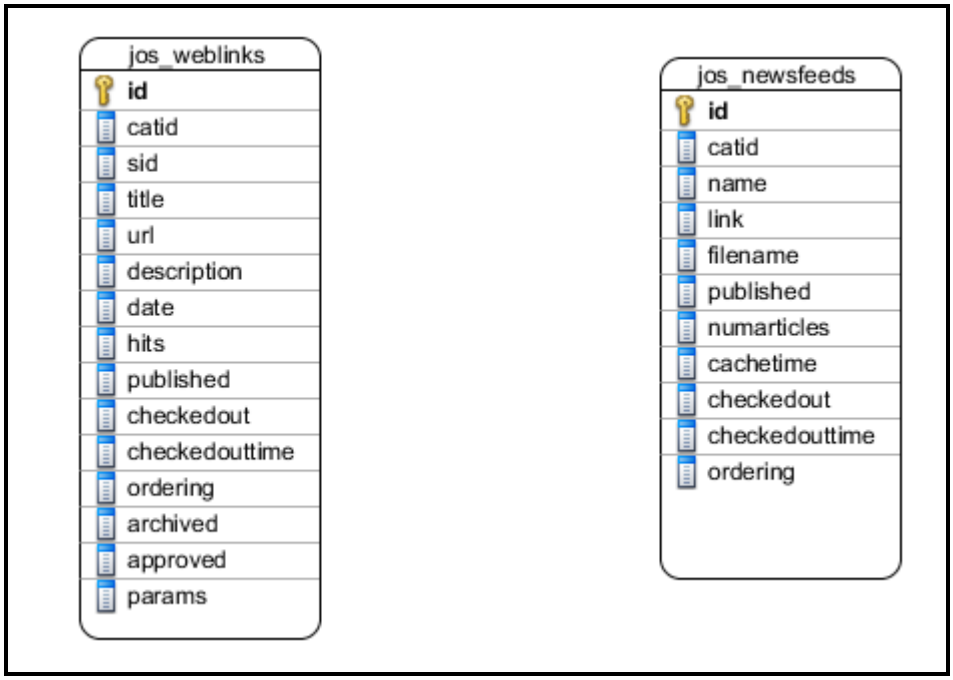


Ilustración 5-2: Tablas para enlaces y noticias

5.3 Diseño arquitectónico

5.3.1 El patrón MVC

La versión 1.5 de Joomla! incluye novedades en la elaboración de componentes, entre las cuales se encuentra la posibilidad de usar el patrón de diseño MVC. Gracias a este patrón se facilita el mantenimiento del componente haciéndolo más fácil de leer por terceras personas y más extensible.

El patrón de diseño MVC se lleva utilizando mucho tiempo en el ámbito del desarrollo web en marcos de trabajo como *Jakarta struts* de apache (java), Java Server Faces de Sun (java) o Symphony (Php).

MVC son las siglas de *Model View Controller*, es decir, Modelo Vista Controlador.

Una aplicación web basada en este patrón separa el código en tres partes bien diferenciadas:

- **El controlador**: el controlador es el punto de entrada de la aplicación web. Constantemente se mantiene a la espera de todas las peticiones, ejecuta la lógica de la aplicación y muestra la vista que sea apropiada para cada caso. Puede ser que haya más de un controlador en un sistema, según como se haya diseñado.
- **El modelo**: El modelo es el encargado de realizar los accesos a la base de datos. Es importante que sea un código genérico y que se pueda reutilizar en otras situaciones de la aplicación. Esto permitirá que cualquier vista pueda usar el modelo. En esta parte nunca se incluirá la lógica, sino que simplemente se encargará de accesos a la base de datos y de validaciones de entrada de datos.
- **La vista**: La vista contiene el código que representará el resultado final en pantalla.

El principal objetivo de usar este patrón, es separar y ordenar el código para que sea más legible y fácil de extender.

En el proyecto se ha considerado lo siguiente:

Existe un Controlador que está a la espera de peticiones de los usuarios. Cuando éste recibe un evento, según la petición solicitada, hace unas funcionalidades u otras. En algunos casos, su función es simplemente mostrar la información de una Vista, mientras que en otros es gestionar los datos recibidos. Además el Controlador puede llamar a cualquier modelo para acceder a la base de datos.

Cabía la posibilidad de diseñar un controlador para cada caso de uso. Esto suponía tener muchos controladores pequeños, con solamente una o dos funciones. La otra alternativa era usar un solo controlador para que lo gestionara todo. En muchos de los casos, esta opción carga demasiado a los controladores. Al tratarse de un proyecto pequeño no genera ninguna complejidad anormal. Ahora bien, si el proyecto aumenta, se puede considerar la posibilidad de tener varios controladores.

En definitiva, cada caso de uso se transforma en una función del controlador. Además, los casos de uso son representados mediante una vista. Se han añadido clases Modelo para los casos de uso donde se requiera una gestión de datos; por ejemplo para una consulta o un formulario.

La siguiente ilustración muestra la estructura de un modelo MVC:

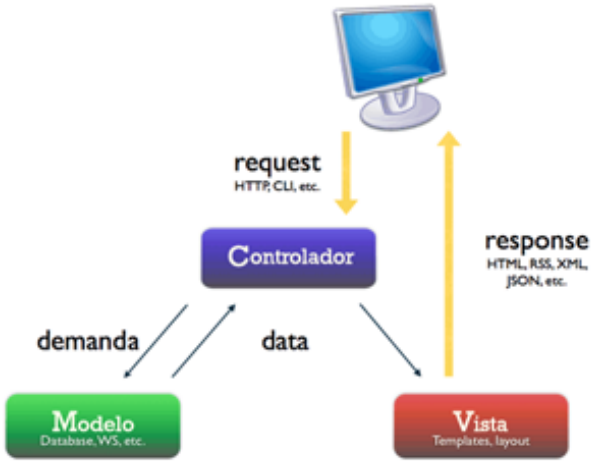


Ilustración 5-3: Modelo MVC

Usar MVC implica aplicar el patrón por capas: Se organiza la estructura lógica en capas separadas, de responsabilidades distintas y relacionadas. Las capas más bajas son servicios generales de bajo nivel, como la gestión de bases de datos, y las más altas son más cercanas al usuario final, como las interfaces.

El patrón por capas se divide en tres grandes bloques:

La capa más alta llamada capa de presentación o vista: Es el resultado final que ve el usuario, por lo que tiene que ser amigable, entendible y fácil de usar. Esta capa solamente se comunica con la capa intermedia o capa de negocio. En este proyecto, las clases Vista del patrón MVC son las encargadas de mostrar la información al usuario.

La capa intermedia o de negocio: Aquí se establecen todas las reglas a cumplir. Esta capa se comunica con la capa de presentación para recibir solicitudes y mostrar resultados, y con la capa de datos, para almacenar o recuperar datos de la BD. En el proyecto el encargado de gestionar toda la capa intermedia es el Controlador.

La capa baja o de datos. En ella residen los gestores de base de datos encargados de almacenar o recuperar toda la información necesaria. Una ventaja de Joomla! es que ya existen varias funciones implementadas, que se encargan de la gestión de los datos. A pesar de esto, en el proyecto se han implementado Modelos para obtener consultas más complejas o personalizadas.

5.4 Diseño estructural o diagrama de despliegue

El diagrama de despliegue muestra como se relacionan entre sí el sistema hardware y el sistema software.

En diagrama de despliegue del componente implementado en el proyecto está compuesto básicamente por dos elementos principales tal como sucede en cualquier sitio web:

Cliente: Terminal utilizado por el usuario para consultar la aplicación. En estos tipos de proyectos se trata de navegadores web.

Servidor: Terminal donde se encuentra Joomla! y sus componentes instalados. Entre ellos se encuentra el componente implementado (llamado PFC). El entorno de ejecución utilizado es WAMP.

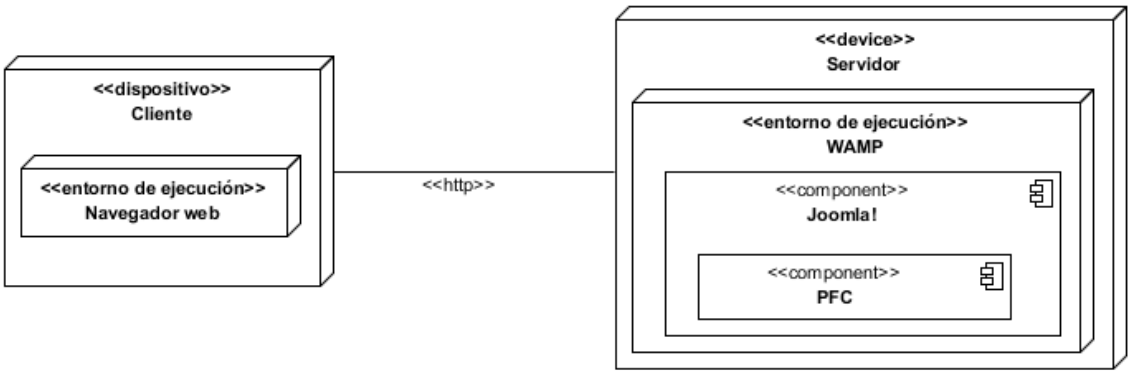


Ilustración 5-4: Diagrama de despliegue

5.5 Diseño de la interfaz

Cuando un componente de Joomla! se instala en un sitio web, éste debe adaptarse lo máximo posible al diseño de la interfaz que se utilice. Cada plantilla Joomla! contiene unos archivos CSS que permiten el diseño automático del componente. Cuando un diseñador quiere crear una nueva plantilla en Joomla!, debe modificar las clases e identificadores CSS de dichos archivos.

Esto no sucede con los componentes: Se puede asignar una clase o un identificador a la mayoría de etiquetas HTML que se usen en la Vista del componente. Eso permite que cada sección de la Vista reciba el formato de la plantilla.

Por ejemplo, asignar un título a la vista con la sentencia * Mi título * permite que dicho título adapte el formato de la entrada *div.componentheading* del archivo CSS.

En la página http://joomlatp.com/joomla-css/joomla-1.5-css-files-list.html se muestran todas las clases e identificadores que están en uso en Joomla! 1.5.

Por lo tanto, diseñar la interfaz de un componente significa asignar correctamente los identificadores y las clases a las etiquetas HTML.

Para no extender el presente apartado, se ha analizado el diseño de los casos de uso del *backend*. En particular, se ha elegido el caso Modificar_compañía_IBEX del *backend*, que a su vez incluye Consultar_lista_compañías_IBEX y Consultar_compañía_IBEX.

5.5.1 Diseño de los casos de uso del *backend*

El diseño del caso de uso Modificar_compañía_IBEX requiere que antes se vea la lista de compañías. La siguiente ilustración muestra la vista de éste último caso de uso:

Ticker	ISIN	Nombre	Fecha	Dirección	Teléfono	URL	Capital permitido	Número de acciones	País	Up Toda
ABG.MC	ES0105200416	ABENGOA,S.A.	2001-07-26	CL SUNP-GU (PALMAS ALTAS) 1 C.P. 41012 SEVILLA	954937111	http://www.bolsamadrid.es/comun/fichaemp/fichavalor.asp?isin=ES0105200416	22617420	90470	España	1
ABE.MC	ES0111845014	ABERTIS INFRAESTRUCTURAS,S.A.	2000-01-03	AV PARC LOGISTIC 12 C.P. 08040 BARCELONA	932305000	http://www.bolsamadrid.es/comun/fichaemp/fichavalor.asp?isin=ES0111845014	2111536524	703846	España	1
ANA.MC	ES0125220311	ACCIONA,S.A.	2003-01-01	CL AVENIDA DE EUROPA 18 C.P. 28943 ALCOBENDAS	916632850	http://www.bolsamadrid.es/comun/fichaemp/fichavalor.asp?isin=ES0125220311	63550000	63550	España	1

Ilustración 5-5: Diseño caso de uso: Pantalla lista de compañías

Los elementos de la vista son los siguientes:

- Una cabecera con el título y los botones para gestionar las compañías. Estos botones tienen que mostrar gráficamente cuál es la acción que se va a aplicar. Joomla! muestra por defecto las imágenes de la ilustración para los botones publicar, despublicar, editar, borrar y nuevo.
- Una lista con todas las compañías registradas en el sistema. La primera columna de la lista tiene que ser una "caja selectora" para poder indicar al sistema qué compañía se quiere publicar, despublicar, editar o borrar.

El formato utilizado para representar las cosas, es el usado por Joomla! por defecto.

Para modificar una compañía se debe seleccionar la caja selectora de la compañía y clicar el botón editar. A continuación aparecerá la Vista del caso de uso Modificar_compañía_IBEX:

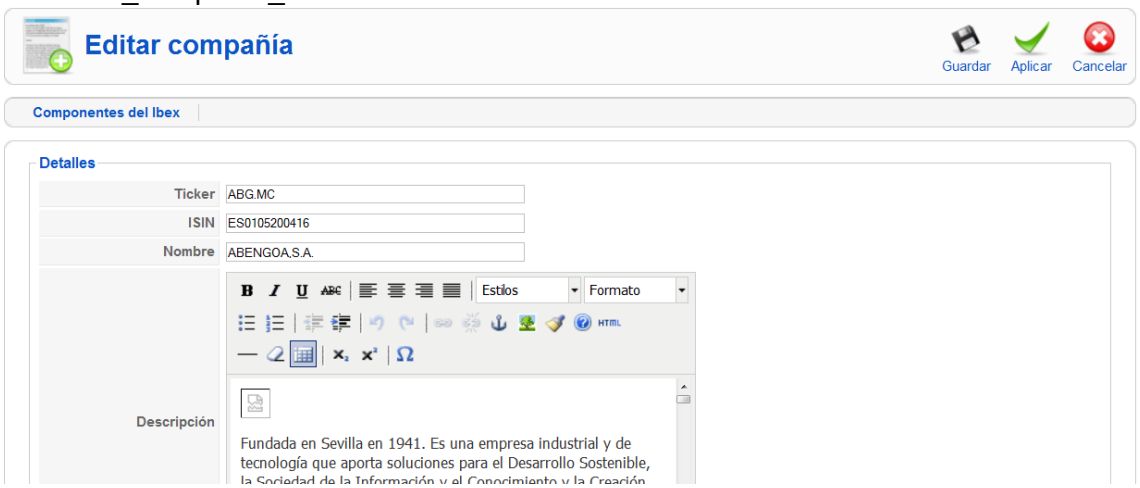


Ilustración 5-6: Diseño caso de uso: Pantalla formulario compañía

Los elementos de la vista son los siguientes:

- Una cabecera con el título y los botones guardar, aplicar o cancelar para aplicar dichas acciones.
- Un formulario donde se pueda introducir los datos de la compañía. Aquí tienen que aparecer los datos de la compañía almacenados en el sistema mediante:
  - Un campo de texto para los atributos *Ticker, ISIN, Nombre, Dirección, Teléfono, URL, Capital Permitido, Número de acciones y País*.
  - Un editor de textos para el atributo *descripción*.
  - Un calendario para el atributo *entrada en bolsa*.
  - Un selector Sí/No para los atributos *Uptoday y publicado.*

Análisis general:

Inicialmente, se tiene que hacer una consulta a la base de datos para mostrar el listado de todas las compañías del IBEX 35. Cuando el usuario decide editar una compañía, el sistema tiene que comprobar que se haya seleccionado alguna compañía. Si no es así, muestra un mensaje de aviso y vuelve a mostrar el listado. Si había una selección, el sistema tiene que mostrar el formulario de la figura anterior, con los datos de la compañía. Cuando el usuario decide, aplicar o guardar los cambios, el sistema tiene que comprobar los datos introducidos y registrarlos si son correctos. Mientras que al usar *aplicar* se sigue en la misma vista, al *guardar o cancelar* se vuelve a mostrar la vista de la lista de las compañías.

El siguiente diagrama de actividades muestra las acciones que se realizan al crear o modificar una compañía:

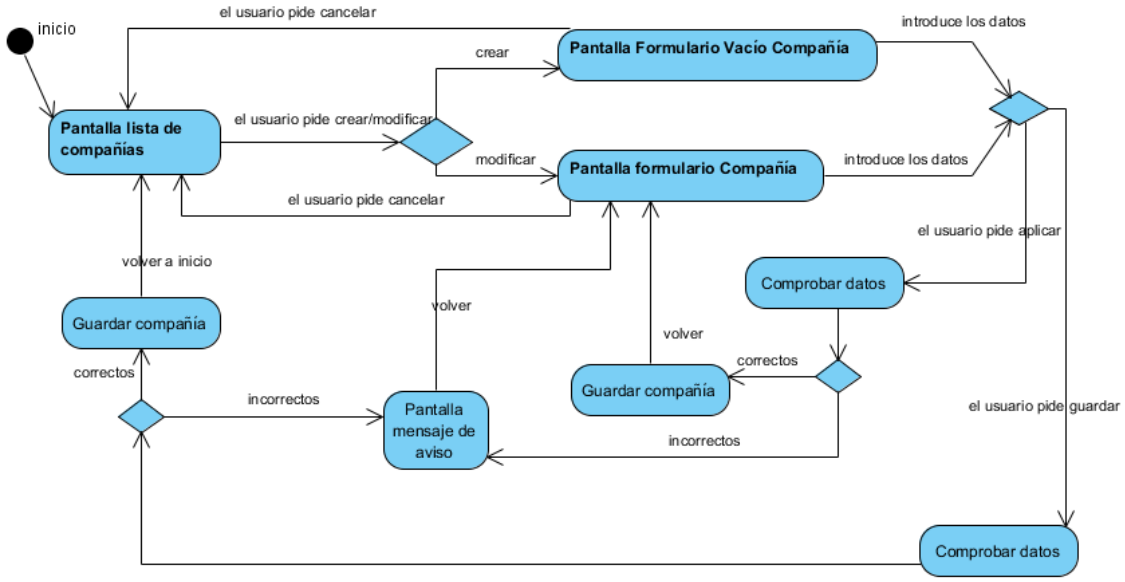


Ilustración 5-7: Diagrama de actividades - Crear/Modificar compañía

Eliminar, publicar o despublicar una compañía sigue la secuencia de eventos mostrada en la siguiente ilustración:

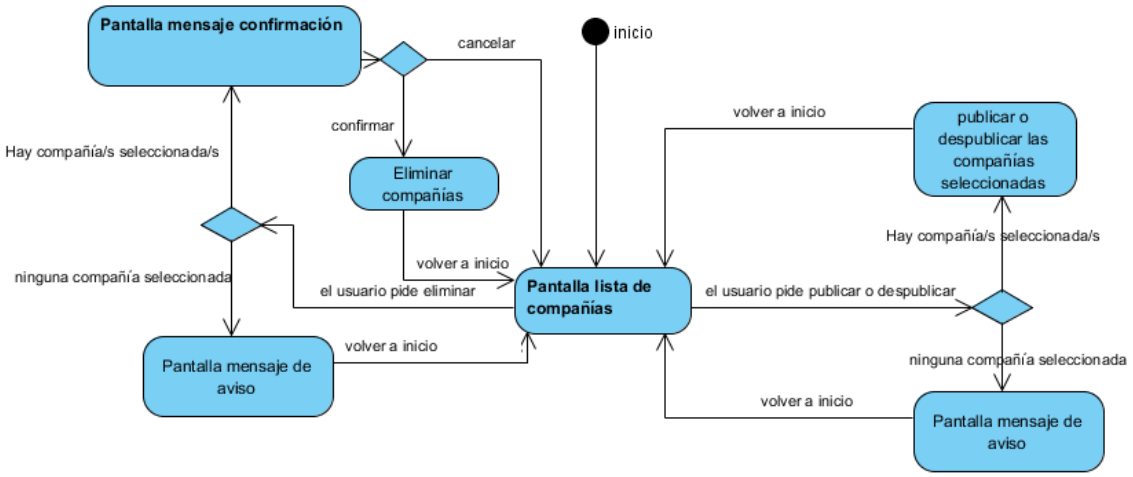


Ilustración 5-8: Diagrama de actividades - Eliminar/Publicar/Despublicar compañía

Capítulo 6

Implementación

6.1 Introducción

En la presente sección se muestra cómo se implementa un componente con la versión 1.5.x de Joomla!. Antes de ver el código escrito, se presenta el entorno de programación utilizado.

6.2 Entorno de programación: lenguajes y tecnologías.

6.2.1 HTML y DHTML

HTML es una pequeña aplicación del SGML *(Standard Generalized Markup Language),* un sistema para definir tipos de documentos estructurados y lenguajes de marcas que representan esos mismos documentos.

Cuando un usuario principal maneja Internet, experimenta tres fases diferentes: Inicialmente, solamente conoce unas pocas páginas, luego encuentra buscadores que facilitan la búsqueda de la información y finalmente descubre que además de leer información el mismo la puede publicar. Lo que un usuario desconoce es cómo se consigue esta interacción de información entre ordenadores y personas. Para que varias personas se comuniquen es necesario que hablen el mismo idioma. EL lenguaje que usan los ordenadores conectados a Internet es HTML.

HTML, siglas de Hyper Markup Language (Lenguaje de marcación de Hipertexto), es el lenguaje de marcado o marcas predominante para la creación de sitios web. Se usa para describir la estructura y el contenido en forma de texto y para completar el texto con objetos tales como imágenes. HTML usa la estructura de “etiquetas” escritas entre corchetes angulares (<,>).

Fue creado el 1986 por el físico Tim Berners-Lee que unió dos herramientas: el concepto Hipertexto, conocido como *link* o enlace, el SGML que sirve para colocar etiquetas o marcas en un texto para indicar como debe ser visto. No fue hasta el 1991 cuando se encuentra la primera publicación en Internet de HTML. Fue un documento creado por el mismo Tim Berners-Lee llamado *HTML Tags* (Etiquetas HTML). Simplemente describe 22 elementos del lenguaje para comprender la simplicidad y el diseño de HTML. EL autor consideraba HTML una ampliación de SGML, pero no se definió como tal hasta el año 1993 cuando la IETF *(Internet Engineering Task Force)* publicó una primera proposición oficial para la especificación de HTML.

El lenguaje HTML es un simple documento, que puede ser creado y editado con cualquier editor de textos básico como el Bloc de Notas de Windows o el Gedit de Linux. También existen programas especializados en la creación de páginas web con la característica WYSIWYG (What You See Is What You Get, es decir, lo que ves es lo que obtienes) que permiten ver a tiempo real lo que se está editando. Simplemente facilitan al programador la creación de documentos HTML de tal forma que a veces ni hace falta ser un especialista del lenguaje. Algunos ejemplos son Macromedia Dreamweaver o Microsoft Front Page.

Estos documentos son mostrados por los navegadores de páginas web en Internet como Mozilla Firefox, Internet Explorer o Opera. Estos reconocen las etiquetas y las interpretan mostrando el contenido del documento en pantalla.

Una extensión de este lenguaje es el HTML dinámico o DHTML (*Dynamic HTML).* Designa el conjunto de técnicas usadas para crear sitios web dinámicos combinando el lenguaje HTML, con un lenguaje interpretado en el lado del cliente como JavaScript, el lenguaje de hojas de estilo en cascada (CSS) y la jerarquía de objetos DOM.

Una página DHTML es simplemente cualquier página en la que los scripts cambian su contenido a medida que el usuario interactúa con ésta. Esto permite que una misma página estática pueda ser vista de distintas formas según la petición de los usuarios. No se debe confundir DHTML con páginas creadas con la ejecución de algún lenguaje de programación en el servidor de la página como ASP, PHP o PERL. Es decir, en una página DHTML, una vez ésta ha sido cargada completamente por el cliente, se ejecuta un código (generalmente en Javascript) que tiene efectos en los valores del lenguaje de definición de la presentación. Así se logra una modificación en la información presentada o en el aspecto visual mientras el usuario está viendo la página.

6.2.2 CSS

Una hoja de estilo en cascada o CSS (*Cascading Style Sheets*), es un lenguaje usado para la definición de la presentación de un documento estructurado escrito en HTML o XML. El *World Wide Web Consortium*, conocido como W3C, es el encargado de generar una especificación estándar de las hojas de estilo para que los navegadores usen la misma nomenclatura.

Esta idea, permite separar el contenido, que se encuentra en los documentos HTML, de la presentación estética, encontrada en hojas CSS. Gracias a ello, el estilo de un documento web es más fácil de modificar y de reusar.

Por ejemplo, cuando se usa un elemento <table> en un documento HTML no se debe proporcionar información de cómo será visualizada la tabla, sino que solamente convendría indicar la estructura de ésta. Toda la información de color, fuente, alineación de texto y otras características debe de ser especificada en la hoja de estilo CSS. CSS proporciona tres vías distintas para aplicar las reglas de estilo:

1- Hojas de estilo externas: Todo el contenido del estilo se almacena en un archivo distinto al código HTML. Es la manera más potente de programar porque separa completamente las reglas de formateo para la página HTML.

2- Hojas de estilo internas: Todo el contenido del estilo se almacena en el mismo documento dentro de la etiqueta <head>.

3- Hojas de estilo en línea (inline): El contenido del estilo se almacena dentro de cada etiqueta HTML mediante el atributo “style”.

Las principales ventajas de usar CSS son:

- Control centralizado de la presentación de un sitio web. Agiliza la actualización del mismo.
- Los navegadores permiten a los usuarios crear su propia hoja de estilo local que será aplicada en un sitio web remoto. Aumenta la accesibilidad.
- Una página puede disponer de varias hojas de estilo según si se quiere consultar a través de un ordenador, de un móvil o de cualquier otro dispositivo.
- La separación de código permite obtener documentos mucho más pequeños. Esto permite una lectura y comprensión más fácil del código.

6.2.3 PHP

PHP o PHP *Hypertext Pre -processor*, es un lenguaje de código abierto interpretado en alto nivel, pensado para desarrolladores Web y que puede ser incluido en documentos HTML. La mayoría de su sintaxis es similar a lenguajes de alto nivel como Java o C y es muy fácil de aprender. Aunque PHP es muy potente y permite hacer muchas cosas, el principal objetivo de este lenguaje es la creación de páginas Web dinámicas de una manera rápida y sencilla.

Inicialmente fue diseñado en Perl, con base en la escritura de un grupo de CGI (*Common GateWay Interface)* binarios escritos en lenguaje C, por el danés-canadiense Rasmus Lerdorf en 1994 con el fin de mostrar su Currículum Vítae y guardar ciertos datos que su página web recibía. El año 1995 fue publicado el "*Personal Home Page Tools*" después de que Lerdorf lo combinara con su propio *Form Interpreter* PHP/FI. EL PHP Tools permitía la creación de páginas personales de manera inmediata. El año 1993, dos israelíes del Technion, Zeev Suraski y Andi Gutmans, reescribieron el código del danés para crear la base PHP3, cambiando el nombre del lenguaje a la forma actual. En el año 1998 se publicó en Internet y comenzaron todas las experimentaciones públicas. Esto ha permitido que se creen nuevas versión del lenguaje llegando actualmente a la versión PHP5. La última versión encontrada es PHP 5.3.2 del 4 de marzo de 2010 donde se amplían mejoras en el ámbito de seguridad. Está previsto que aparezca la versión PHP6 que permitirá la mejora de rendimiento y de seguridad, y tendrá un mejor soporte para la Programación Orientada a Objetos, MyQSL y XML.

Existen tres campos usados en los scripts PHP:

- **Scripts en la parte del servidor:** Es el más usual para la implementación de PHP. Se necesita un parseador PHP, un servidor Web y un navegador. El resultado del programa PHP se obtiene a través del navegador conectando con un servidor Web.
- **Scripts en línea de comandos:** El Script es creado y usado sin la necesidad de un servidor web o navegador. Solamente con el parseador PHP se pueden ejecutar los scripts desde Cron en Linux o el Planificador de tareas en Windows. Suelen ser usados para tareas simples de procesado de texto.
- **Escribir aplicaciones gráficas clientes**: PHP no es considerado un buen lenguaje para la creación de gráficas. Pero hay herramientas que lo permiten. PHP-GTK es una extensión de PHP que permite escribir dichos programas.

PHP puede ser usado desde cualquier Sistema Operativo del mercado y además soporta la mayoría de servidores web mediante los módulos implementados. Esto permite una gran usabilidad de este lenguaje de programación.

Actualmente, brinda la posibilidad de usar programación Orientada a Objetos (POO). Aunque aún no se han implementado todas las características estándares de la programación Orientada a Objetos, PHP se parece cada vez más a los lenguajes que usan este tipo de programación. En el proyecto se ha optado por usar este tipo de programación ya que todos los componentes y módulos de Joomla! lo usan. Gracias a POO un sitio web es más fácil de leer, comprender, mantener y reutilizar.

Entre las múltiples características de PHP cabe destacar el soporte para la gran cantidad de base de datos. Permite una fácil conexión con la mayoría de motores de base de datos que se usan en la actualidad.

Para enumerar todas las características de PHP se necesitaría un libro entero como el *PHP and MYSQL Development [W09].*

6.2.4 JavaScript

Javascript es un lenguaje de scripting orientado a objetos, usado principalmente para acceder a objetos en aplicaciones. Se usa integrado en un navegador web permitiendo el desarrollo de interfaces de usuario mejoradas y páginas web dinámicas.

Javascript es un dialecto de ECMAScript y se caracteriza por ser un lenguaje basado en prototipos, con entrada dinámica y con funciones de primera clase. Al tener influencia de varios lenguajes de programación, tiene un diseño similar a los lenguajes Java y C.

El lenguaje fue inventado por Brendan Eich en la empresa *Netscape Communications*; empresa que desarrolló los primeros navegadores web comerciales. Apareció por primera vez en el producto de Netscape llamado *Netscape Navigator 2.0.*

Inicialmente los autores lo llamaron Mocha y más tarde LiveScript. El 4 de diciembre de 1995 fue rebautizado con el nombre de Javascript en un anuncio conjunto entre Sun Microsystems y Netscape.

Tradicionalmente, se usaba en páginas web HTML, para el uso de operaciones en el marco de la aplicación cliente y sin acceder a funciones del servidor. Javascript se ejecuta en el agente de usuario.

Javascript puede incluirse en cualquier documento y es compatible con HTML en el navegador del cliente, ya sea PHP, ASP, JSP y SVG. Incluir código directamente en una estructura HTML es una práctica invasiva y no recomendada. El método correcto que define la W3C es incluir Javascript como un archivo externo, tanto por cuestiones de accesibilidad, como práctica y velocidad en la navegación.

A pesar de todo, los clientes Web, tales como Netscape Navigator/Communicator o Microsoft Internet Explorer interpretan sentencias JavaScript colocadas en un documento HTML. Cuando un cliente web solicita una página de este tipo, el servidor envía por la red el contenido completo del documento donde se incluye el código HTML y las sentencias Javascript. El cliente lee la página de forma secuencial, representando visualmente el código HTML y ejecutando las sentencias Javascript. Las sentencias Javascript colocadas en una página Web pueden dar respuestas a eventos de usuario como el clic de un botón del ratón, la entrada de datos en un formulario y la navegación por la página.

Lo más importante de Javascript es que se puede trabajar sin la necesidad de transmitir datos al servidor.

6.2.5 Ajax

El término Ajax es un acrónimo de *Asyncronuous JavaScript + XML* traducido como Javascript asíncrono + XML.

Ajax es una técnica de desarrollo Web que permite crear aplicaciones más rápidas y más cómodas para el usuario. A través de esta técnica, el cliente puede interactuar con el servidor de manera asíncrona, sin la necesidad de recargar la página cuando se actualiza el contenido de algunas partes de ésta.

Ajax es una combinación de cuatro tecnologías ya existentes:

- XHTML y CSS, usadas para la presentación del contenido.
- DOM, para la manipulación e interacción dinámica de la presentación.
- XML, XSLT y JSON, para el intercambio y la manipulación de la información del contenido.
- XMLHttpRequest, para el envío asíncrono de la información.

El lenguaje JavaScript es usado para unificar las anteriores tecnologías.

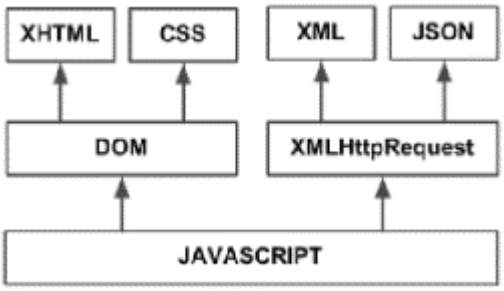


Ilustración 6-1: Tecnologías usadas en Ajax

A pesar que el término Ajax fuera creado en el 2005, la historia de las tecnologías que permiten Ajax se remonta una década antes con la iniciativa de Microsoft en el desarrollo de Scripting Remoto.

Sin embargo, la carga asíncrona de una parte de la página, sin la recarga total de ésta, se creó el año 1996 con el elemento *iframe* para IE 3 y el 1997 con el elemento *layer* para Netscape 4.

En 1998, se introdujo el *Microsoft's Remote Scripting*, una manera más elegante que iframe para el envió asíncrono de la información a través de un applet Java y JavaScript. Esta técnica funcionó en ambos navegadores, IE4 y Netscape Navigator 4. Microsoft la utilizó en el Outlook Web Access provisto con la versión 2000 de Microsoft Exchange Server.

Los desarrolladores web, crearon una gama de técnicas scripting para conseguir los mismos resultados en distintos navegadores. Se llegó al año 2002, cuando se realizó una modificación por parte de la comunidad de usuarios al *Microsoft's Remote Scripting* para reemplazar el applet Java por XMLHttpRequest. Desde entonces, XMLHttpRequest está implementado en la mayoría de los navegadores.

Una aplicación AJAX sigue unos pasos en su funcionamiento:

1. El cliente produce algún evento, como el clic del mouse o la introducción de un dato, a través del navegador.
2. Dicho evento es gestionado con JavaScript y es enviado al servidor asíncronamente mediante XMLHttpRequest.
3. El servidor procesa la petición y devuelve el resultado asíncronamente con XMLHttpResponse.
4. El resultado es gestionado con JavaScript para mostrar la información en las secciones de la página deseada.
5. Sin la necesidad de la recarga de la página, este ciclo comienza de nuevo cuando un usuario produzca un evento.

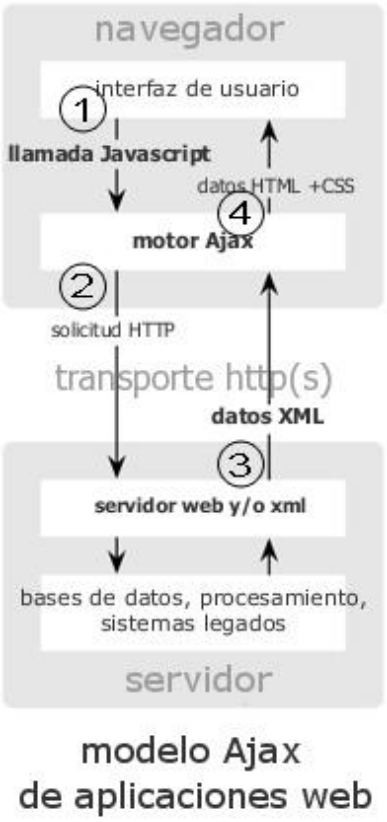


Ilustración 6-2: Funcionamiento de Ajax

En la sección Añadir AJAX al componente se explica cómo se usa Ajax en un componente Joomla!

6.2.6 SQL

SQL no es el acrónimo de *Structure Query Language* como antiguamente se ha dicho ya que el ISO lo define oficialmente como *Database Language SQL.* No es un lenguaje estructurado *"Structured"* ya que se puede usar en bloques o procedimientos, ni solamente sirve para consultas "*Query".*

Es un lenguaje formal declarativo para manipular información en una base de datos.

El año 1970 E.F.Codd propuso el modelo relacional y asociado a éste un sublenguaje de acceso a los datos basado en el cálculo de predicados. Basándose en la idea de Codd, IBM definió el lenguaje SEQUEL (*Structured English QUEry Language)* que se integró en el Sistema de Gestores de Base de Datos (SGBD) experimental *System R* desarrollado también por la misma empresa.

EL año 1979 se introdujo por primera vez SQL en un programa comercial a través de Oracle. A lo largo de los siguientes años SEQUEL pasó a ser un predecesor de una versión mejorada de SQL. Esta mejora supuso que SQL fuera el lenguaje por excelencia de todos los SGBD y pasara a ser estandarizada por el ANSI el año 1986 con el nombre de SQL-86 o SQL-1. Al siguiente año este estándar fue adoptado por ISO.

Desde entonces se han ido ampliando las funcionalidades del lenguaje llegando a la versión SQL 2008.

SQL es un lenguaje de acceso a bases de datos que explota la flexibilidad y potencia de los sistemas relacionales permitiendo una gran variedad de operaciones.

Es un lenguaje declarativo de "alto nivel" o "de no procedimiento". Gracias a su fuerte base teórica y su orientación al manejo de conjuntos de registros, y no a registros individuales, permite una alta productividad en codificación y la orientación a objetos. De esta forma una sola sentencia equivale a uno o más programas usados en un lenguaje de bajo nivel orientado a registros.

SQL se utiliza para gestionar todas las funciones que un sistema gestor de base de datos proporciona a sus usuarios, incluyendo definición, recuperación, manipulación e integración de datos, control de acceso y distribución de datos entre usuarios.

6.2.7 HighCharts

HighCharts es una librería de gráficas escrita en JavaScript que ofrece una forma sencilla de añadir gráficas interactivas a una página web. HighCharts actualmente soporta gráficas de líneas, áreas, columnas, barras y circulares [HIGHCHARTS].

Funciona en todos los navegadores incluyendo el iPhone/iPad y el Internet Explorer desde la versión 6. Los navegadores estándares usan el elemento Canvas y en algunos casos SVG (*Scalable Vector Graphics)* para el *"rendering"* de las gráficas. En Internet Explorer las gráficas son dibujadas usando VML (*Vector Markup Language)*.

La librería HighCharts se puede usar para páginas web no comerciales como escuelas u organizaciones no gubernamentales. Para aun uso comercial es necesario comunicarse con el autor de la librería.

Highcharts está basado en las tecnologías de los navegadores nativos por lo que no requieres *plugins* del sitio cliente como Flash o Java. Además, no es necesario instalar nada en el servidor. Para ejecutar la librería se requieren tres archivos JavaScript: EL núcleo highcharts.js, un emulador para Internet Explorer y el *framework* jQuery o MooTools. Por lo tanto, no es necesaria una implementación en PHP o ASP.Net.

6.3 Implementación del componente

Esta sección es un tutorial para aprender a implementar un componente desde el inicio. Se ha añadido el código necesario para seguir la explicación con más facilidad. Si un lector no está interesado en implementar un componente, no es necesario que lea los códigos detalladamente. Para entender la explicación se requieren conocimientos básicos de PHP y de HTML.

Asumiremos que ya se ha instalado Joomla! correctamente tal como se explica en el apéndice A.

Todos los archivos se encuentran en el directorio raíz *pfc* del servidor WAMP. El directorio donde se instala Joomla! puede llamarse como desee eligiéndose el nombre durante el proceso de instalación.

Se mostrará una pequeña funcionalidad del proyecto indicando los pasos a seguir para facilitar su comprensión. Concretamente, se quiere hacer un componente que muestre una lista con todas las compañías del IBEX 35 y que, al consultarlas, se muestre por pantalla información detallada de la compañía elegida.

Para ello, hay que implementar dos partes:

- Una para poder crear, editar y borrar las compañías del IBEX 35 desde el *backend*.
- Otra para que se muestren los resultados en el *frontend* del sitio web.

A continuación, se indican los pasos a seguir en la implementación.

6.3.1 Creación de la estructura de ficheros del componente

El primer paso consiste en la creación de la estructura de ficheros para el componente. Como se ha explicado en la sección de Joomla! de la documentación [Ver sección 3.3.3-El *Frontend* y El *Backend*] Joomla! contiene dos partes bien diferenciadas, el *frontend* y el *backend. P*ara implementar el componente hay que tener en cuenta dichas partes:

1 - Dentro del directorio **pfc->administrator->components** se crea una carpeta llamada **com_finanzas** que llevará todo el código para gestionar el contenido del *backend*. Para que Joomla! localice y ejecute la componente correctamente es imprescindible que lleve el prefijo **com_** y que se siga la estructura MVC ilustrada en el siguiente diagrama:

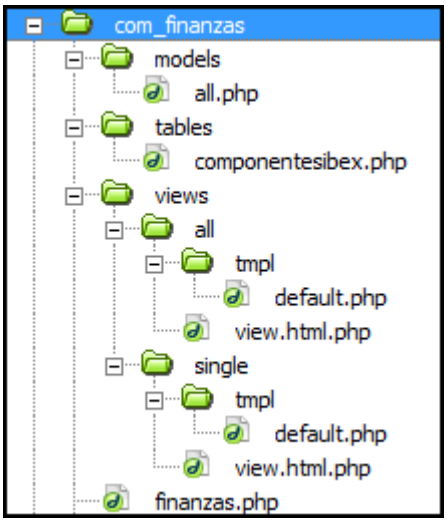


Ilustración 6-3: Estructura de ficheros del *backend*

La carpeta **com_finanzas** contiene tres carpetas y un archivo: las tres carpetas son ***models***, ***views*** y ***tables*** y se encargan de separar el código para las tareas de los modelos, las vistas y las tablas de la BD; El archivo **finanzas.php** es el controlador del componente.

Dentro de *views* se encuentran dos carpetas. La primera, llamada *all*, contiene el código para mostrar todas las compañías del IBEX 35 en una tabla. La segunda, llamada *single,* mostrará la información de una compañía en particular. Los archivos *view.html.php* son los que hacen llamadas a los modelos para obtener información de la BD y enviarla al archivo *default.php.* Este último archivo contiene el código HTML para mostrar el contenido.

En *tables* se incluye el archivo *companiasibex.php* cuya función es crear un objeto que representa a una compañía del IBEX 35.

2- Dentro de la carpeta **pfc->components** se crea nuevamente un directorio llamado **com_finanzas** donde se guardan los ficheros para el *frontend*.

La estructura que sigue el directorio se ilustra en el siguiente diagrama:

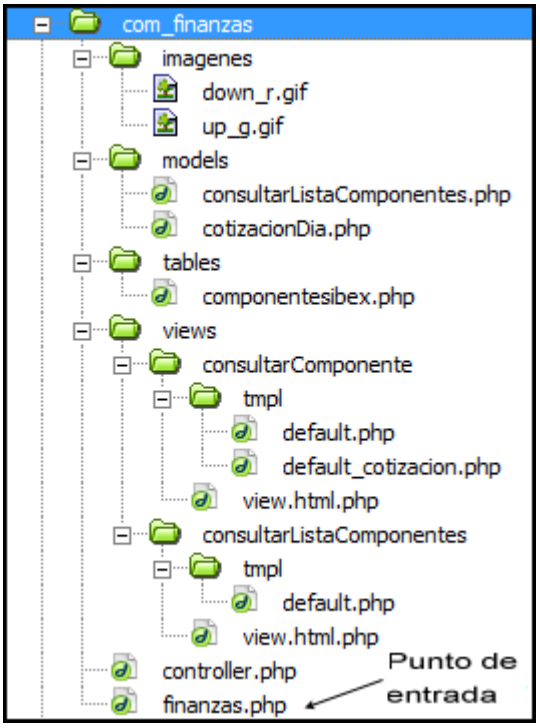


Ilustración 6-4: Estructura de ficheros del *frontend*

La carpeta **com_finanzas** contiene tres carpetas y dos archivos: las tres carpetas son ***models***, ***views*** y ***tables*** encargadas de separar el código para las tareas de los modelos, las vistas y las tablas de la BD; El archivo *controller.php* es el controlador del componente; y el archivo *finanzas.php* es el punto de entrada de la aplicación cuya función es crear el controlador y delegarle tareas.

Para el *frontend* hay una nueva carpeta **imágenes** donde se encuentran dos imágenes mostradas en las vistas. Simplemente son dos flechas que indican si el valor de una compañía está en números positivos o negativos.

6.3.2 Registro del componente

El siguiente paso consiste en registrar el componente en la BD. Para ello, se introduce en el navegador http://localhost/Phpmyadmin/index.php?db=pfc para situarse en la BD. A continuación se ejecuta la siguiente sentencia en la pestaña SQL del programa PhpMyAdmin:

```
INSERT INTO jos_components (name, link, admin_menu_link,
admin_menu_alt, 'option', admin_menu_img, params)
VALUES ('Portafolio de finanzas', 'option=com_finanzas',
'option=com_finanzas', '', 'com_finanzas',
'js/ThemeOffice/component.png', '');
```

Código 6-1: Inserción del componente en la BD

Una vez registrado, el componente aparece en el menú **Componentes** del *backend* aunque aún no sé puede utilizar al no haberse implementado:

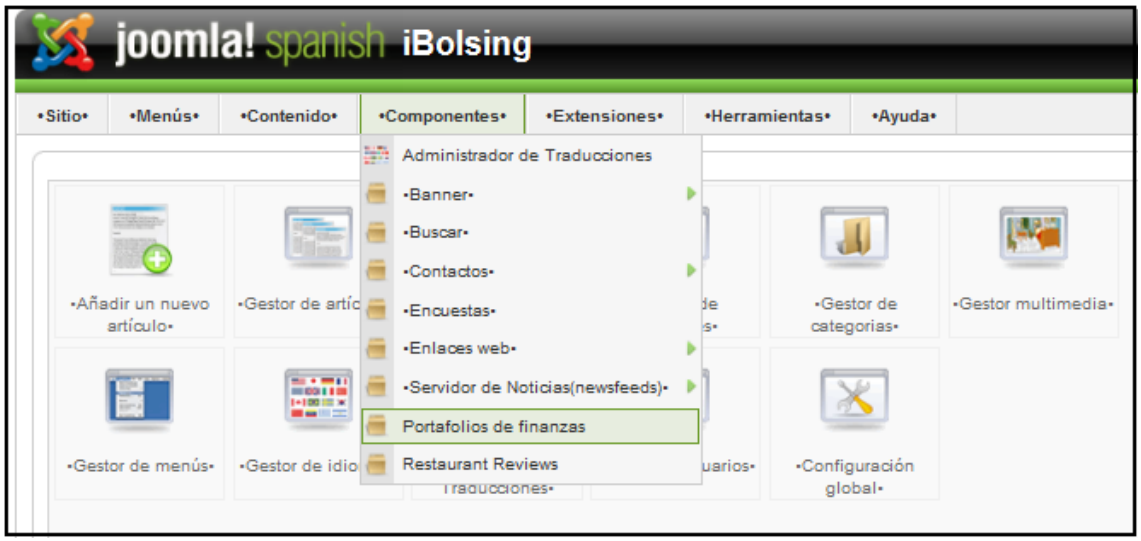


Ilustración 6-5: Registro del componente en el *backend*

6.3.3 Creación de la tabla en la BD

Antes de construir una interfaz para mostrar las compañías del IBEX 35, es necesario crear un espacio en la BD donde se almacenará la información. Asumiendo que el prefijo de las tablas de Joomla! es **jos_** (se indica en la sección **Sitio** | **Configuración global** | **Servidor** del *backend*), se introduce la siguiente sentencia en la consola SQL de PhpMyAdmin:

```
CREATE TABLE IF NOT EXISTS `jos_companiasibex` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `ticker` varchar(10) NOT NULL,
  `isin` varchar(12) NOT NULL,
  `nombre` varchar(100) NOT NULL,
  `descripcion` text NOT NULL,
  `fecha_inicio` date NOT NULL,
  `domicilio` varchar(50) NOT NULL,
  `telefono` int(13) NOT NULL,
  `url` varchar(100) NOT NULL,
  `uptoday` tinyint(1) NOT NULL,
  `published` tinyint(1) NOT NULL,
  `capital_permitido` bigint(20) NOT NULL,
  `numero_acciones` int(11) NOT NULL,
  `pais` varchar(30) NOT NULL,
  PRIMARY KEY (`id`)
);
```

Código 6-2: Creación de la tabla compañíasIbex en la BD

Cada fila de la tabla representa a una compañía

6.3.4 Creación de la clase JTable

En PHP se pueden escribir individualmente funciones encargadas de las consultas SQL que añaden, actualizan y eliminan elementos de la BD. Sin embargo, es preferible no introducir sentencias SQL en el código ya que hace vulnerable al sistema de los ataques *hackers*. Afortunadamente, el equipo Joomla! ha escrito funciones que automatizan estas consultas. Concretamente, la clase **JTable** crea, lee, actualiza y elimina registros de una sola tabla de la BD.

Para sacar provecho de **JTable**, hay que extenderla a la clase hija **jos_companiasibex**.

Por lo tanto, se añade el siguiente código en los archivos **companiasibex.php** situados en los directorios *pfc->administrator->components->com_finanzas->tables y pfc ->components->com_finanzas->tables*:

```
<?php
defined('_JEXEC') or die('Restricted access');
jimport('joomla.database.table');
class TableCompaniasibex extends JTable
{
	var $id = null;
	var $ticker = null;
	var $isin = null;
	var $nombre = null;
	var $descripcion = null;
	var $fecha_inicio = null;
	var $domicilio = null;
	var $telefono = null;
	var $url = null;
	var $uptoday = null;
	var $published = null;
	var $capital_permitido = null;
	var $numero_acciones = null;
	var $pais = null;
	function __construct(&$db)
	{
		parent::__construct('#__companiasibex','id',$db);
	}
}
```

Código 6-3: Clase TableCompaniasIbex

La primera línea de código `defined('_JEXEC') or die('Restricted access')` es usado para dar seguridad a Joomla!. Ninguna aplicación externa puede acceder a un fichero con esta definición.

Para poder usar tablas de la BD se importan los archivos de Joomla! necesarios con `jimport('joomla.database.table')`.

Cuando se extiende la clase **JTable**, se añaden como variables todas las columnas de la tabla **jos_companiasibex** de la BD y se inicializan a **null.** Además, se sobrescribe le método _construct cuya función es la de llamar al constructor padre usando el nombre de la tabla de la BD (con el prefijo #__), la llave primaria y el objeto BD.

6.3.5 Implementación del *backend*

Después de realizar todas las instalaciones y configuraciones, se implementa el *backend* del componente. En este caso en particular, en la zona administrativa debe aparecer una lista de las compañías del IBEX 35 con sus propiedades. Además, se debe permitir añadir, modificar, guardar y eliminar las compañías.

Para ello hay que tener un controlador como punto de entrada del componente. Éste estará a la espera de peticiones de usuarios para realizar o delegar tareas.

Todos los ficheros que se implementan a continuación se encuentran dentro de *pfc->administrator->components->com_finanzas* [Ver sección 6.3.1-Creación de la estructura de ficheros del componente].

6.3.5.1 El controlador

El controlador del componente lleva por nombre *finanzas.php*. Dentro de éste, se añade una clase llamada **FinanzasController** que extiende a **JController.** Además el controlador debe ser creado y llamado desde el mismo fichero. Esto se consigue introduciendo las siguientes instrucciones:

```
<?php
defined( '_JEXEC' ) or die( 'Restricted access' );
jimport('joomla.application.component.controller');

class FinanzasController extends JController{}

$controller = new FinanzasController();
$controller->execute( $task );
$controller->redirect();
```

Código 6-4: creación e instanciación de la Clase FinanzasController

El método *execute* de **JController** se encarga de ejecutar la función o petición del usuario que lleva como nombre *$task*. Por defecto, en Joomla! hay funciones que permiten guardar, aplicar una acción, eliminar, editar y publicar elementos de la clase **JTable**. Todas estas funciones se implementarán dentro de la clase **FinanzasController** según nuestras necesidades.

Inicialmente, cuando se muestra el listado de las compañías del IBEX 35 en el *backend*, se proporcionará por defecto las opciones de publicar, quitar publicación, editar, borrar y nuevo.

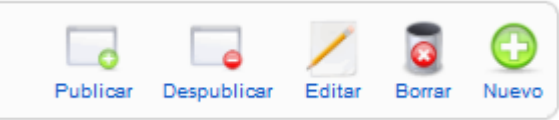


Ilustración 6-6: Botones usados en el listado de compañías

En cambio, cuando se esté editando una compañía específica, las acciones que se podrán aplicar serán guardar, aplicar y cancelar.

Ilustración 6-7: Botones usados en la edición/creación de una compañía

Esto se consigue añadiendo un *switch* antes de la clase **FinanzasController** donde se crea una barra de herramientas con **JToolBarHelper**:

```
switch ($task) {
case 'add':
	JToolBarHelper::save();
	JToolBarHelper::apply();
	JToolBarHelper::cancel();
	break;
default:
	JToolBarHelper::title( JText::_( 'Componentes de IBEX 35' ),
'generic.png' );
	JToolBarHelper::publishList();
	JToolBarHelper::unpublishList();
	JToolBarHelper::editList();
	JToolBarHelper::deleteList();
	JToolBarHelper::addNew();
	break;
}
```

Código 6-5: Creación de los botones

Una vez creados los botones, solamente falta implementar las funciones que realicen sus tareas correspondientes.

De momento, solamente se declaran dentro de la clase **FinanzasController**:

`function add(){}`	Añadirá una nueva compañía del IBEX35
`function save(){}`	Guardará la compañía en la BD
`function edit(){}`	Editará una compañía existente
`function publish(){}`	Permitirá que se pueda publicar/*despublicar* la compañía dentro de la página web.
`function remove(){}`	Eliminará una o varias compañías de la BD. A pesar de que en **JToolBarhelper** le llama *deleteList*, se llama a la función *remove* cuando se clica al botón.
`function display(){}`	Muestra la vista que se pida.

6.3.5.2 Las vistas

Dentro del directorio *view* se implementan todas las vistas del componente.

6.3.5.2.1 Vista All

En el ejemplo, la vista *all* se encargará de mostrar todas las compañías del IBEX 35 almacenadas en la BD. Para ello, se añade el siguiente código en el fichero *view.html.php* que se encuentra dentro de la carpeta *all*:

```
<?php
defined( '_JEXEC' ) or die( 'Restricted access' );
jimport( 'joomla.application.component.view');
class FinanzasViewAll extends JView
{
	function display($tpl = null)
	{
		$rows =& $this->get('data');
		$this->assignRef('rows', $rows);
		parent::display($tpl);
	}
}
```

Código 6-6: Clase FinanzasViewAll

En la definición de la clase se ha seguido una norma que establece el marco de trabajo Joomla!, que consiste en poner primero el **nombre del componente** que lo llama, seguido de **View** y del **nombre de la vista**:

NombreComponenteViewNombreVista

Esta norma se sigue también para los modelos.

En la primera línea de la función *display* se llama al método *getData()* del modelo que tenga como nombre **FinanzasModelAll**. Cuando no se indica el modelo que se quiere instanciar, Joomla! recoge la información del modelo por defecto cuyo nombre coincide con el de la vista. El método *getData()* devuelve todas las compañías del IBEX 35 en formato lista.

Esta lista se pasa por referencia al "*layout*" mediante *assingRef*.

Finalmente se ejecuta el método *display* de la clase padre donde se puede indicar por parámetro el nombre del *layout* que se quiere visualizar. En este caso, se envía el parámetro *null* para indicar que se visualice el *layout* por defecto, *default.php.*

El archivo *default.php* contendrá el siguiente código:

```
<?php
defined( '_JEXEC' ) or die( 'Restricted access' );
?>
<form action="index.php" method="post" name="adminForm">
<table class="adminlist">
  <thead>
    <tr>
      <th width="20">
        <input type="checkbox" name="toggle"
             value="" onclick="checkAll(<?php echo
             count( $this->rows ); ?>);" />
      </th>
      <th class="title">Ticker</th>
      <th width="15%">ISIN</th>
      <th width="15%">Nombre</th>
      <th width="10%">Fecha Inicio</th>
      <th width="10%">Domicilio</th>
      <th width="10%">Telefono</th>
      <th width="10%">URL</th>
      <th width="10%">Capital Permitido</th>
      <th width="10%">Numero Acciones</th>
        <th width="10%">Pais</th>
      <th width="5%" nowrap="nowrap">Uptoday</th>
      <th width="5%" nowrap="nowrap">Published</th>
    </tr>
  </thead>

  <?php
  jimport('joomla.filter.output');
  $k = 0;
  for ($i=0, $n=count( $this->rows ); $i < $n; $i++)
  {
    $row = &$this->rows[$i];
    $checked = JHTML::_('grid.id', $i, $row->id );
    $published = JHTML::_('grid.published', $row, $i );
    $link = JFilterOutput::ampReplace( 'index.php?option=' . $option .
'&task=edit&cid[]='. $row->id );
```

```
    ?>
    <tr class="<?php echo "row$k"; ?>">
      <td>
        <?php echo $checked; ?>
      </td>
      <td>
        <a href="<?php echo $link; ?>"><?php echo $row->ticker; ?></a>
      </td><td>
        <?php echo $row->isin; ?>
      </td>
      <td>
        <?php echo $row->nombre; ?>
      </td>
      <td>
        <?php echo $row->fecha_inicio; ?>
      </td>
      <td>
        <?php echo $row->domicilio; ?>
      </td>
      <td>
        <?php echo $row->telefono; ?>
      </td>
      <td>
        <?php echo $row->url; ?>
      </td>
      <td>
        <?php echo $row->capital_permitido; ?>
      </td>
      <td>
        <?php echo $row->numero_acciones; ?>
      </td>
      <td>
        <?php echo $row->pais; ?>
      </td>
      <td align="center">
        <?php echo $row->uptoday;?>
      </td>
        <td align="center">
```

```
        <?php echo $published;?>
      </td>
    </tr>
    <?php
    $k = 1 - $k;
  }
  ?>
</table>
<?php echo JHTML::_('form.token');?>
<input type="hidden" name="option" value="<?php echo $option;?>" />
<input type="hidden" name="task" value="" />
<input type="hidden" name="boxchecked" value="0" />
</form>
```

Código 6-7: Archivo *default.php* de la vista All

Se empieza definiendo un formulario llamado *adminForm* que apunta a *index.php*. Una tabla con la clase *adminlist* es creada junto a sus cabeceras. Todas las cabeceras son propiedades de la compañía IBEX a excepción de la primera. Ésta actúa como una caja "selecciona todos" que selecciona todas las compañías IBEX de la pantalla.

Después de las cabeceras, se inicializa un bucle a través de la variable *rows*, recibida del *view.html.php*. Las variable $i y $n son inicializadas a 0 y al número de *rows(compañías)* respectivamente. El bucle itera mientras haya compañías para ser mostradas. Dentro del bucle, se obtiene una referencia de la compañía actual para poder mostrar y editar su contenido. El valor de $k se alterna entre 0 y 1 para usar dos clases diferentes del archivo CSS con distintas propiedades de fondo.

La mayoría de las propiedades de una compañía son mostradas directamente, pero algunas de las columnas tienen un trato especial. Usando la función **JHTML::_('grid.id')** se puede generar el código HTML para una caja de selección que será reconocida por el JavaScript del *backend*. La función **JHTML::_('grid.published')** genera una imagen basada en el valor de *published* del componente.

Una vez iterado todo el bucle, se han introducido tres variables escondidas.

La primera, contiene el valor de **opción** para que se pueda redireccionar al componente adecuado. En este caso **com_finanzas**.

La segunda, se llama ***task*** y obtiene el valor de la tarea solicitada por el usuario. Este valor se aplica antes de enviar el formulario gracias a JavaScript.

Cuando alguna de las cajas de selección se marca, es decir se elige una compañía del IBEX 35, la tercera variable escondida, ***boxchecked,*** recibe el valor 1. Volverá a tener un valor 0 cuando no haya ninguna compañía seleccionada.

Finalmente, cuando el archivo HTML está implementado, se actualiza la función **FinanzasController::display()** del controlador (archivo *finanzas.php*) con el siguiente código:

```
function display()
	{
		$view = JRequest::getVar('view');
		if (!$view) {
			JRequest::setVar('view', 'all');
		}
		parent::display();
	}
```

Código 6-8: Función display() del controlador

Display es llamada cuando ninguna tarea *$task* coincide con las funciones implementadas en la clase **FinanzasController**.

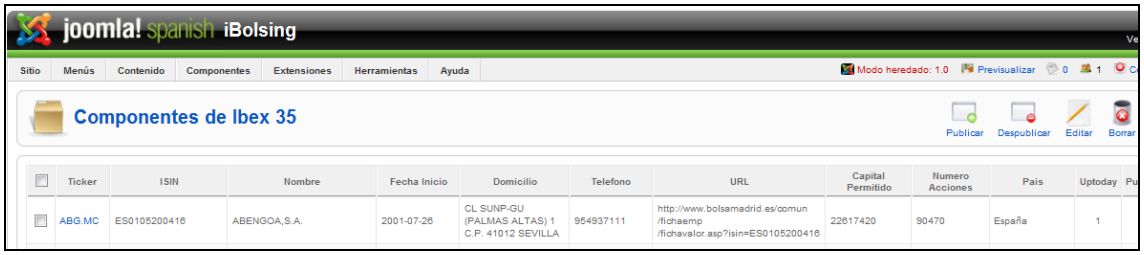
Con el código implementado, se podría ver en el *backend* una lista con todas las compañías del IBEX 35 que hay en la BD.

La tabla jos_companiasibex de la BD está vacía. Para introducir una compañía en la BD se escribe la siguiente instrucción SQL en PhpMyAdmin:

```
INSERT INTO `#__companiasibex` (`id`, `ticker`, `isin`, `nombre`,
`descripcion`, `fecha_inicio`, `domicilio`, `telefono`, `url`, `uptoday`,
`published`, `capital_permitido`, `numero_acciones`, `pais`) VALUES
(2, 'ABG.MC', 'ES0105200416', 'ABENGOA,S.A.', 'De momento no hay perfil',
'2001-07-26', 'CL SUNP-GU (PALMAS ALTAS) 1  C.P. 41012 SEVILLA', 954937111,
'http://www.bolsamadrid.es/comun/fichaemp/fichavalor.asp?isin=ES0105200416',
1, 1, 22617420, 90470, 'EspaÃ±a');
```

Código 6-9: Inserción de una compañía en la BD

Si todo ha funcionado correctamente, se debería mostrar al introducir en el navegador *http://localhost/pfc/administrator/index.php?option=com_finanzas* una imagen similar a la siguiente ilustración:



Código 6-10: Lista de compañías del IBEX35 en el *backend*

Pero un administrador de una página no suele tener acceso a *PhpMyAdmin*. Para que un administrador pueda gestionar compañías mediante una interfaz, se debe crear una vista. A esta vista la llamamos *single*.

Gracias a la vista *all* ya se pueden implementar las funciones borrar y publicar/quitar publicación de una compañía del IBEX 35.

REMOVE:

Se abre el controlador *finanzas.php* y se actualiza la función *remove():*

```
function remove()
{
	JRequest::checkToken() or jexit( 'Invalid Token' );
	global $option;
	$cid = JRequest::getVar('cid', array(0));
	$row =& JTable::getInstance('companiasibex', 'Table');
	foreach ($cid as $id) {
		$id = (int) $id;
		if (!$row->delete($id)) {
			JError::raiseError(500, $row->getError() );
		}
	}
	$s = '';
	if (count($cid) > 1) {
		$s = 's';
	}
	$this->setRedirect('index.php?option=' . $option, 'Review' . $s
. ' deleted.');
}
```

Código 6-11: Función remove()

Como la función hace modificaciones a la BD, los valores obtenidos por *Request* deben ser comprobados con *JRequest::checkToken().* Si todo es correcto, se reciben los identificadores de todas las compañías de IBEX 35 seleccionados con la caja de selección y se obtiene una instancia de la clase *TableCompaniasibex*.

A continuación se recorre identificador por identificador para aplicar la función *delete()* de Joomla!, que elimina la compañía de la BD. Finalmente, se redirecciona al controlador. Al no haber ninguna tarea especificada por *Request,* el controlador ejecuta la función *display()* que muestra la vista *all.*

PUBLISH/UNPUBLISH:

Para implementar la función *publish* se introduce el código del siguiente diagrama en la clase **FinanzasController**:

```
function publish()
{
	JRequest::checkToken() or jexit( 'Invalid Token' );
	global $option;
	$cid = JRequest::getVar('cid', array());
	$row =& JTable::getInstance('companiasibex', 'Table');
	$publish = 1;
	if ($this->getTask() == 'unpublish') {
		$publish = 0;
	}
	if(!$row->publish($cid, $publish))
	{
		JError::raiseError(500, $row->getError() );
	}
	$s = '';
	if (count($cid) > 1) {
		$s = 's';
	}
	$msg = 'Review' . $s;
	if ($this->getTask() == 'unpublish') {
		$msg .= ' unpublished';
	} else {
		$msg .= ' published';
	}
	$this->setRedirect('index.php?option=' . $option, $msg);
}
```

Código 6-12: Función publish()

Se aplica la misma idea que *remove*(), simplemente que ahora el valor de *publish* varía de 1 a 0 según si se publica o no la compañía.

La función *publish* de la clase padre puede publicar todas las compañías a la vez sin la necesidad de aplicar un bucle que recorra los identificadores uno a uno.

Ya que la función que quita la publicación hace lo mismo, cambiando el valor de *publish* a 0 (parte en negrita del código), se reúsan instrucciones.

Joomla! permite reusar código registrando la tarea *unpublish* como *publish. Es decir*, se ejecutará la función *publish()* cuando el usuario clica al botón *despublicar*. Se añade la siguiente línea al final del archivo del controlador para que se cumpla lo mencionado:

```
$controller = new FinanzasController();
$controller->registerTask('unpublish', 'publish');
$controller->execute( $task );
$controller->redirect();
```

Código 6-13: Registro de la tarea unpublish()

6.3.5.2.2 Vista single

Para poder añadir las acciones de editar, añadir, guardar y aplicar en el controlador se implementa una vista donde se muestra el contenido de una compañía IBEX 35 específica.
El fichero *view.html.php* de la carpeta *single* contendrá el siguiente código:

```
<?php
defined( '_JEXEC' ) or die( 'Restricted access' );

jimport( 'joomla.application.component.view');

class FinanzasViewSingle extends JView
{
	function display($tpl = null)
	{
		$row =& JTable::getInstance('Companiasibex', 'Table');
		$cid = JRequest::getVar( 'cid', array(0), '', 'array' );
		$id = $cid[0];
		$row->load($id);
		$this->assignRef('row', $row);

		$editor =& JFactory::getEditor();
		$this->assignRef('editor', $editor);

		$this->assignRef('uptoday', JHTML::_('select.booleanlist',
'uptoday', 'class="inputbox"', $row->uptoday));
		$this->assignRef('published',
JHTML::_('select.booleanlist', 'published', 'class="inputbox"', $row-
>published));
		$this->assignRef('fecha_inicio',	JHTML::_('calendar',
$row->fecha_inicio, 'fecha_inicio', 'fecha_inicio'));

		parent::display($tpl);
	}
}
```

Código 6-14: Clase FinanzasViewSingle

Este código crea una clase basada en **JView** llamada **FinanzasViewSingle**. Esta clase simplemente contiene la función *display()* usada para añadir datos y elementos a la vista. Como se ha explicado anteriormente, se crea una instancia de la clase **TableCompaniasibex** llamada *row*. *Row* obtiene los valores de la compañía IBEX 35 de la BD con identificador *$id* cuyo valor es recibido por *JRequest*.

Además se crean mediante JHTML un editor de texto, dos listas booleanas y un calendario haciendo referencia a los valores de descripción, uptoday, published y fecha de entrada de la compañía en el IBEX 35.

El editor de texto sigue el formato WYSIWYG(What You See Is What You Get - los que se ve es lo que se obtiene) que permite al usuario generar un texto con formato e imágenes sin necesidad de saber HTML.

Cuando todos los elementos son asignados a la vista, se llama a la función *parent::display()* encargada de cargar un fichero de salida basado en el valor de *$tpl*. Si éste es nulo, se carga el fichero *default.php* de la carpeta *tmpl*.

En el fichero default.php, se introduce el siguiente código:

```
<?php
defined( '_JEXEC' ) or die( 'Restricted access' );
JHTML::_('behavior.calendar');
$editor =& JFactory::getEditor();
if ($this->row->ticker) {
     JToolBarHelper::title(   JText::_(   'Edit   Component'   ),
'addedit.png' );
} else {
     JToolBarHelper::title(   JText::_(   'Add   Component'   ),
'addedit.png' );
}
?>
<script language="javascript" type="text/javascript">
function submitbutton(pressbutton) {
     if (pressbutton == 'save' || pressbutton == 'apply') {
          var  descripcion  =  <?php  echo  $editor->getContent(
'descripcion' ); ?>
          if (document.adminForm.ticker.value == '') {
               alert("Please   enter   the   ticker   of   the
component.");
          } else if (document.adminForm.isin.value == '') {
               alert("Please   enter   the   isin   of   the
component.");
          } else if (document.adminForm.name.value == '') {
               alert("Please   enter   the   name   of   the
component.");
          }    else    if    (descripcion    ==    ''    &&
document.adminForm.descripcion.value == '') {
               alert("Please  enter  the  description  of  the
component.");
          } else {
               submitform(pressbutton);
          }
     } else {
          submitform(pressbutton);
     }
}
</script>
<form    action="index.php"    method="post"    name="adminForm"
id="adminForm" enctype="multipart/form-data">
  <fieldset class="adminform">
    <legend>Detalles</legend>
    <table class="admintable">
    <tr>
      <td width="
100" align="right" class="key">
        Ticker:
      </td>
      <td>
        <input   class="text_area"   type="text"   name="ticker"
id="ticker"  size="50"  maxlength="250"  value="<?php  echo  $this-
```

```
>row->ticker;?>" />
      </td>
    </tr>
    <tr>
      <td width="100" align="right" class="key">
        ISIN:
      </td>
      <td>
        <input    class="text_area"    type="text"    name="isin"
id="isin"  size="50"  maxlength="250"  value="<?php  echo  $this-
>row->isin;?>" />
      </td>
    </tr>
     <tr>
      <td width="100" align="right" class="key">
        Nombre:
      </td>
      <td>
        <input   class="text_area"   type="text"   name="nombre"
id="nombre" size="50" maxlength="250" value="<?php echo $this-
>row->nombre;?>" />
      </td>
    </tr>
     <tr>
      <td width="100" align="right" class="key">
        Descripcion:
      </td>
      <td>
        <?php
        echo  $this->editor->display(  'descripcion',    $this-
>row->descripcion, '100%', '150', '40', '5' ) ;
        ?>
      </td>
    </tr>
    <tr>
      <td width="100" align="right" class="key">
        Domicilio:
      </td>
      <td>
        <input  class="text_area"  type="text"  name="domicilio"
id="domicilio"   size="50"   maxlength="250"   value="<?php   echo
$this->row->domicilio;?>" />
      </td>
    </tr>
    <tr>
      <td width="100" align="right" class="key">
        Telefono:
      </td>
      <td>
        <input  class="text_area"  type="text"  name="telefono"
id="telefono"   size="50"   maxlength="250"   value="<?php   echo
$this->row->telefono;?>" />
      </td>
```

```
    </tr>
      <tr>
      <td width="100" align="right" class="key">
        URL:
      </td>
      <td>
        <input    class="text_area"    type="text"    name="url"
id="url"  size="50"  maxlength="250"  value="<?php  echo  $this-
>row->url;?>" />
      </td>
    </tr><tr>
      <td width="100" align="right" class="key">
        Capital Permitido:
      </td>
      <td>
        <input           class="text_area"           type="text"
name="capital_permitido"    id="capital_permitido"    size="50"
maxlength="250"       value="<?php       echo       $this->row-
>capital_permitido;?>" />
      </td>
    </tr>
      <tr>
      <td width="100" align="right" class="key">
        Numero de acciones:
      </td>
      <td>
        <input           class="text_area"           type="text"
name="numero_acciones"      id="numero_acciones"      size="50"
maxlength="250"       value="<?php       echo       $this->row-
>numero_acciones;?>" />
      </td>
    </tr>
    <tr>
      <td width="100" align="right" class="key">
        Pais:
      </td>
      <td>
        <input    class="text_area"    type="text"    name="pais"
id="pais"  size="50"  maxlength="250"  value="<?php  echo  $this-
>row->pais;?>" />
      </td>
    </tr>
    <tr>
      <td width="100" align="right" class="key">
        Uptoday:
      </td>
      <td>
        <?php echo $this->uptoday; ?>
      </td>
    </tr>
    <tr>
      <td width="100" align="right" class="key">
        Fecha inicio:
```

```
      </td>
      <td>
           <?php echo $this->fecha_inicio; ?>
      </td>
    </tr>
    <tr>
      <td width="100" align="right" class="key">
        Publicado:
      </td>
      <td>
        <?php echo $this->published; ?>
      </td>
    </tr>
    </table>
  </fieldset>
  <input type="hidden" name="id" value="<?php echo $this->row-
>id; ?>" />
  <input   type="hidden"   name="option"   value="<?php   echo
$option;?>" />
  <input type="hidden" name="task" value="" />
  <?php echo JHTML::_( 'form.token' ); ?>
</form>
```

Código 6-15: Fichero *default.php* de la vista Single

La mayor parte de este fichero consiste en código HTML con un poco de PHP. La mayoría de código PHP consiste en el uso de *echo* para mostrar en pantalla las propiedades de la compañía IBEX 35 que se habían asignado con *assignRef()* en el archivo *view.html.php*. Sin embargo, algunas propiedades más complejas no se muestran con simples *echo's.*

La primera línea de código, como en todos los ficheros PHP de Joomla!, comprueba que el fichero se accede desde Joomla! y no directamente.

Después, *JHTML::__('behavior.calendar');* carga el JavaScript necesario para ejecutar el calendario emergente que será mostrado. Aunque se está haciendo la llamada aquí en este archivo, el código JavaScript será automáticamente cargado en el <head> del documento HTML principal. Esto es porque Joomla! almacena toda la salida mientras el código se ejecuta y después la envía al navegador como último paso.

También se está produciendo la salida del valor **$option** aunque nunca se asigna en la vista. La clase padre **JView** recoge la opción automáticamente antes de cargar el fichero de salida.

Además, se envía la **tarea** que recibirá el controlador y el **identificador** de la compañía del IBEX 35.

Finalmente, se muestra la salida del resultado de *JHTML::_('form.token');* que añade un token generado automáticamente al formulario para verificar la sesión de un usuario.

Ahora que ya se ha implementado la vista *single*, se pueden implementar los métodos *add(), save(), edit()* del controlador en el archivo *finanzas.php*:

ADD y EDIT:

```
function add()
{
	JRequest::setVar('view', 'single');
	$this->display();
}
```

Código 6-16: Función add()

Estos métodos simplemente asignan al componente finanzas la vista *single* y llaman al método *display()*. De esta forma, cuando se clica a los botones *new o edit* de **JToolBarHelper** se mostrará en el navegador la vista *single:*

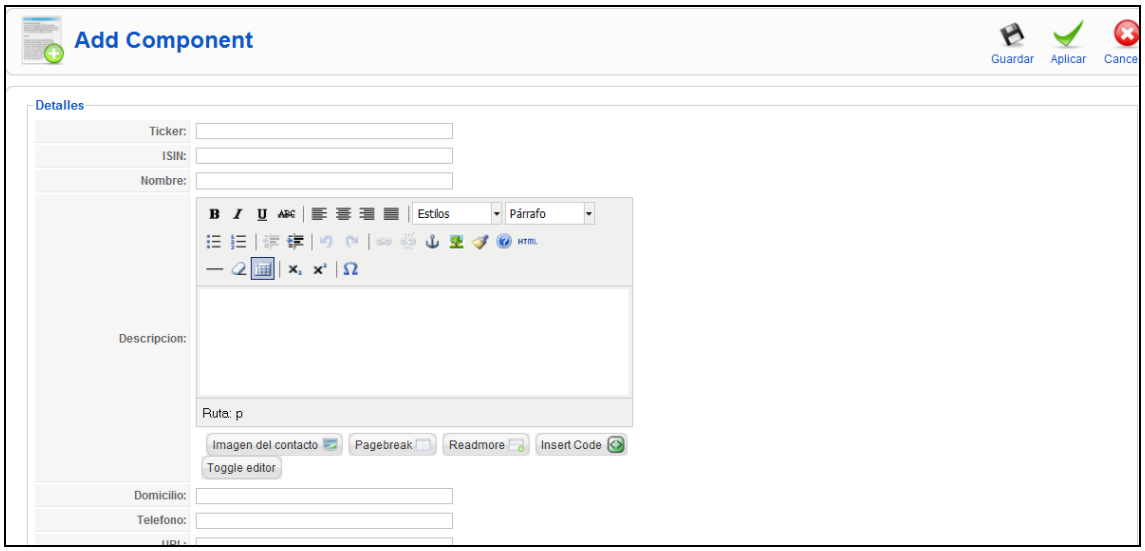


Ilustración 6-8: Vista Single del *backend*

Al clicar el botón *cancelar* se ejecuta el método *display()* del controlador que asigna la vista *all* y la muestra. Esto es debido a que no hay ninguna función implementada en el controlador con dicho nombre.

SAVE o APPLY:

Guardar y aplicar ejecutarán la misma función llamada *save()*. Esto implica añadir `$controller->registerTask('apply', 'save');` al final del fichero, tal como se hizo para *publish/unpublish*.

```
function save()
{
	JRequest::checkToken() or jexit( 'Invalid Token' );
	global $option;
	$row =& JTable::getInstance('companiasibex', 'Table');
	if (!$row->bind(JRequest::get('post')))
	{
		JError::raiseError(500, $row->getError() );
	}
	$row->descripcion = JRequest::getVar( 'descripcion', '', 'post', 'string', JREQUEST_ALLOWRAW );
	if(!$row->fecha_inicio)
	{
		$row->fecha_incio = date( 'Y-m-d H:i:s' );
	}
	if (!$row->store())
	{
		JError::raiseError(500, $row->getError() );
	}
	if ($this->getTask() == 'apply') {
		$this->setRedirect('index.php?option=' . $option . '&task=edit&cid[]=' . $row->id, 'Changes Applied');
	} else {
		$this->setRedirect('index.php?option=' . $option, 'Review Saved');
	}
}
```

Código 6-17: Función Save/Apply

Primero, *JRequest::checkToken()* determina si el token ha sido enviado para la solicitud (*Request*). Como esta función modifica la BD, *checkToken()* es usado para asegurarse que la solicitud es legítima.

Después, crea una instancia de la clase **TableCompaniasibex**, que representa a una compañía del IBEX 35. El método *bind()* es usado para cargar todos los valores del formulario a la instancia creada. Para reducir el riesgo de ataques de inyección SQL, se llama a *JRequest::get()* que desinfecta los valores del $_POST. Si la función da error, se muestra un mensaje por pantalla y se para la ejecución.

Después de hacer el *binding*, se pueden manipular todas las variables de la instancia directamente:

1- Cómo *descripción* acepta contenido HTML y *bind()* descarta automáticamente código HTML, se necesita una especial manipulación. Para obtener la información, se llama a *getVar()* del *JRequest*. Los parámetros de la función son el nombre de la variable, el valor por defecto, el tipo de envío (get o post), el tipo de dato esperado y el indicador JREQUEST_ALLORAW.

2- La variable fecha_inicio se manipula para que tenga un formato año-mes-día usado en la BD.

Finalmente, se llama al método *store()* que recoge todas las variables de la compañía del IBEX 35 y crea una inserción o actualización en la BD según el valor de **id.** Si id tiene valor se crea una actualización, si no se crea una inserción. Si hay algún error en la ejecución del método, se muestra por pantalla y se para la ejecución. Por el contrario, se redirecciona el componente.

Cuando se ha clicado el botón "*aplicar"* se redirecciona a la vista *single* de la compañía IBEX 35 con identificador **id**. Al clicar "*guardar"* se redirecciona a la vista *all,* por lo que se muestra el listado de todas las compañías del IBEX 35.

El código en negrita muestra la parte añadida cuando se quiere usar el método *aplicar*.

6.3.5.3 El modelo

Dentro de la carpeta *models* se encuentran todos los modelos del componente. Los modelos acceden a información de la BD. En este caso, sólo es necesario el modelo *all* llamado desde la vista cuyo con nombre es el mismo.

Se introduce el siguiente código en el fichero *all.php* del directorio models:

```
<?php
defined( '_JEXEC' ) or die( 'Restricted access' );
jimport('joomla.application.component.model');
class FinanzasModelAll extends JModel
{
	var $data = null;
	function getData()
	{
		if (empty($this->data)) {
			$query = "SELECT * FROM #__companiasibex";
			$this->data = $this->_getList($query);
		}
		return $this->data;
	}
}
```

Código 6-18: Clase finanzasModelAll

Para poder beneficiarse de la clase **JModel** hay que hacer una importación del código de Model.

Siguiendo la nomenclatura de Joomla! al modelo se le llama **FinanzasModelAll** y extiende **JModel**.

Dentro, se implementa el método *getData()* que devuelve una lista con las compañías del IBEX 35 que se encuentren en la tabla *jos_companiasibex* de la BD. Joomla! se encarga de realizar la conexión a la BD automáticamente y de forma segura.

6.3.6 Implementación del *frontend*

La implementación del *backend* ha permitido poder crear componentes del IBEX 35 en la BD. Cuando un usuario final visite la aplicación estará interesado en ver las compañías y sus cotizaciones actuales. El *frontend*, es el encargado de mostrar estos datos. Para ello, hay que implementar todos los archivos que se encuentran en la estructura de ficheros del *frontend* [Ver sección 6.3.1-Creación de la estructura de ficheros del componente

6.3.6.1 Punto de entrada de la aplicación

El archivo *finanzas.php* es el punto de entrada de la aplicación. Su funcionalidad es la de crear el controlador **FinanzasController**, ejecutar la tarea solicitada por el usuario y redireccionar el componente.

Se introduce el siguiente código en el archivo mencionado:

```
<?php
defined('_JEXEC') or die('Restricted access');
require_once (JPATH_COMPONENT.DS.'controller.php');
$controller = new FinanzasController();
$controller->execute( JRequest::getCmd('task'));
$controller->redirect();
```

Código 6-19: Creación e instanciación del controlador del *frontend*

6.3.6.2 El controlador

Al tratarse únicamente de una aplicación que muestra datos por pantalla (no hay formularios de entrada) el controlador solamente tiene dos funciones.

La primera, llamada *display(),* simplemente recoge la vista actual y la muestra mediante el método padre *display()*:

```
function display()
{
	$view = JRequest::getVar('view');
	parent::display();
}
```

Código 6-20: Función display() del controlador

Y la segunda, llamada *consultarCompania()*, se ejecuta cuando se quiere mostrar una compañía del IBEX 35. Este método obtiene la vista *consultarCompania*, le asigna el modelo *cotizacionDia* y lo ejecuta. Ahora esta vista podrá usar dos modelos:

1- cotizacionDia
2- El modelo por defecto *consultarCompania.*

En este caso, el segundo modelo no existe, por lo que no se usará.

```
function consultarCompania()
{
	$view = & $this->getView('ConsultarCompania','html');
	$view->setModel($this->getModel('cotizacionDia', 'finanzasModel'
), true);
	$view->display();
}
```

Código 6-21: Función consultarCompania() del controlador

Lógicamente, en el proyecto final, el controlador tiene muchas más funciones que las mencionadas.

6.3.6.3 Las vistas

Tal como pasaba en el *backend*, hay dos vistas: una para mostrar la lista de las compañías del IBEX 35 y la otra para mostrar el contenido de cada una individualmente.

6.3.6.3.1 Vista consultarListaCompania

En el fichero *view.html.php* de la carpeta co*nsultarListaCompania* se añade exactamente el mismo código que en la sección [Ver sección 6.3.5.2.1Vista All]. Explicar el código sería añadir información redundante.

El archivo *default.php* contendrá el siguiente código:

```
<?php defined( '_JEXEC' ) or die( 'Restricted access' ); ?>
<div class="componentheading"><?php echo JText::_('Components for IBEX
35') ?></div>
<ul>
      <?php

      foreach ($this->rows as $row)
      {
            $link = JRoute::_('index.php?option=com_finanzas&id=' .
$row->id . '&task=consultarCompania&view=consultarComponente');
            echo '<li><a href="' . $link . '">' . $row->ticker .
'</a>';

            echo '</li>';
      }

      ?>
<ul>
```

Código 6-22: Archivo *default.php* de la vista consultarListaCompania

Se añade el título del contenido con las propiedades de la clase *componentheading* del archivo CSS y se crea una lista con los enlaces de cada uno de los componentes del IBEX 35. El enlace llama a la tarea *consultarCompania* del controlador y envía el identificador de la compañía.

Al ejecutar en el navegador la dirección

http://localhost/pfc/index.php?option=com_finanzas&view=consultarListaComponentes

se observará una imagen como la siguiente:

Código 6-23: Lista de compañías del IBEX35 en el *frontend*

6.3.6.3.2 Vista consultarCompania

La vista **consultarCompania** recibe el identificador de la compañía por parámetro. A continuación, crea una instancia de la clase **Tablecompaniasibex**, carga la compañía especificada por el *id* con la función *load() y* la asigna por referencia al *layout* mediante *AssignRef()*.

Seguidamente, obtiene una instancia del modelo para ejecutar el método *getcotización()*, que obtiene la cotización actual en el mercado de valores de la compañía con identificador *ticker*. Los datos se obtienen descargando un fichero de la página web de *YahooFinance*. Todos los datos de la cotización se pasan por referencia.

Las variables uptoday y fecha_inicio las prepara y las asigna por referencia al *layout*.

Además, crea un enlace de retroceso para volver a la lista de compañías.

La instrucción `echo"<meta http-equiv='refresh' content='60'>";` permite que toda la página se refresque cada período temporal. Así se actualiza el valor de la compañía a tiempo real. Gracias a AJAX se puede refrescar solamente una sección de la página [Ver 6.3.7-Añadir AJAX al componente]. Para no complicar más el código, no se añade la funcionalidad AJAX a continuación.

Todo esto se consigue introduciendo el siguiente código en el archivo *view.html.php* de la carpeta *consultarCompania*.

```
<?php
defined( '_JEXEC' ) or die( 'Restricted access' );

jimport( 'joomla.application.component.view');

class FinanzasViewConsultarCompania extends JView
{
	function display($tpl = null)
	{
		$id = (int) JRequest::getVar('id', 0);
		$componente	=&	JTable::getInstance('companiasibex',
'Table');
		$componente->load($id);
		$model = $this->getModel();
		$cotizaciones	=	$model	->	getCotizacion($componente-
>ticker);
		if ($componente->published == 0) {
			JError::raiseError(404, 'The component you requested
is not available.' );
		}

		if ($componente->uptoday == 1) {
			$uptoday = 'Si';
		} else {
			$uptoday = 'No';
		}

		$fecha = JHTML::Date($componente->fecha_inicio);

		$backlink = JRoute::_('index.php?option=com_finanzas');
		echo"<meta http-equiv='refresh' content='60'>";
		$this->assignRef('cotizaciones', $cotizaciones);
		$this->assignRef('componente', $componente);
		$this->assignRef('uptoday', $uptoday);
		$this->assignRef('fecha', $fecha);
		$this->assignRef('backlink', $backlink);

		parent::display($tpl);
	}
}
```

Código 6-24: Clase FinanzasViewConsultarCompania

Ahora hay dos *layouts* que muestran por pantalla información:
El *layout* por defecto, *default.php,* que muestra el perfil de la compañía en una tabla y el *layout default_cotizacion.php*, con la cotización actual en el mercado.
Los dos archivos siguen una estructura parecida a los *layouts* vistos hasta el momento, por lo que no se explicará cómo se ha implementado. Simplemente es HTML junto con *echo's* de PHP. El archivo *default_cotizacion.php* usa las imágenes para indicar si un componente está creciendo o decreciendo.

A continuación se muestra el código que se inserta en los dos archivos:

Default.php

```
<?php defined( '_JEXEC' ) or die( 'Restricted access' ); ?>
<p    class="componentheading"><?php    echo    htmlspecialchars($this-
>componente->ticker); ?></p>
<h1><em><p class="createdate"><?php echo JText::_('In IBEX Since')?>:
<?php echo $this->fecha; ?></p></em></h1>
<?php         if($this->cotizaciones!=null)         echo         $this-
>loadTemplate('cotizacion');?>
<p> </p>
<table  cellspacing="0"  cellpadding="4"  width="20%"  frame="border"
bordercolor="#CCCCCC" align="right">
      <tbody>
      <tr>
            <td        bgcolor="#FFFFFF"><b>ISIN:</b><?php        echo
htmlspecialchars($this->componente->isin); ?></td>
      </tr>
</tbody></table>
<table cellspacing="0" cellpadding="4" width="100%"  bgcolor="#B0C1D9"
frame="border" bordercolor="#CCCCCC">
      <tr>
            <td             colspan="2"><strong><?php             echo
JText::_('Name')?>:</strong>   <?php   echo   htmlspecialchars($this-
>componente->nombre); ?></td>
      </tr>
      <tr>
            <td             colspan="2"><strong><?php             echo
JText::_('Address')?>:</strong>  <?php  echo  htmlspecialchars($this-
>componente->domicilio); ?></td>
      </tr>
      <tr>
            <td             colspan="2"><strong><?php             echo
JText::_('Country')?>:</strong>  <?php  echo  htmlspecialchars($this-
>componente->pais); ?></td>
      </tr>
      <tr>
            <td             colspan="2"><strong><?php             echo
JText::_('Phone')?>:</strong>   <?php   echo   htmlspecialchars($this-
>componente->telefono); ?></td>
      </tr>
      <tr bgcolor="#EEEEEE">
            <td colspan="2">
            <table    cellspacing="0"    cellpadding="4"    width="60%"
frame="border" bgcolor="#FFFFFF" bordercolor="#CCCCCC" align="center">
      <tbody>
<tr>
            <td align="center"><strong><?php echo JText::_('Amount of
capital')?></strong></td>
            <td align="center"><strong><?php echo JText::_('Number of
shares')?></strong></td>
            <td   align="center"><strong><?php   echo   JText::_('Up
today')?></strong></td>
      </tr>
            <tr>
            <td    align="center"><?php    echo    $this->componente-
>capital_permitido; ?></td>
            <td   align="center"><?php   echo   htmlspecialchars($this-
```

```
>componente->numero_acciones); ?></td>
            <td align="center"><?php echo $this->uptoday ?></td>
      </tr>
</tbody></table>
            </td>
      </tr>
      <tr bgcolor="#EEEEEE">
            <td colspan="2">
<?php echo $this->componente->descripcion; ?></td>
      </tr>
      </tbody>
</table>
      <em><strong><?php echo JText::_('Website from La Bolsa de Madrid
about')?></strong>   <a   href="<?php   echo   htmlspecialchars($this-
>componente->url); ?>"><?php echo htmlspecialchars($this->componente-
>nombre); ?></a></em>
<p> </p>
<p   align="center"><a   href="<?php   echo   htmlspecialchars($this-
>backlink);   ?>"><?php   echo   JText::_('Return   to   the   list   of
components')?></a></p>
```

Código 6-25: Archivo *default.php* de la vista ConsularCompania

Default_cotizacion.php:

```
<?php defined( '_JEXEC' ) or die( 'Restricted access' ); ?>
      <table align="center" width="80%">
      <?php
      echo "<tr>";echo "<td><b>";
      echo JText::_('Last Trade');
      echo ":</b></td><td align='right'
id='transaccion'><big><b>".$this-
>cotizaciones[1]."&euro;</big></b></td>";
      echo "<td width=30%></td>";echo "<td><b>";
      echo JText::_('Open');
      echo ": </b></td><td align='right'>".$this-
>cotizaciones[5]."</td>";
      echo "</tr>";echo "<tr>";echo "<td><b>";
      echo JText::_('Time');
      echo ": </b></td><td align='right' id='hora'>".$this-
>cotizaciones[2]."</td>";
      echo "<td width=30%></td>";
      echo "<td><b>";
      echo JText::_('Range');
      echo ": </b></td><td align='right'>".$this->cotizaciones[7]." -
".$this->cotizaciones[6]."</td>";
      echo "</tr>";
      echo "<tr>";
      $string = substr($this->cotizaciones[4],0,-4);
      if($string=='-'){
            echo "<td><b>";
            echo JText::_('Change');
            echo ": </b></td><td align='right' id='valor'><font
      color='#FF0000'><img height='14' border='0' width='10'
      src='components/com_finanzas/imagenes/down_r.gif'> ";
      }
      else{
            echo "<td><b>";
            echo JText::_('Change');
            echo ": </b></td><td align='right' id='valor'><font
      color='#0000FF'><img height='14' border='0' width='10'
      src='components/com_finanzas/imagenes/up_g.gif'> ";
      }
      echo substr($this->cotizaciones[4],1)."</font></td>";
      echo "<td width=30%></td>";
      echo "<td><b>";
      echo JText::_('Volume');
      echo": </b></td><td align='right'>".$this-
>cotizaciones[8]."</td>";
      echo "</tr>";echo "<tr>";echo"<td colspan><b>";
      echo JText::_('Volatility');
      echo": </b></td><td align='right'>".$this->volatilidad."%</td>";
      echo "</td>";
      echo"<td colspan='4' align='right'><small><em>";
      echo JText::_('Prices of the day');
      echo " ".$this->cotizaciones[3];
      echo "</small></em></td>";echo "</tr>";
      ?>
</table>
```

Código 6-26: Archivo *default_cotizacion.php* de la vista All

El resultado al consultar una compañía tiene que parecerse al presentado en el siguiente diagrama:

Ilustración 6-9: Vista de consultarCompañía del *frontend*

6.3.6.4 El modelo

En el *frontend* hay dos modelos, *consultarListaCompania y cotizacionDia*.

El primero devuelve una lista con todas las compañías del IBEX 35 que hay en la BD. Este modelo se construye igual que el modelo *all* del *backend*. Lo único que cambia es que la consultar obtiene las *companiasibex* publicadas.

```
$query = "SELECT * FROM #__companiasibex WHERE published = '1' ORDER
BY ticker ASC";
```

El segundo descarga un fichero de *YahooFinance* mediante la URL http://es.old.finance.yahoo.com/d/quotes.csv?s=".$ticker."&f=sl1d1t1c1ohgv&e=.csv.

El fichero contiene los datos de la compañía del IBEX 35 con identificador *$ticker.* Estos datos, devueltos en un vector, son la cotización actual en el mercado de valores, el tiempo, la variación, la apertura, el rango y el volumen. El código completo es el siguiente:

```
<?php
defined( '_JEXEC' ) or die( 'Restricted access' );
jimport('joomla.application.component.model');
class finanzasModelcotizacionDia extends JModel
{
	function getCotizacion($ticker)
	{
	$url="http://es.old.finance.yahoo.com/d/
	quotes.csv?s=".$ticker."&f=sl1d1t1c1ohgv&e=.csv";
	$datos = '';
	$fp = fopen($url , "r");
	while (!feof($fp)) {
		$datos .= fread($fp, 1000);
	}
	fclose($fp);
	$cotizaciones = preg_split("[;|\n]", $datos);
	return $cotizaciones;
	}
}
```

Código 6-27: Clase finanzasModelcotizacionDia

6.3.7 Añadir AJAX al componente

Mucha gente se pregunta cómo se puede utilizar AJAX en Joomla!. A primera vista parece complicado ya que cuando se crea un componente, éste siempre aparece dentro de un *template*, y la actualización de los datos supone cargar la página entera. En AJAX no se quiere que aparezcan dentro de *templates*, ya que solamente se quieren enviar ciertos datos en formato XML o JSON.

Para evitar que un componente aparezca dentro de un *template* existen las vistas Raw, que son vistas que se muestran sin la envoltura del *template.* Otra opción es la de enviar el resultado directamente desde la variable *task* del controlador AJAX.

Joomla! es muy productivo, y con AJAX no es una excepción. Su patrón MVC, permite disponer de un controlador para las peticiones AJAX en el componente, que será el que gestione los eventos asíncronos. En nuestro caso, se añade una función llamada **cotizaciónDia()** dentro del controlador **finanzasController** encargada de ello.

Por otra parte, Joomla! parece estar pensado para ser usado con JSON[5], ya que los métodos getList del modelo y loadObjectList del objeto JDatabase devuelven listas de objetos, que son ideales para ser impresas en formato JSON y procesadas con Javascript en la zona cliente.

Por eso, se usa la clase JSON que se encuentra en PHPClases. Para poderla usar, dentro del directorio **com_finanzas** de la estructura de ficheros del *frontend*, se crea la carpeta **include** y se añade el fichero **json.class.php** con el contenido descargado.

Además se añade a la vista el archivo JavaScript que contiene todas las funciones de creación del objeto Ajax, del envío, recepción y manipulación de la información asíncronamente. Nuevamente, se crea una carpeta **scripts** dentro de **com_finanzas** y se genera el archivo Javascript llamado **cotizacionTicker.js**.

Dentro de la función **display()** del archivo **view.html.php** de la vista **consultarCompañía** se sustituye la instrucción `echo"<meta http-equiv='refresh' content='60'>";` por `JHTML::script('cotizacionTicker.js','components/com_finanzas/scripts/',true);.`

Una vez generada toda la estructura de ficheros, se explica cómo se añade AJAX en el componente. Para ello se extenderá la implementación con la siguiente funcionalidad:

Anteriormente, se había añadido el comando HTML `<meta http-equiv='refresh' content='60'>` en la vista consultarCompañia para que se actualizara el valor de la última transacción en el mercado de valores de la compañía. Este comando refresca toda la página cada 60 segundos. Esto supone un reenvío innecesario de información repetida. Con Ajax, solamente se refrescará la zona donde se indica el valor de la transacción, sin la necesidad de una actualización de la página.

[5] **JSON**, acrónimo de *JavaScript Object Notation*, es un formato ligero para el intercambio de datos. JSON es un subconjunto de la notación literal de objetos de JavaScript que no requiere el uso de XML.

Siguiendo el patrón de funcionamiento de Ajax [Ver sección 6.2.5-Ajax] se implementan los siguientes pasos:

1- *El cliente produce algún evento, como el clic del mouse o la introducción de un dato, a través del navegador.*

En este caso, se añade la función setInterval dentro del archivo JavaScript para activar el evento actualizarValores() cada 60 segundos:

```
setInterval("actualizarValores()",60000);
```

Código 6-28: Paso 1 del patrón Ajax - Activación del evento

2- *Dicho evento es gestionado con JavaScript y es enviado al servidor asíncronamente mediante XMLHttpRequest.*

Para ello, se añade la función actualizarValores() dentro del archivo JavaScript encargada de crear el objeto Ajax y de generar la llamada asíncrona con la URL:

```
'index.php?option=com_finanzas&no_html=1&task=cotizacionDia&ticke
r='+ticker;
```

El parámetro no_htm se establece a 1 para indicar a Joomla! que no imprima el *template* entero, solamente la respuesta del componente. El parámetro cotizacionDia indica al controlador que ejecute la función con este nombre para obtener la información de la compañía el identificador ticker.

La dirección URL es enviada mediante la función *open* usando el método GET. Cuando el resultado esté listo, la función **cotización** lo recibe y lo manipula.

```
function actualizarValores(){
      oXML = AJAXCrearObjeto();
      var
      ticker=document.getElementById('componentheading').innerH
      TML;
      var url = 'index.php?option=com_finanzas&no_html=1
      &task=cotizacionDia&ticker='+ticker;
      oXML.open('GET', url);
      oXML.onreadystatechange = cotizacion;
      oXML.send(' ');
}
```

Código 6-29: Paso 2 del patrón Ajax – Envío asíncrono

Todas las funciones que crean un objeto Ajax son como la del siguiente código:

```
function AJAXCrearObjeto(){
  var obj;
  if(window.XMLHttpRequest) { // no es IE
    obj = new XMLHttpRequest();
  } else { // Es IE o no tiene el objeto
     try {
       obj = new ActiveXObject('Microsoft.XMLHTTP');
    }  catch (e) {
      alert('El navegador utilizado no está soportado');
    }
  }
  return obj;
}
```

Código 6-30: Creación del objeto Ajax

3- *El servidor procesa la petición y devuelve el resultado asíncronamente con XMLHttpResponse.*

Como ya se ha mencionado, el controlador, que se encuentra al lado del servidor, recibe la petición. En nuestro caso, el controlado incluye el fichero json.class.php y llama al modelo cotizacionDia para obtener la cotización

```
function cotizacionDia()
{
      include(JPATH_COMPONENT.DS."include".DS."json.class.php");
      $ticker = JRequest::getString('ticker','');
      $model =& $this->getModel("cotizacionDia");
      $cotizaciones = $model -> getCotizacion($ticker);
      $json = new JSON;
      echo $json->serialize($cotizaciones);
}
```

Código 6-31: Paso 3 del patrón Ajax – procesamiento de la información

4- *El resultado es gestionado con JavaScript para mostrar la información en las secciones de la página deseada.*

Inicialmente, se comprueba estado del objeto responsable de la conexión. Puede tomar cualquiera de los cinco valores:

- (0) No iniciado — Es el valor inicial de readyState.
- (1) Abierto — El método open ha tenido éxito.
- (2) Enviado — Se ha completado la solicitud pero ningún dato ha sido recibido todavía.
- (3) Recibiendo — Antes de recibir un mensaje body (en caso que exista). Todas las cabeceras HTTP han sido recibidas.
- (4) Respuesta completa — Todos los datos han sido recibidos.

Cuando toma el valor 4, se recibe el resultado y se añade en la casilla de la tabla que tiene como identificador valor. Según si el resultado obtenido es positivo o negativo se añade una imagen u otra.

```
function cotizacion(){
if (oXML.readyState  == 4)
{
      resultado = eval('(' + oXML.responseText + ')');
      document.getElementById('transaccion').innerHTML
      ='<b><big>'+resultado[1]+'&euro; </big></b></td>';
      document.getElementById('hora').innerHTML=resultado[2];
      if ( resultado[4].substring(0,1)=='-')
      {
      document.getElementById('valor').innerHTML='<font
      color="#FF0000"><img height="14" border="0" width="10"
      src="components/com_finanzas/imagenes/down_r.gif">
      '+resultado[4].substring(1,5)+'</font>';
      }
      else
      {
      document.getElementById('valor').innerHTML='<font
      color="#0000FF"><img height="14" border="0" width="10"
      src="components/com_finanzas/imagenes/up_g.gif">
      '+resultado[4].substring(1,5)+'</font>';
      }
}
}
```

Código 6-32: Paso 4 del patrón Ajax – Gestión del resultado

5- *Sin la necesidad de la recarga de la página, este ciclo comienza de nuevo cuando un usuario produzca un evento.*

En nuestro caso, se produce la llamada al evento cada minuto.

6.3.8 Creación de traducciones

Cuando se instala el paquete *Joomla_1.5.15-Spanish-pack_completo*, se generan tres idiomas para la aplicación. Cada paquete de idioma tiene sus propios archivos. Éstos se encuentran en dos carpetas llamadas *language*; una dentro de la carpeta *administrator* y la otra en el directorio raíz. Para cada idioma se genera un directorio nuevo dentro de *language.*

Estos directorios son nombrados según el código de idiomas ISO. Por ejemplo, el paquete de inglés Británico es en-GB, el castellano es-ES y el catalán ca-ES. Los archivos de idioma para cada extensión son introducidos dentro de estas carpetas. En la explicación se usará el idioma español.

Para empezar a traducir la interfaz del componente a uno de los tres idiomas se abre cualquiera de los ficheros ya implementados; por ejemplo *pfc->administrator->components->com_finanzas->views->all->tmpl->default.php*. Todas las salidas de datos están hechas con la sentencia PHP **JText::_('Name').**

Esta función es usada a través de Joomla! para traducir texto dentro de la interfaz de usuario. Si Joomla! no puede encontrar la traducción del texto introducido, simplemente muestra el texto base. Es decir, si en el diccionario no hubiera una entrada para *Name*, se mostraría por pantalla *Name* en vez de *Nombre*.

Hay palabras que se traducen automáticamente en todas las extensiones gracias al fichero **language/es-ES/es-ES.ini** que se carga siempre para las traducciones. Si hay palabras que no se han traducido automáticamente con *JText::__()*, hay que crear un nuevo fichero para el componente. Este fichero llamado **es-ES.com_finanzas.ini** se coloca dentro de la carpeta *pfc->administrator->languages->es-ES* para el *backend* y dentro de *pfc->languages->es-ES* para el *frontend*. Ahora simplemente hay que ir escribiendo en el fichero las palabras que se traducirán.

Por ejemplo si hay las salidas

<?php *echo JText::_('Close')?>*

<?php *echo JText::_('Companies for IBEX 35')?>*

<?php *echo JText::_('Country')?>*

en el archivo **es-ES.com_finanzas.ini** se introduce el texto de traducción:

```
CLOSE=Cierre
COMPANIES FOR IBEX 35=CompaÃ±Ãas del IBEX 35
COUNTRY=PaÃs
```

A pesar de que el texto introducido dentro de *JText::_()* está en mayúsculas y minúsculas, las definiciones en **es-ES.com_finanzas.ini** deben ir en mayúsculas. Si se intenta usar en la definición tanto mayúscula como minúscula, *JText* no leerá la entrada durante la traducción. Los signos de puntuación son aceptados en la traducción. Los acentos y los caracteres especiales son tratados siguiendo la nomenclatura utf-8.

En la siguiente tabla se muestra la relación de los caracteres especiales más usados:

letra	Á	é	í	ó	ú	Á	É	Í	Ó	Ú	ñ	Ñ	¡	¿
UTF-8	Ã¡	Ã©	Ã	Ã³	Ã°	Ã	Ã‰	Ã	Ã“	Ãš	Ã±	Ã‘	Â¡	Â¿

Tabla 6-1: Traducción a UTF-8

Finalizada la traducción, hay que ir a la sección *extensiones->gestión de idiomas* de la zona del *backend* y predeterminar el idioma español para la interfaz:

Ilustración 6-10: Gestor de idiomas

Si la traducción se ha realizado correctamente, aparecerán las palabras en castellano.

6.3.8.1 Depuración de un idioma

Muchas veces, cuando se está traduciendo un sitio web se nos olvida traducir alguna palabra. Joomla! permite que el administrador controle las palabras no traducidas. Hay que ir a **Sitio | Configuración global |** y clicar en **Sistema** del *backend*. A la derecha de la interfaz, hay una caja con el nombre **Parámetros de depuración de errores** con la opción **Depurar el idioma**. Se marca la opción de **Sí** y se guarda. Se puede comprobar que hay palabras escritas entre dos puntos *(p.e:·Nombre·)* y palabras entre "¿¿??" *(p.e: ¿¿Ticker??)*. Estas últimas son las que no se han encontrado en la traducción y son las que hay que añadir en el diccionario.

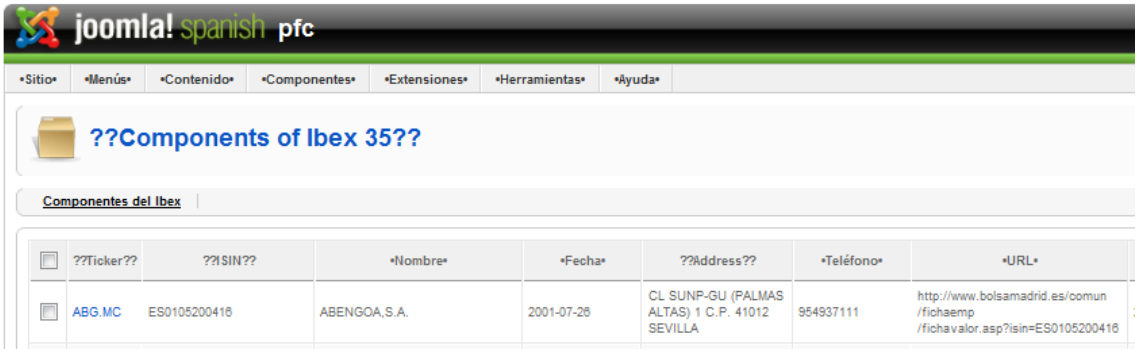


Ilustración 6-11: Ejemplo de depuración de idioma

6.4 Creación del paquete de instalación del componente

Una vez implementado el componente, se crea el paquete de instalación de éste para que pueda ser usado en todos los sitios web Joomla. En el apéndice B se explica cómo instalar el paquete que se creará a continuación.

Primero hay que generar la siguiente estructura de ficheros:

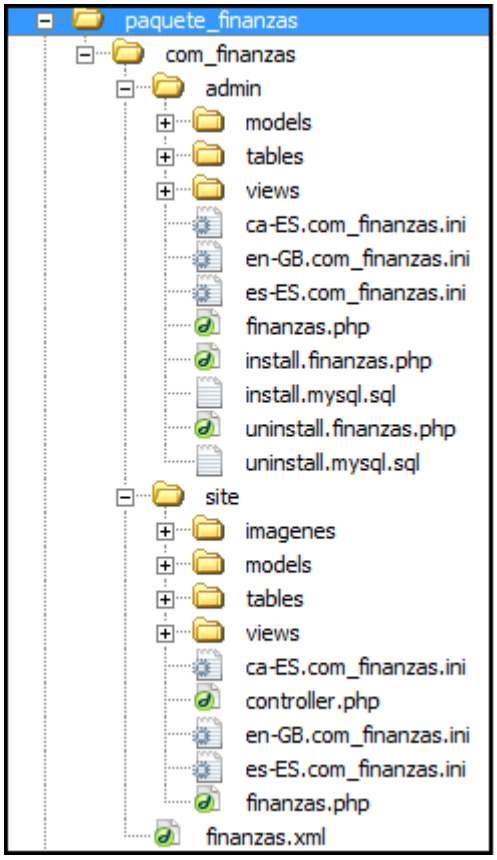


Ilustración 6-12: Estructura de ficheros del paquete instalador

Se crea la carpeta *admin*, con el contenido del *backend* del componente, la carpeta *site*, con el contenido del *frontend* y el archivo xml instalador *finanzas.xml*.

Dentro de la carpeta admin, se crean 4 archivos nuevos:

- **Install.finanzas.php:** Muestra el mensaje de instalación.
- **Install.mysql.sql:** Contiene las sentencias SQL necesarias para crear tablas e introducir datos en la BD.
- **Uninstall.finanzas.php:** Muestra el mensaje de desinstalación.
- **Uninstall.mysql.sql:** Contiene las sentencias para eliminar la información del componente en la BD.

Además, se añaden los archivos necesarios para la gestión de idiomas.

A continuación se explicará y se mostrará el código de todos los archivos nuevos creados:

6.4.1 Finanzas.xml

Este archivo es el más importante en la instalación del componente. Su contenido sirve para informar a Joomla! donde se encuentran todos los ficheros implementados.

Su contenido es:

```
<?xml version="1.0" encoding="utf-8"?>
<install type="component" version="1.5.0">
 <name>Finanzas</name>
 <author>Antoni Aguilo Tarre</author>
 <creationDate>Abril 2010</creationDate>
 <copyright>(C) 2010</copyright>
 <authorEmail>antoni.aguilo@gmail.com</authorEmail>
 <version>1.5.0</version>
 <license>MIT</license>
 <description>Portafolio de finanzas</description>
 <installfile>install.finanzas.php</installfile>
 <uninstallfile>uninstall.finanzas.php</uninstallfile>
 <install>
  <sql>
   <file driver="mysql" charset="utf8">install.mysql.sql</file>
  </sql>
 </install>
 <uninstall>
  <sql>
   <file driver="mysql" charset="utf8">uninstall.mysql.sql</file>
```

```
  </sql>
 </uninstall>
 <languages folder="site">
  <language tag="es-ES">es-ES.com_finanzas.ini</language>
  <language tag="es-ES">ca-ES.com_finanzas.ini</language>
  <language tag="en-GB">en-GB.com_finanzas.ini</language>
 </languages>
 <files folder="site">
  <filename>finanzas.php</filename>
  <filename>controller.php</filename>
  <folder>imagenes</folder>
  <folder>models</folder>
  <folder>tables</folder>
  <folder>views</folder>
 </files>
 <administration>
  <menu>Portafolio de finanzas</menu>
  <submenu>
   <menu link="option=com_finanzas">Companias del IBEX</menu>
  </submenu>
  <languages folder="admin">
   <language tag="es-ES">es-ES.com_finanzas.ini</language>
   <language tag="es-ES">ca-ES.com_finanzas.ini</language>
   <language tag="en-GB">en-GB.com_finanzas.ini</language>
  </languages>
  <files folder="admin">
   <folder>models</folder>
   <folder>tables</folder>
   <folder>views</folder>
   <filename>install.mysql.sql</filename>
   <filename>uninstall.mysql.sql</filename>
   <filename>finanzas.php</filename>
   <filename>install.finanzas.php</filename>
   <filename>uninstall.finanzas.php</filename>
  </files>
 </administration>
</install>
```

Código 6-33: Archivo finanzas.xml

Todos los ficheros se listan en formato XML.

Debajo de las etiquetas informativas de la instalación, se encuentran las etiquetas <installfile> y uninstallfile>. Sirven para indicar qué archivos se ejecutan en el momento de la instalación y la desinstalación.

Después hay las etiquetas <install> y uninstall>. Éstas contienen las etiquetas <sql> y <file> y son usadas para añadir el archivo de consultas SQL para la BD.

Para los componentes hay que distinguir entre los archivos *backend* y los *frontend*. La etiqueta <files folder=”admin”> que se encuentra dentro de <administration> lista los ficheros del *backend*. Dentro de administration también se listan los archivos de idiomas y se crea un menú en el *backend* para el componente.

Todos los ficheros de implementación y de lenguaje del *frontend* se listan dentro de la etiqueta <files folder=”site”>.

6.4.2 Install.finanzas.php

Este fichero muestra un mensaje en el *backend* cuando se ha instalado el componente correctamente. Su contenido es:

```
<?php
defined('_JEXEC') or die('Restricted access.');
function com_install(){
	?>
	<div class="header"><?php echo JText::_('Finanzas has been installed successfully')?></div>
	<?php
}
?>
```

Código 6-34: Archivo install.finanzas.php

6.4.3 Uninstall.finanzas.php

Este fichero muestra un mensaje en el *backend* cuando se ha desinstalado el componente correctamente. Su contenido es:

```
<?php
defined('_JEXEC') or die('Restricted access.');
function com_uninstall(){
	?>
	<div class="header"><?php echo JText::_('Finanzas has been uninstalled successfully')?></div>
	<?php
}
?>
```

Código 6-35: Archivo uninstall.finanzas.php

6.4.4 Install.mysql.sql

Para poder usar las compañías del IBEX 35 en el componente se tienen que introducir los datos de éstas en la BD. Para lograr este objetivo, se abre el fichero *install.mysql.sql* y se introduce el siguiente código:

```
CREATE TABLE IF NOT EXISTS `#__companiasibex` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `ticker` varchar(10) NOT NULL,
  `isin` varchar(12) NOT NULL,
  `nombre` varchar(100) NOT NULL,
  `descripcion` text NOT NULL,
  `fecha_inicio` date NOT NULL,
  `domicilio` varchar(50) NOT NULL,
  `telefono` int(13) NOT NULL,
  `url` varchar(100) NOT NULL,
  `uptoday` tinyint(1) NOT NULL,
  `published` tinyint(1) NOT NULL,
  `capital_permitido` bigint(20) NOT NULL,
  `numero_acciones` int(11) NOT NULL,
  `pais` varchar(30) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=MyISAM  DEFAULT CHARSET=utf8 AUTO_INCREMENT=37 ;

INSERT INTO `#__menu` (`id`, `menutype`, `name`, `alias`, `link`,
`type`, `published`, `parent`, `componentid`, `sublevel`, `ordering`,
`checked_out`, `checked_out_time`, `pollid`, `browserNav`, `access`,
`utaccess`, `params`, `lft`, `rgt`, `home`) VALUES
(67,    'mainmenu',    'CompaÃ±Ãas    del    IBEX    35',    'compibex',
'index.php?option=com_finanzas&view=consultarListaCompanias',
'component', 1, 0, 36, 0, 10, 0, '0000-00-00 00:00:00', 0, 0, 0, 0,
'page_title=\nshow_page_title=1\npageclass_sfx=\nmenu_image=-
1\nsecure=0\n\n', 0, 0, 0);
```

Código 6-36: Archivo install.mysql.sql

La primera consulta, crea la tabla jos_*companiasibex* con todas variables explicadas en el ejemplo de la creación del componente. Se ha añadido el calificador IF NOT EXISTS para que no se genere un error si la tabla ya existe en la BD. Además, se usa #__ como prefijo de la tabla para que Joomla! cree el prefijo que esté en la configuración.

La segunda consulta, inserta en el menú principal un enlace llamado ***Compañías del IBEX 35***. Concretamente se llama a la URL "*index.php?option=com_finanzas&view=consultarListaCompanias"*. Como se ha visto en la implementación, se mostraría la lista de compañías del IBEX 35 al clicar este enlace.

6.4.5 Uninstall.mysql.sql

El archivo de desinstalación simplemente eliminará la tabla *companiasibex* y el enlace al componente. Por lo tanto hay que introducir el siguiente código:

```
DROP TABLE #__companiasibex;

DELETE FROM #__menu WHERE id = 67;
```

Código 6-37: Archivo uninstall.mysql.sql

Capítulo 7

Pruebas

7.1 Introducción

Los procesos que se siguen para verificar la calidad de un producto software se llaman pruebas software, o *testing* en inglés. Se usan para encontrar posibles fallos o *bugs* en la implementación y para garantizar la calidad del programa. En informática se distingue entre errores de programación (*bugs)* y defectos de forma. Un error de programación es un error sistemático del programa. Por el contrario, en un defecto de forma el programa no se realiza lo que el usuario desea.

En resumen, el *testing* es una fase en el desarrollo software que consiste en probar las aplicaciones construidas y verificar las fases anteriores. Así, se ejecuta el programa y mediante técnicas experimentales se intenta descubrir los errores que tiene. Es importante saber que el *testing* puede probar la presencia de errores, pero no la ausencia de ellos. Durante el proceso se desarrollo, se crean versiones del programa, llamadas *alpha,* donde el programa incompleto dispone de funcionalidades básicas para poder ser testeado. Pero cuando realmente se verifica la totalidad del programa es en la puesta en funcionamiento: Muchos usuarios ejecutando la aplicación permite que ésta se exprima al máximo.

La calidad se determina a través de unas pruebas que permitan comprobar el grado de cumplimiento respecto las especificaciones descritas al inicio del proyecto.

Otra práctica, más común en proyectos pequeños, es que las pruebas se realicen a medida que el desarrollo avanza y sean continuas hasta que éste finaliza.

Se han encontrado muchos planteamientos a la hora de realizar el proceso de pruebas de software. Se ha llegado a la conclusión, que para verificar productos complejos es más efectivo realizar un proceso de investigación en lugar de seguir los pasos predeterminados en los libros de desarrollo software.

En este proyecto, se ha elegido hacer una combinación de ambas prácticas. A medida que se iban añadiendo casos de uso en el sistema, se iban verificando. Esta forma de probar se llama **prueba unitaria**: Consiste en comprobar cada uno de los bloques de código por separado. La idea es escribir casos de prueba para cada función o método en el componente para que se traten de forma independiente del resto.

Una vez, todos los casos de prueba se habían verificado, se aseguraba que todo el conjunto interactuara correctamente con las **pruebas de integración**: Se realizan en el ámbito del desarrollo de software una vez son aprobadas las pruebas unitarias. Lógicamente, estas pruebas son las últimas que comprueban la ejecución del programa y preceden a las pruebas de validación.

Validar un sistema simplemente consiste en verificar la pregunta: ¿Es lo que realmente quiere el cliente?

En los siguientes apartados se describen las pruebas de integración realizadas. Se han probado todas las funciones del programa conjuntamente en un entorno local y en un servidor externo.

7.2 Entorno de pruebas

La aplicación se ha probado desde los sistemas operativos Windows Vista y Linux utilizando los navegadores Mozilla Firefox e Internet Explorer. Hay que destacar que ambos navegadores tienen un funcionamiento interno distinto. Esto solamente afecta en el diseño de la página.

7.3 Pruebas realizadas

7.3.1 Instalación del componente

Con el archivo del código fuente (*Joomla_1.5.15-Spanish-pack_completo)* se ha instalado Joomla! en un servidor desde cero tal como se explica en el apéndice A. Seguidamente, se ha instalado el proyecto entero con el archivo instalador com_finanzas.zip. Este proceso, tarda un poco porque hay muchos datos introducidos en la base de datos Joomla!. Para que la instalación sea satisfactoria los archivos del paquete instalador deben tener la estructura definida en el apéndice B.

Este test funcionó correctamente en ambos servidores y sistemas operativos. Automáticamente, el sistema ya está dentro del sitio web y está listo para ser ejecutado.

7.3.2 Gestión Administrativa *(backend)*

Para la gestión administrativa, hay que identificarse en el *backend* de Joomla! con el nombre de usuario y contraseña de un SuperAdministrador. Dentro de este espacio se han realizado cuatro pruebas:

Creación compañías

Se ha rellenado el formulario de compañía y se ha comprobado que los datos se hayan introducido dentro de la tabla *jos_compania* de la base de datos. En el formulario se comprueba que los valores del formulario ticker, ISIN, nombre y mercado de valores no sean nulos. Es responsabilidad del usuario la introducción de datos erróneos tales como un nombre incorrecto; por ejemplo ABEE.MC en vez de ABE.MC.

Automáticamente, al instalar el componente se han introducido las 35 compañías actuales del IBEX 35 en el sistema. La creación de una compañía ha sido satisfactoria.

Eliminación compañías

Se ha seleccionado una o varias compañías y se han eliminado. En la tabla *jos_companias* de la base de datos se han eliminado sus respectivos registros.

Modificación compañías

Se ha elegido cualquier compañía de la lista y se han modificado varios valores. Los cambios se han visto reflejados en la base de datos. Esta prueba se ha realizado para cada compañía.

Publicación/Despublicación de compañías

Las compañías pueden ser publicadas o no en el *frontend* del sistema. Desde la zona administrativa se han publicado solamente algunas compañías. Efectivamente, en el *frontend* se han visto solamente aquellas compañías que estaban publicadas. Al ser un sistema que de momento solamente está pensado para el IBEX 35, una compañía que no forme parte de este mercado no se verá en la aplicación. Una ampliación es que todo tipo de compañía pueda ser vista.

7.3.3 Gestión del *frontend* con un usuario

En el *frontend* hay que diferenciar entre dos bloques de funcionalidades: El primer bloque consiste en los elementos que un usuario no registrado puede ver en la aplicación. Mientras que el segundo son todas las acciones realizadas desde una zona privada.

7.3.3.1 Zona no registrada

En la zona no registrada, un usuario solamente puede hacer consultas. Por lo tanto, estas pruebas son solamente verificar que los datos consultados corresponden a la petición del usuario:

Consulta del inicio de la aplicación

La aplicación muestra dos artículos relacionados con la aplicación. El primero explica qué es el IBEX 35 tal como se ha documentado en la presente memoria. El segundo, corresponde a la explicación de los términos financieros usados en la aplicación: qué es el stop-*loss*, la varianza y otras definiciones.

Consulta de enlaces web

Se ha comprobado que los enlaces web que se proponen corresponden realmente a las páginas web solicitadas.

Consulta de noticias

Se ha consultado el *feed* de noticias durante los distintos días de una semana y se ha comprobado que efectivamente, se muestran noticias actuales del sitio consultado. En esta sección se encuentran noticias de la página de Eleconomista y de Yahoo Finanzas.

Consulta de la cotización actual

Al consultar la cotización actual se ha mostrado una tabla con los datos actuales de las 35 compañías del IBEX 35. Si se deja la página unos minutos se puede comprobar que los datos se van actualizando automáticamente sin la completa carga de la página gracias a AJAX. Los datos, recogidos de la base de datos de Yahoo, tienen una demora de 15 minutos. La carga inicial de esta página es un poco lenta porque se realiza el cálculo complejo de la volatilidad.

Consulta de precios históricos

En un formulario, se ha introducido una fecha inicial y una final, y se ha comprobado que los valores obtenidos corresponden a dicho intervalo. Los valores son mostrados en una gráfica y en una tabla. Si la fecha inicial es mayor a la fecha final, el sistema muestra un mensaje error.

El sistema tiene almacenados unos valores en la base de datos. Si no encuentra los valores que el usuario solicita, los recoge de Yahoo y los introduce en la tabla *jos_datoshistoricos* de su base de datos. Se ha comprobado que se realiza dicha acción dejando la base de datos vacía y haciendo tales peticiones. Inicialmente, al instalar el componente, se introducen la mayoría de valores en la base de datos, para que los valores consultados se recojan de la base de datos y así ser mostrados más rápidamente.

Consulta de una compañía

Se ha comprobado, compañía por compañía, que al consultarlas se muestra su perfil y su cotización actual. Todos estos datos se encuentran dentro de la tabla *jos_compania.*

7.3.3.2 Zona registrada

Inicialmente se ha comprobado el módulo de los registros de usuarios que Joomla! dispone por defecto. Éste permite registrar a un usuario, identificarse, solicitar su contraseña y su nombre de usuario. En localhost, el sistema de envío de correo no funciona, por lo que se han tenido que modificar los valores de *block* y *sendEmail* de la tabla *jos_user* de la base de datos para poder trabajar con usuarios. Estos valores, bloquean a un usuario hasta que no haya confirmado su registro a través un correo. Los registros han funcionado correctamente, pudiendo añadir tantos usuarios como ha querido. La tabla *jos_cliente* complementa la tabla *jos_users* con más atributos del usuario. En la zona administrativa se proporciona un formulario para que el usuario introduzca los datos de su perfil. Se han ido añadiendo varios valores al formulario y se ha comprobado que funciona correctamente. Es responsabilidad del usuario introducir datos incorrectos tales como una dirección o un teléfono inexistentes.

Una vez registrado un usuario, se comprueba que se pueda identificar correctamente. Dentro de su zona privada se encuentran las funcionalidades que se muestran a continuación:

Rellenar formulario Compra (título activo)

Se muestra un formulario donde hay una lista desplegable, un calendario y dos cajas de introducción de los datos cantidad comprada y precio de Stock. Se ha comprobado que el desplegable contiene las 35 compañías del IBEX 35 y que el calendario se genera correctamente. Además, se ha rellenado el formulario con distintos valores para verificar la reacción del sistema:

- Se han introducido cantidades negativas o no numerales: El sistema muestra el mensaje de error.
- Se han introducido precios negativos o no numerales: El sistema muestra el mensaje de error.
- El formato de la fecha introducida no es válido: Se muestra el mensaje de error.

Además, se permite que el usuario consulte el precio máximo y mínimo de stock de la compañía seleccionada para una fecha concreta. Se ha consultado en otros sitios web este intervalo es el correcto. Es responsabilidad del usuario introducir un precio que se encuentre dentro del intervalo seleccionado.

Una vez rellenado el formulario con valores correctos, se ha comprobado que se almacenen correctamente dentro de la tabla *jos_tituloactivo* y que se muestren en una tabla ordenada por fecha.

Modificar título activo

En la misma interfaz donde se rellenan los formularios para los títulos activos, se permite modificar sus valores. Para ello, se ha clicado a la imagen *"editar"* y se han introducido los datos nuevos en una nueva interfaz. De nuevo, se ha intentado introducir datos incorrectos para comprobar que el sistema los comprueba correctamente. Finalmente, se han apreciado los cambios registrados en la tabla.

Borrar título activo

Al hacer clic a eliminar un título activo, se ha verificado que el registro se haya eliminado.

Rellenar formulario Venta (título histórico)

Rellenar un formulario de venta tiene la siguiente peculiaridad: Para poder vender una acción se ha tenido que comprar previamente. El sistema muestra al usuario un desplegable con todos los títulos activos del usuario. A partir de estos, es cuando se puede rellenar el formulario de venta. Se han ido introduciendo distintos datos para comprobar que el sistema los verifica correctamente:

- Se han introducido cantidades negativas o no numerales: El sistema muestra el mensaje de error.
- Se han introducido precios negativos o no numerales: El sistema muestra el mensaje de error.
- Se han vendido parte de las acciones de un título activo: El sistema crea un registro en la tabla *jos_titulohistorico* con los valores del formulario y resta la cantidad de acciones compradas del título activo en la tabla *jos_tituloactivo*.
- Se han vendido todas las acciones de un título activo: El registro del título activo se elimina de la tabla *jos_tituloactivo* y se crea uno nuevo en *jos_titulohistorico* con los valores introducidos.
- El formato de la fecha introducida no es válido: Se muestra el mensaje de error.

Como en el formulario de compras, se permite al usuario consultar el intervalo de precios de stock de la compañía para el día seleccionado.

Modificar título vendido

Las pruebas realizadas para modificar un título vendido son las mismas que las gestionadas para los títulos activos.

Borrar título vendido

Las pruebas realizadas son las mismas que para los títulos activos.

Consultar datos del portafolio

Para consultar el estado actual del portafolio, se han creado varios escenarios distintos y así ver como se muestran los resultados:

(1) Hay títulos activos de **una** compañía:

El sistema muestra una tabla con los valores de los títulos de la compañía junto con el *stop-loss*, el precio actual y la rentabilidad simple acumulada de cada título. Indica que no hay títulos vendidos y que por lo tanto, aún no hay ganancias.

(2) Hay títulos vendidos de **una** compañía:

La tabla de títulos activos desaparece mostrándose un mensaje indicador de que no hay títulos activos. Se muestra una tabla con los beneficios totales de los títulos vendidos de esa compañía, que se puede desplegar para ver los detalles de cada título vendido. La ganancia total coincide con los beneficios totales de la compañía ya que solamente se han vendido títulos de una compañía.

(3) Hay títulos activos de **varias** compañías:

El sistema muestra una tabla con los valores de los títulos de las compañías junto con el *stop-loss*, el precio actual y la rentabilidad simple acumulada de cada título. La tabla está ordenada según la fecha de la compra. Además, indica que no hay títulos vendidos y que por lo tanto, aún no hay ganancias.

(4) Hay títulos vendidos de **varias** compañías:

La tabla de títulos activos desaparece mostrándose un mensaje indicador de que no hay títulos activos. Se muestra una tabla con un listado de todas las compañías de las cuales se han vendido títulos. Para cada compañía se muestra el beneficio total, que se puede desplegar para ver los detalles de cada título vendido. La ganancia total del portafolio coincide con la suma de los beneficios totales de todas las compañías.

(5) Hay título activos y títulos vendidos de **una** compañía:

El sistema muestra dos tablas: Una con los títulos activos y otra con los títulos vendidos. El comportamiento de cada tabla es el descrito en los puntos (1) y (2) respectivamente.

(6) Hay títulos activos y títulos vendidos de **varias** compañías:

El sistema muestra las dos tablas mencionadas en el punto anterior y además tiene un comportamiento igual a (3) y (4).

En los casos donde hay títulos activos se puede modificar el valor del porcentaje usado para el cálculo del *stop-loss* y la fecha para la consulta de la rentabilidad del portafolio de ese día. Se han ido variando estos valores para ver que los cálculos de realizan correctamente. Para el cálculo de la rentabilidad, cuando se indica una fecha donde el mercado no proporciona valores (fin de semana) se muestra un mensaje de aviso.

Además, se indica el número de títulos activos, vendidos y totales. En todos los casos se ha comprobado con la información de las tablas que el cálculo se realiza correctamente.

7.3.4 Gestión de usuario (*frontend*) con varios usuarios

Todas las pruebas anteriores se han efectuado con varios usuarios y se ha comprobado que cada uno tiene sus datos íntegros. Además, el sistema se ha mostrado seguro en todo momento y no se ha podido acceder a datos de otro usuario. Como en todo sistema multiusuario, si un usuario usa una contraseña poco segura, el sistema no se puede responsabilizarse de ello.

Capítulo 8

Conclusiones y mejoras futuras

8.1 Introducción

Este apartado concluye la memoria del proyecto con una valoración del trabajo realizado durante los cincos meses del proyecto, una conclusiones finales y las posibles ampliaciones futuras que se pueden realizar.

8.2 Conclusiones y valoración de los objetivos

Este proyecto ha servido para comprobar la grandeza de los CMS de código abierto. No solamente permiten la gestión automática de sitios web, sino que además permiten ampliar tu propio sitio con cualquier componente personalizado. Gracias a la comunidad Joomla!, un usuario con conocimientos de programación puede crear sus propios sistemas complementados con las extensiones ya existentes en Joomla!.

Concretamente, el componente implementado puede servir como herramienta para inversores. Al tratarse de un portafolio, cualquier usuario puede simular diferentes carteras de valores para ver como se producen ganancias o pérdidas. Esto permite al usuario, tener una idea de la tendencia de los mercados de valores.

Cabe destacar que, a simple vista, la aplicación parece solamente una serie de páginas web implementadas en PHP. En realidad es mucho más que esto: Se trata de una estructura de ficheros, que cumplen las normas y las seguridad que se pide en Joomla!, que se pueden instalar en cualquier sitio web Joomla! y adaptarse a las necesidades del usuario.

En los objetivos iniciales quedaron claramente detallados los tres grandes objetivos generales que se tenían de este proyecto. Por eso, se valorarán los objetivos específicos de cada uno:

1. *Crear un portafolio de finanzas para que pueda ser utilizado en cualquier página web gestionada con Joomla!*

   1.1. Aprender aspectos financieros

   Objetivo logrado: Al tratarse de una interfaz que indica al usuario como va su cartera de valores, se han tenido que aprender las operaciones que se realizan en los mercados financieros y los términos que usan para poder lograr todos los demás objetivos.

   1.2. Aprender el funcionamiento de la bolsa

Objetivo logrado: Se ha aprendido el funcionamiento del IBEX 35 en general. En el capítulo 2, se resumen todos los conocimientos adquiridos.

1.3. Conocer la evolución de los sistemas de información

Objetivo logrado: Al crear una interfaz que se instala en un sistema de gestión de contenidos, se ha tenido que aprender cómo han evolucionado los sistemas de información de los sitios web. Concretamente, en el apartado 3.2 se ha explicado la evolución de las páginas web.

1.4. Conocer el entorno Joomla!

Objetivo logrado: Para trabajar sobre cualquier sistema requiere primero tener un buen conocimiento de éste. El libro *Joomla! 1.5: A User's Guide: Building a Successful Joomla!* [N08] me permitió aprender todo el funcionamiento a nivel de usuario. El capítulo 3 hace un resumen de las características principales de Joomla!

1.5. Implementar y explicar componentes con Joomla!

Objetivo logrado: Este es uno de los objetivos más importantes del proyecto. Saber implementar un componente requiere conocer la estructura interna del sistema. En el capítulo 5 se muestra la arquitectura y la estructura de la base de datos del sistema Joomla!. Además el capítulo 6 sirve como pequeño manual para futuros usuario para saber los pasos que hay que seguir para crear una extensión desde cero: Estructura de ficheros, creación de base de datos…

1.6. Explicar cómo se implementan componentes.

Objetivo logrado: como se ha mencionado en el punto anterior, el capítulo 6 cumple este objetivo.

1.7. Añadir elementos a tu propio componente

Objetivo logrado: En la interfaz se han añadido los módulos de registro de usuarios y el de gestión de noticias. Este objetivo permite enriquecer mucho más el componente que se está implementado ya que se pueden añadir muchísimas de las funcionalidades que Joomla! dispone.

1.8. Especializarse en los lenguajes PHP, AJAX MySQL

Objetivo logrado: La implementación de Joomla! se basa en el uso de PHP y MySQL para la gestión del contenido y los datos respectivamente. Gracias al libro *PHP and MYSQL Development* se han adquirido muchos conocimientos de dichos lenguajes. Al usar contenido dinámico, se han aprendido los aspectos básicos de AJAX.

2. *Aprender a realizar un proyecto software de tamaño medio aplicando todas sus fases: análisis, diseño, implementación, pruebas y cálculo de costes.*

2.1. Conocer las fases de un proyecto software

Objetivo logrado: Es uno de los primeros objetivos que se tienen que cumplir para la realización del proyecto. Cuando existe un periodo temporal cerrado, hay que crear una planificación donde todas las fases se vean bien diferenciadas. Al tratarse de un proyecto software donde se han simulado los roles de analista, diseñador e implementador, se han tenido que aprender cada una de las fases.

2.2. Conocer herramientas UML

Objetivo logrado: Gracias al programa Visual Paradigm se han creado todos los diagramas utilizados en el análisis y el diseño de la aplicación.

2.3. Realizar una buena fase de pruebas

Objetivo logrado: Para conseguir una aplicación con buenos resultados, se requiere realizar una buena fase de pruebas. A pesar de que en numerosos proyectos existen muchas versiones de la aplicación, en este caso se ha intentado ajustar lo máximo posible las pruebas para que el resultado sea favorable.

3. *Aprender a realizar la documentación de un proyecto final de carrera.*

3.1. Escribir una memoria que permita al lector entender el trabajo realizado durante el proyecto.

Objetivo logrado: Valorarse uno mismo sobre si ha logrado este objetivo es algo complicado. Al tener que redactar un capítulo donde se explica cómo se implementa un componente en Joomla!, uno de los principales objetivos, la documentación se ha extendido un poco. Personalmente, he intentado ser lo más claro y directo posible para que el lector siga la lectura.

3.2. Hacer una presentación clara y que se entienda.

Objetivo logrado: La valoración es la misma que la hecha en el punto anterior.

8.3 Ampliaciones

Este proyecto se ha centrado en la creación de un portafolio para el IBEX 35. En el futuro, se podría ampliar la aplicación para que abarcara más mercado de valores. En el análisis se creó una clase llama Mercado de Valores para permitir dicha ampliación.

Sería interesante también, que un usuario no solamente consultara información de los valores y de sus acciones, sino que además, el sistema le informara de si es arriesgado comprar o vender en ese momento. Se podrían crear algoritmos que calcularan, mostrando el porcentaje de acierto o de riesgo, qué acciones serían aconsejables comprar o vender.

Una ampliación interesante a tener en cuenta, seria la creación de un foro y un sistema de envío de correo como se comenta en el capítulo 4 *-Análisis.* Intercambiar información entre usuarios especializados en el tema siempre es enriquecedor para la persona que usa la aplicación.

Toda la implementación se ha realizado en PHP. Este lenguaje se asemeja cada vez más a los lenguajes de alto nivel como Java o C++. A pesar de todo, es interesante ampliar el proyecto para que todas las funciones de cálculo sean realizadas por una aplicación externa, implementada en un lenguaje de alto nivel. Así se conseguiría una velocidad mayor en la carga del contenido.

Finalmente, otra ampliación interesante sería que el usuario pudiera descargar en formato *pdf* o *excel* la información que consulta del sitio web. La mayoría de información viene dada en tablas o en gráficas fáciles de formatear.

8.4 Valoración personal

Acabar un proyecto final de carrera supone la finalización del paso por la universidad. Por eso, me gustaría hacer una valoración más personal de lo que ha supuesto para mí realizar este proyecto.

Al comenzar la carrera de informática, uno no tiene claro que es lo que quiere hacer. La informática abarca tantos conceptos que al principio uno se siente desconcertado y mareado. Realizar este proyecto me ha ayudado a tener una idea clara de lo que quiero hacer en el futuro. Me gustaría trabajar en el mundo de los sistemas de información, y en concreto, en la creación de proyectos software. Me ha gustado mucho, la secuencia de fases que componen el desarrollo de un proyecto.

He tenido que estar lejos de la Facultad de Informática durante el desarrollo del proyecto. Esto conlleva muchas más complicaciones de lo que uno piensa al principio. Los recursos técnicos que uno dispone, al no estar en la universidad, son mínimos: Solamente he dispuesto de un ordenador portátil, y toda la información, la he tenido que conseguir a través de libros comprados o de internet. Por eso, me ha resultado más complicado de lo normal solucionar dudas o consultas. De todas formas, tener que espabilarse uno mismo, estando tantas horas sólo detrás de un ordenador, ayuda mucho a reflexionar y a madurar las cosas. Además, el director del proyecto me ha ayudado en todo momento.

Por otro lado, la ejecución de este proyecto me ha servido de ayuda para ampliar conocimientos adquiridos durante mis estudios en las distintas asignaturas de la carrera tales como Ingeniería de requisitos, Sistemas de información o Proyecto de Ingeniería del Software y Base de Datos.

En general, me siento orgulloso del trabajo logrado, al cumplir todos los objetivos iniciales, y de los conocimientos que he adquirido durante estos 5 meses.

Capítulo 9

Referencias

9.1 Bibliografía

[A06] Arlow IlaNeustadt, J. *UML 2.* Ed.Anaya Multimedia. 2006.

[B07] Blakeley Silver, T. *Joomla! Template design*. Pakt Publishing, Junio de 2007.

[B08] Ballard, P. *Sams Teach Yourself Ajax, JavaScript, and PHP All in One*. Sams Publishing. Junio de 2008.

[G08] Graf, H. *Building Websites with Joomla! 1.5*. Pakt Publishing, 2008.

[K07] Kennard,J. *Building Websites Mastering Joomla 1.5 Extension and Framework Development*. Pakt Publishing, Noviembre de 2007.

[L03] Larman, C. *UML y Patrones:Introducción al análisis y diseño orientado a objetos.* Ed.Prentice Hall. 2003

[L07] LeBlanc, J. *Learning Joomla 1.5 Extension Development*. Pakt Publishing, Mayo de 2007.

[N08] M.North, B .*Joomla! 1.5: A User's Guide: Building a Successful Joomla! Powered Website*. Prentice Hall. Mayo de 2008.

[R07] Rahmel, D. *Professional Joomla!*. Wiley Publishing, 2007.

[SP07] Stevens, P. Pooley, R. *Utilización de UML en ingeniería del sotfware con objetos y componentes*. Ed. Pearson Addison Wesley, 2007.

[T07] Trevejo Alonso, J.A. *Joomla! Para principiantes.* 13 de marzo de 2007.

[W09] Welling,L. Thomson, L. *PHP and MYSQL Development.* Pearson Education,Inc. Setiembre 2009.

9.2 Fuentes web

9.2.1 Documentos web

[B10] *Technical Regulations for the composition anf calculation of the Sociedad de Bolsas, indexes*. Sociedad de Bolsas, 2010 [ref. Marzo de 2010].

Disponible en;

http://www.sbolsas.com/dgata/normdef-in.pdf

[K09] Dr.A.A.Kotzé, *Certain Uncertain Volatility Constantly*. Absa Corporate and Merchant Bank, 2001 [ref. Octubre de 2009].
Disponible en;

http://www.quantonline.co.za/documents/Volatility2Vol.pdf

[R09] Reynoso, G. *Introducción a Joomla! CMS*. 14 de Abril de 2009 [ref. 9 de Octubre de 2009].

Disponible en;

http://www.scribd.com/doc/19616524/1-Introduccion-a-Joomla-Cms

[S09] Serrats,C. *Programació de components MVC per Joomla! 1.5.x*. CESI informàtica i comunicacions, 2009 [ref. 18 de Marzo de 2009].
Disponible en;

http://downloads.joomlacode.org/frsrelease/3/7/6/37642/JoomlaSpanish.pdf

9.2.2 Sitios web

BOLSAMADRID: Portal web oficial de la Bolsa de Madrid. Contiene toda la información de los mercados de valores, los índices actuales e históricos y los perfiles de las compañías actuales de la bolsa Española.

Disponible en:

http://www.bolsamadrid.es/esp/portada.htm

ELECONOMISTA: Portal financiero donde se puede encontrar noticias sobre mercados y cotizaciones, divisas, IBEX 35, fondos de inversión y actualidad de valores.

Disponible en:

http://www.eleconomista.es/

HIGHCHARTS: Librería HIGHCHART implementada en JavaScript. Contiene explicación de cómo añadir gráficas interactivas en la aplicación web. Actualmente soporta gráficas de barras, columnas, líneas, áreas, circulares y de superficie.

Disponible en:

http://www.highcharts.com/

INFOMERCADOS: Sitio Web que contiene noticias y las cotizaciones de los mercados de valores más importantes.

Disponible en:

http://www.infomercados.com/

JOOMLA. Sitio web Joomla!. Proporciona un punto central de información , discusión y colaboración entre los usuarios Joomla!. Entre ellos se incluyen administradores de sistemas, profesores, investigadores, diseñadores y desarrolladores.

Disponible en:

http://www.joomlaspanish.org/

NOSOLOCODIGO: Blog relacionado con la actualidad de Joomla!, CSS, JavaScript y todos los lenguajes de programación usadoes en las aplicaciones web. Contiene una entrada muy interesante sobre la implementación de componentes y módulos Joomla!.

http://www.nosolocodigo.com/

PHPNET: Documentación sobre PHP, contiene explicaciones de las características de PHP, e información complementaria. Todas las funciones incluyen explicaciones y ejemplos.

Disponible en:

http://www.php.net/manual/es/index.php

WIKIPEDIA: La enciclopedia libre por excelencia. Autodefinida como un esfuerzo colaborativo por crear una enciclopedia gratis, libre y accesible por todos. Permite revisar, escribir y solicitar artículos.

Disponible en:

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

YAHOOFINANZAS: Todo sobre la economía en Yahoo! Finanzas. Sitio donde se encuentra la cotización de la bolsa en directo, el cambio de divisas de todos los tipos, las tendencias de la bolsa y las noticias más actuales del mercado financiero. Todos los valores del IBEX 35 se han descargado de este portal.

Disponible en:

http://es.finance.yahoo.com/

Apéndice A

Manual de instalación de Joomla! 1.5.x

1. Prerrequisitos

Para poder instalar Joomla! en un servidor local desde *localhost* hay que tener instalados previamente en el servidor el siguiente software:

- PHP 4.2.x o superior. *www.php.net*

  Lenguaje de código abierto usado para la implementación de Joomla!.

- MYSQL 3.23.x o superior. *www.mysql.com*

  Sistema de gestión de Base de Datos relacional de código abierto.

- PhpMyAdmin

  Software creado en PHP/MYSQL para la administración de la base de datos sin la necesidad de escribir líneas de comando *sql*.

- Apache 1.13.19 *www.apache.org*

  Servidor web HTTP de código abierto para plataformas Unix, Microsoft Windows, Macintosh y otras.

Todo este software se puede obtener directamente descargándose los sistemas de infraestructura de internet

WAMP para Microsoft Windows,

LAMP para Linux

XAMP para cualquier Sistema Operativo (actualmente para GNU/Linux, Microsoft Windows, Solaris, MacOS X.

El sistema usado para el proyecto es WampServer 2.0 obtenido de la página *http://www.wampserver.com/download.php* y que incluye *Apache 2.2.11, MySQL 5.1.36 y PHP 5.3.0.*

2. Instalación de WAMP

Para la instalación de WAMP se descarga dicho fichero y se ejecuta. En los pasos de instalación se pide el directorio raíz donde se alojarán los archivos para la página web, *c:\wamp\www* por defecto, y el navegador que usará.

Ilustración A-1: Estructura de WAMP

Una vez finalizada la instalación, se puede hacer una pequeña prueba para comprobar que los servidores se han instalado correctamente. En el navegador web se escribe: *http:\\localhost*.

Si todo ha ido correctamente aparecerá una página como la que se ilustra a continuación.

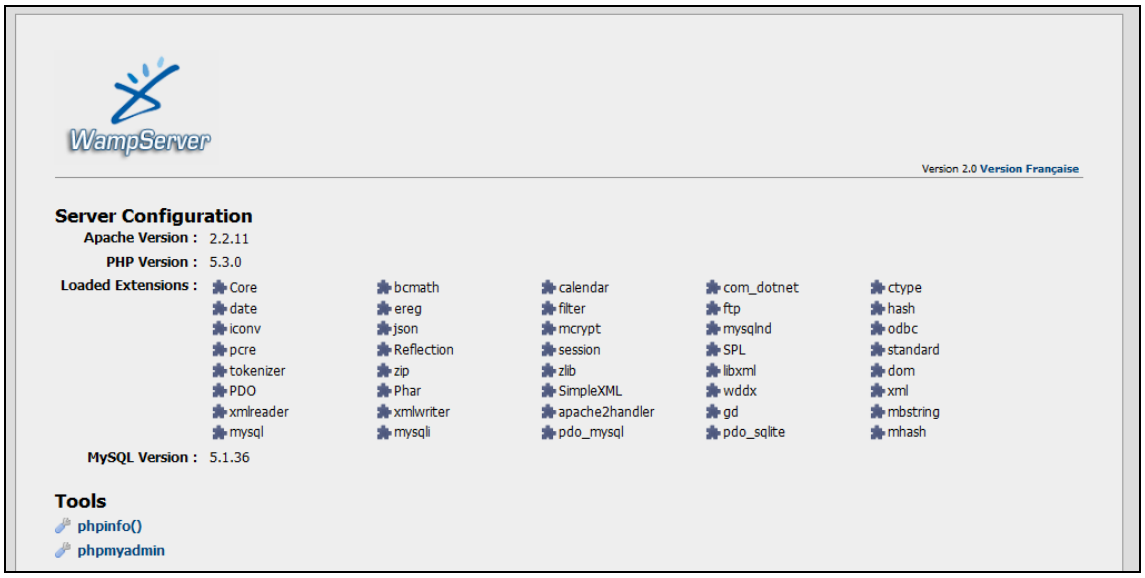


Ilustración A-2: Visualización de WAMP en el navegador

3. Instalación de Joomla

Antes de comenzar el proceso de instalación de Joomla! es necesario crear una base de datos en *MySQL*. Para ello hay que ejecutar *PhpMyadmin* a través de WAMP

Ilustración A-3: Opciones del menú WAMP

y crear una base de datos con el nombre *pfc*.

Ilustración A-4: creación de una BD en PhpMyAdmin

Además, en el directorio *root* de la instalación de WAMP (directorio www) se crea un directorio llamado *pfc* (c: \wamp\www*pfc*, por ejemplo).

A continuación, se descarga el código fuente Joomla! de la página web oficial *http://www.joomlaspanish.org/* o bien desde el fichero *Joomla_1.5.15-Spanish-pack_completo* del CD. La versión más estable, y por lo tanto, la usada en el PFC es Joomla! 1.5.15. Se descomprime el fichero Joomla! dentro del directorio *pfc* creado.

Escribiendo http:\\localhost\pfc en el navegador se abre la página de inicio de Joomla! Ahora es cuando realmente comenzará a instalarse Joomla! en el servidor local. Los pasos a seguir son:

1. Indicar el idioma. En el paquete descargado se puede elegir entre catalán, castellano e inglés.

2. Comprobar que todo el software instalado es compatible con la versión de Joomla! 1.5.15.

3. Leer la licencia de GNU/GPL.

4. Configurar la Base de Datos.

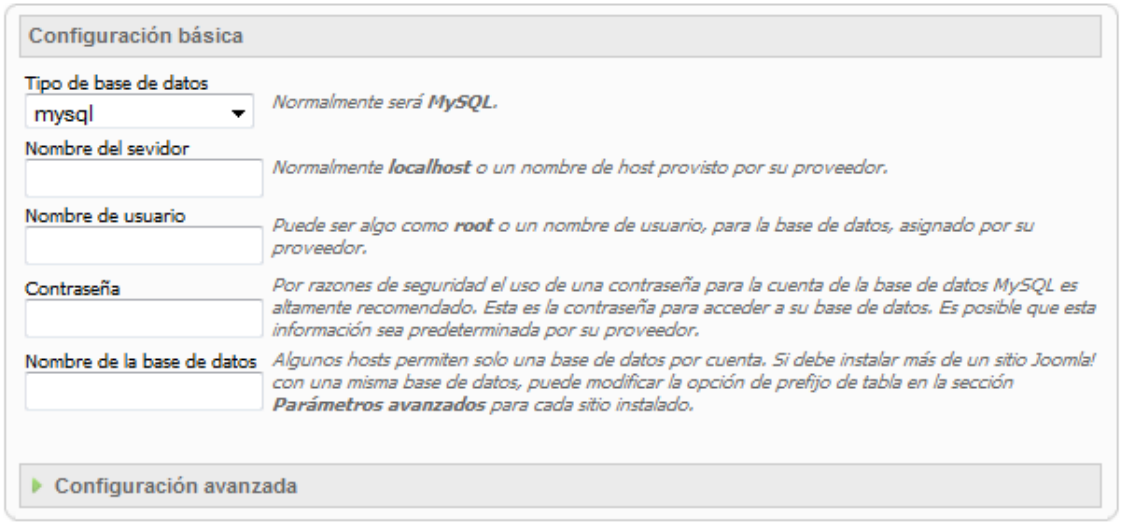


Ilustración A-5: Configuración de la BD con Joomla!

Los datos que hay que introducir son:

- Tipo de base de datos: **mysql**
- Nombre del servidor: **localhost**
- Nombre de usuario MySQL: **root**
- Contraseña: ***en blanco***
- Nombre de la base de datos MySQL: **pfc**

5. Configurar FTP. Se deja la opción "**NO**".
6. Introducir el nombre del sitio web, el correo electrónico y la contraseña. Opcionalmente, se pueden instalar los códigos de ejemplo.
7. Y finalmente, ir a la carpeta donde se encuentran los archivos Joomla! (c:\wamp\www\pfc) y eliminar la carpeta "*Installation".*

Una vez finalizada la instalación se puede visitar al sitio Joomla! mediante http://localhost/pfc/. Lógicamente, la página no tiene contenido.

Ilustración A-6: Página inicial del *frontend* de Joomla!

La zona de administración se visita con la dirección *http://localhost/pfc/administrator/*.

Ilustración A-7: Página inicial del *backend* de Joomla!

Apéndice B

Manual de instalación del Componente

1. Instalación automática

Para instalar el componente, hay que seguir una serie de pasos:

1. Escribir en el navegador *http://localhost/pfc/administrator/*.
2. Introducir el nombre de usuario, por defecto *admin,* y la contraseña elegida en el paso 6 de la instalación de Joomla!.
3. Ir al menú Extensiones->Instalar/Desinstalar

Ilustración B-1: Menú Extensiones del *backend* de Joomla!

4. Subir e instalar el archivo del CD con nombre com_finanzas.zip.

Ilustración B-2: Instalación del componente

5. Si todo ha funcionado correctamente, se mostrará un mensaje indicando que el componente se ha instalado correctamente.

Ilustración B-3: Mensaje de instalación correcta del componente

Además, en el menú componentes, aparece un nuevo submenú con el nombre de *Portafolio de finanzas*. Allí se encuentra la información de las compañías del IBEX 35.

Finalmente, introduciendo en el navegador *http://localhost/pfc/* se comprueba que ya hay contenido en el sitio web.

Ilustración B-4: Página del *frontend* de Joomla! con el componente instalado

2. Instalación manual

Para hacer una instalación manual del componente, hay que seguir los siguientes pasos:

Descomprimir el archivo *com_finanzas_manual.zip* del CD. El contenido del zip es el siguiente:

1- **Carpeta Site**: Contiene todos los archivos del componente para el *frontend*. Copiar el contenido que hay dentro (carpeta com_finanzas) y pegarlo dentro de la carpeta components de Joomla!

2- **Carpeta Admin**: Contiene todos los archivos del componente para el *backend*. Copiar el contenido que hay dentro (carpeta com_finanzas) y pegarlo dentro de la carpeta administrator->components de Joomla!.

3- **consultas.zip:** están todas las consultas SQL para introducir los datos en la base de datos con *phpMyAdmin*. Hay que ir a la base de datos que usa Joomla! y a la pestaña importar. Entonces, seleccionar examinar y allí elegir el archivo consultas.zip. Este archivo automáticamente instalará todas las tablas, insertará todas las filas, el componente y algunos conectores necesarios.

4- **Carpeta language_admin**: Contiene el archivo es-ES.com_finanzas. Copiarlo y pegarlo en administrator->language->es_ES**.** Este paso se repite para los archivos de catalán e inglés.

5- **Carpeta language_site**: Contiene el archivo es-ES.com_finanzas. Copiarlo y pegarlo en language->es_ES. El paso se repite para los archivos de catalán e inglés.

Apéndice C

Casos de uso

1. Casos de uso de un usuario

Siguiendo la jerarquía de usuarios, estos casos de uso son aplicables para cada tipo de usuarios: Para los registrados y los no registrados.

1.1. Diagrama

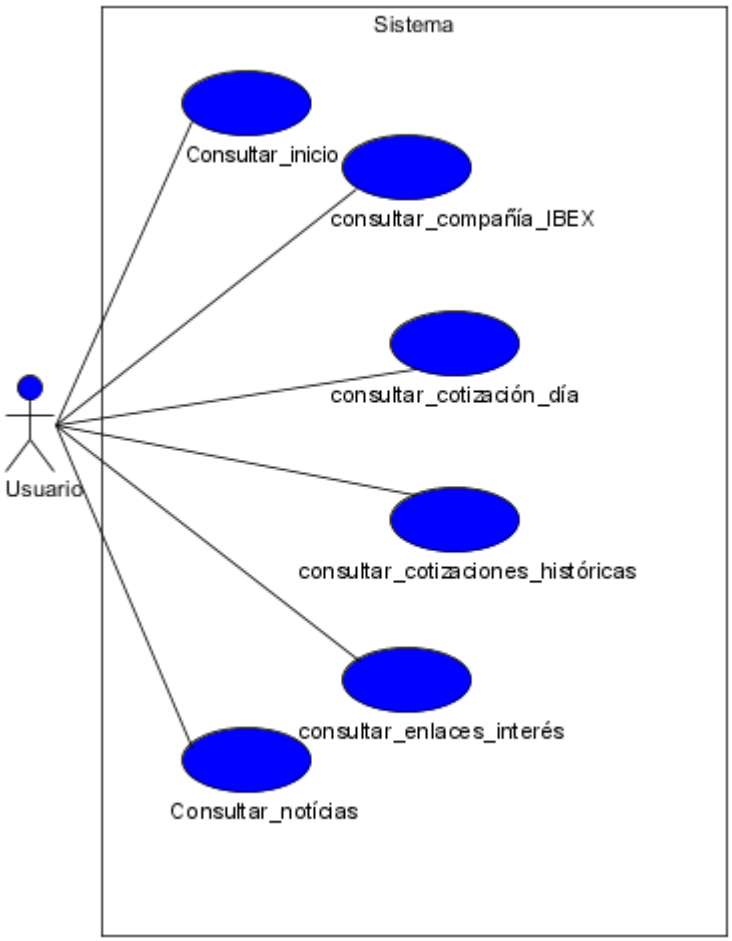


Ilustración C-1: Diagrama Casos de Uso – Usuario

1.2. Especificación

1.2.1. Consultar_inicio

Caso de uso	
Ir a inicio de página	
Descripción	
Un usuario quiere visitar la página del componente de la página web.	
Actor principal	
Usuario	
Precondición	
Ninguna	
Pos condición	
El sistema ha mostrado la página inicial del sistema.	
Curso típico de acontecimientos	
Actor	**Sistema**
1. El usuario pide que se muestre la página inicial del sistema.	2. El sistema muestra la página inicial del componente
Cursos alternativos	
No hay	

Tabla C-1: CU - Consultar_inicio

1.2.2. Consultar_compañía_IBEX

Caso de uso	
Consultar información de una compañía del IBEX 35	
Descripción	
Un usuario quiere obtener información sobre una compañía de IBEX 35.	
Actor principal	
Usuario	
Precondición	
Ninguna	
Pos condición	
El sistema muestra la información sobre la compañía del IBEX 35 seleccionada.	
Curso típico de acontecimientos	
Actor	**Sistema**
1. El usuario pide que se muestre información sobre las compañías del IBEX 35.	2. El sistema muestra un listado con todas las compañías del IBEX 35.
3. El usuario selecciona una compañía	4. El sistema muestra el perfil de la compañía seleccionada.
Cursos alternativos	
No hay	

Tabla C-2: CU - Consultar_compañía_IBEX

1.2.3. Consultar_cotización_día

Caso de uso	
Consultar cotización del día	
Descripción	
Un usuario quiere consultar la cotización del día.	
Actor principal	
Usuario	
Precondición	
Ninguna	
Pos condición	
Se muestra la información de las cotizaciones del día actual.	
Curso típico de acontecimientos	
Actor	**Sistema**
1. El usuario pide que se muestre la cotización del día.	2. El sistema muestra las cotizaciones de las 35 compañías del Ibex 35.
Cursos alternativos	
No hay	

Tabla C-3: CU - Consultar_cotización_día

1.2.4. Consultar_cotizaciones_históricas

Caso de uso	
Consultar cotizaciones históricas	
Descripción	
Un usuario quiere consultar las cotizaciones históricas del IBEX 35 o de una compañía.	
Actor principal	
Usuario	
Precondición	
Ninguna	
Pos condición	
El sistema muestra las cotizaciones del día o días seleccionados.	
Curso típico de acontecimientos	
Actor	**Sistema**
1. El usuario pide que se le muestren las cotizaciones del IBEX 35 o de una compañía.	2. El sistema pide que se seleccione el día o el intervalo de días donde mostrar las cotizaciones y la compañía del IBEX 35 que quiere consultar.
3. El usuario indica el día o el intervalo de días y la compañía o el IBEX 35.	4. El sistema muestra las cotizaciones de la compañía seleccionada o del IBEX 35 en el intervalo de fecha seleccionado. .
Cursos alternativos	
No hay	

Tabla C-4: CU - Consultar_cotizaciones_históricas

1.2.5. Consultar_enlaces_interés

Caso de uso		
Consultar enlaces de interés		
Descripción		
Un usuario quiere ver los enlaces de interés.		
Actor principal		
Usuario		
Precondición		
Ninguna		
Pos condición		
Se muestran los enlaces relacionados con finanzas y Joomla!.		
Curso típico de acontecimientos		
Actor	**Sistema**	
1. El usuario pide que se muestren los enlaces de interés.		
	2. El sistema muestra los enlaces de interés.	
3. El usuario selecciona el enlace que quiere ver.		
	4. El sistema redirige la página hacía la referencia seleccionada.	
Cursos alternativos		
No hay		

Tabla C-5: CU - Consultar_enlaces_interés

1.2.6. Consultar_noticias

Caso de uso		
Consultar noticias de finanzas		
Descripción		
Un usuario quiere ver noticias actuales sobre finanzas.		
Actor principal		
Usuario		
Precondición		
Ninguna		
Pos condición		
El sistema muestra las últimas noticias recogidas sobre finanzas		
Curso típico de acontecimientos		
Actor	**Sistema**	
1. El usuario pide que se le muestren noticias sobre finanzas. 3. El usuario selecciona el título de la noticia deseada.	2. El sistema muestra los titulares de las noticias recogidas en otras páginas. 4. El sistema muestra el contenido completo de la noticia. .	
Cursos alternativos		
No hay		

Tabla C-6: CU - Consultar_noticias

2. Casos de uso de un usuario no registrado

Un usuario no registrado tiene el permiso de consultar la información abierta del sitio web (casos de uso anteriores) y además tiene la opción de registrarse a la zona privada.

2.1. Diagrama

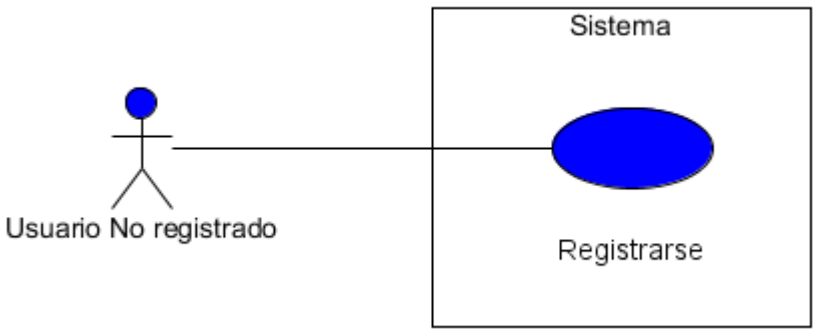


Ilustración C-2: Diagrama Casos de Uso - Usuario no registrado

2.2. Especificación

2.2.1. Registrarse

Caso de uso	
Registrarse	
Descripción	
Un usuario quiere registrarse en el sistema para usar la aplicación.	
Actor principal	
Usuario no registrado	
Precondición	
El usuario no está registrado	
Pos condición	
Se ha registrado un nuevo usuario en el sistema.	
Curso típico de acontecimientos	
Actor	**Sistema**
1. El usuario pide registrarse en el sistema.	

3. El usuario introduce sus datos. 5. El usuario confirma su solicitud de ingreso recibida en el correo.	2. El sistema muestra los campos que hay que rellenar. 4. El sistema comprueba que los datos estén correctos y manda un correo de confirmación al usuario .

Cursos alternativos
3a. La información entrada es incorrecta. 3a1. El sistema notifica al usuario que los datos son incorrectos. 3a2. El usuario entra de nuevo los datos. 3a3. El caso de uso continúa en el paso 3.

Tabla C-7: CU - Registrarse

3. Casos de uso del usuario registrado – cliente

3.1. Diagrama

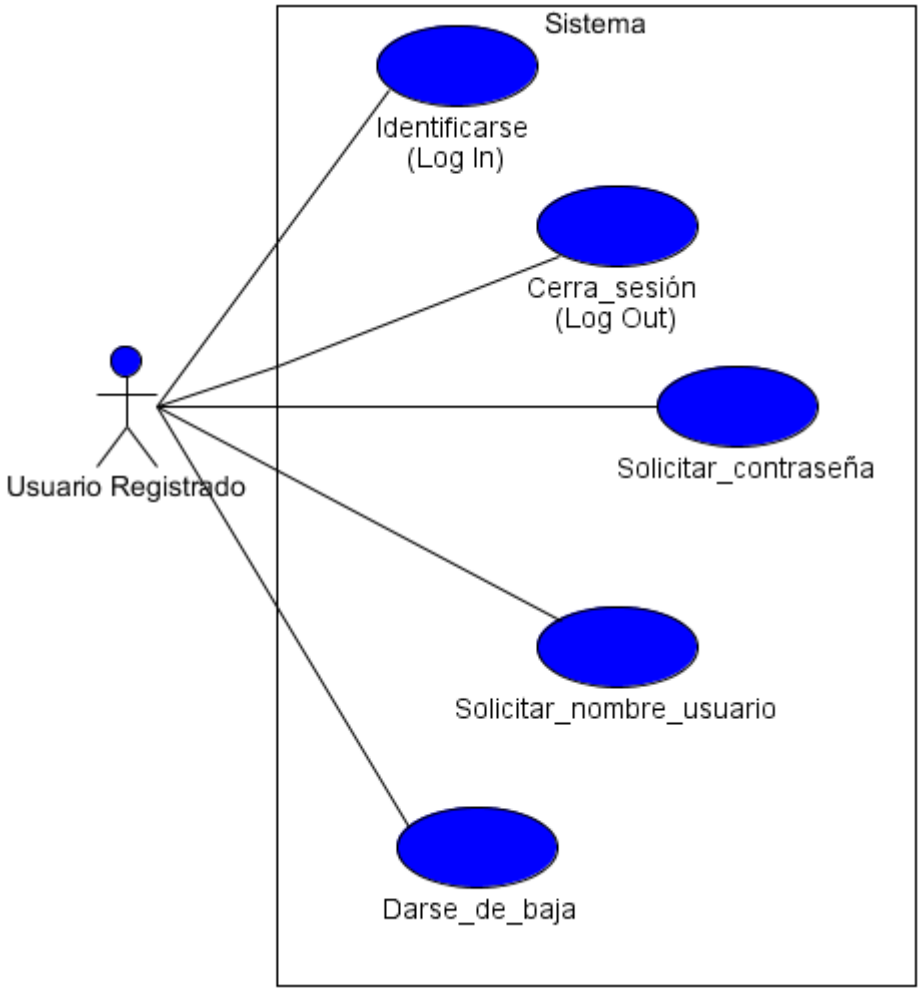


Ilustración C-3: Diagrama Casos de Uso - Usuario registrado (cliente)

3.2. Especificación

3.2.1. Identificarse (Login)

Caso de uso		
Identificarse (Login)		
Descripción		
Un usuario registrado en el sistema se identifica para acceder a su contenido.		
Actor principal		
Usuario registrado (cliente)		
Precondición		
El usuario está registrado.		
Pos condición		
Se inicia la sesión del usuario.		
Curso típico de acontecimientos		
Actor	**Sistema**	
1. El usuario registrado pide acceder a su cuenta 2. El usuario registrado introduce sus datos.	3. El sistema comprueba que los datos estén correctos. 4. El sistema permite el acceso al usuario registrado	
Cursos alternativos		
3a. La información entrada es incorrecta. 3a1. El sistema notifica al usuario que los datos son incorrectos. 3a2. El usuario entra de nuevo los datos. 3a3. El caso de uso continúa en el paso 3.		

Tabla C-8: CU - Identificarse (Login)

3.2.2. Cerrar_sesión (Logout)

Caso de uso		
Logout		
Descripción		
Un usuario registrado quiere cerrar la sesión abierta en el sistema.		
Actor principal		
Usuario registrado (cliente)		
Precondición		
El usuario está registrado y tiene una sesión abierta.		
Pos condición		
Se cierra la sesión del usuario.		
Curso típico de acontecimientos		
Actor	**Sistema**	
1. El usuario pide cerrar su sesión.	2. El sistema cierra la sesión del usuario.	
Cursos alternativos		
No hay		

Tabla C-9: CU - Cerrar_sesión (Logout)

3.2.3. Solicitar_contraseña

Caso de uso	
Solicitar contraseña	
Descripción	
Un usuario registrado quiere pedir su contraseña al sistema.	
Actor principal	
Usuario registrado (cliente)	
Precondición	
El usuario está registrado.	
Pos condición	
Se envía la nueva contraseña al usuario	
Curso típico de acontecimientos	
Actor	**Sistema**
1. El usuario pide al sistema que se la indique su contraseña introduciendo su correo electrónico.	
	2. El sistema envía un correo con una clave de verificación del correo electrónico.
3. El usuario introduce la clave recibida en el correo	
	4. El sistema verifica la clave y envía la nueva contraseña al usuario
Cursos alternativos	
2a. El correo no se encuentra en la base de datos 2a1. El sistema notifica al usuario que el correo electrónico es incorrecto. 2a2. El usuario entra de nuevo el dato. 3a3. El caso de uso continúa en el paso 2. 4a. La clave introducida por el usuario es incorrecta. 4a1. El sistema notifica al usuario que la clave es incorrecta. 4a2. El usuario entra de nuevo el dato. 4a3. El caso de uso continúa en el paso 4.	

Tabla C-10: CU - Solicitar_contraseña

3.2.4. Solicitar_nombre_ usuario

Caso de uso	
Solicitar nombre de usuario	
Descripción	
Un usuario registrado quiere pedir su nombre de usuario al sistema.	
Actor principal	
Usuario registrado (cliente)	
Precondición	
El usuario está registrado.	
Pos condición	
Se envía el nombre de usuario del usuario registrado.	
Curso típico de acontecimientos	
Actor	**Sistema**
1. El usuario pide al sistema que se la indique su nombre de usuario introduciendo su correo electrónico.	2. El sistema envía un correo con el nombre de usuario.
Cursos alternativos	
2a. El correo no se encuentra en la base de datos 2a1. El sistema notifica al usuario que el correo electrónico es incorrecto. 2a2. El usuario entra de nuevo el dato. 3a3. El caso de uso continúa en el paso 2.	

Tabla C-11: CU - Solicitar_nombre_usuario

3.2.5. Darse_de_baja

Caso de uso	
Darse de baja	
Descripción	
Un usuario registrado quiere darse de baja del sistema.	
Actor principal	
Usuario registrado (cliente)	
Precondición	
El usuario está registrado y ha iniciado sesión.	
Pos condición	
Se da de baja el usuario del sistema junto con toda la información relacionada con el usuario.	
Curso típico de acontecimientos	
Actor	**Sistema**
1. El usuario pide al sistema que se le dé de baja.	2. El sistema pide confirmación de baja al usuario y lo elimina del sistema.
Cursos alternativos	
No hay.	

Tabla C-12: CU - Darse_de_baja

3.3. Diagrama

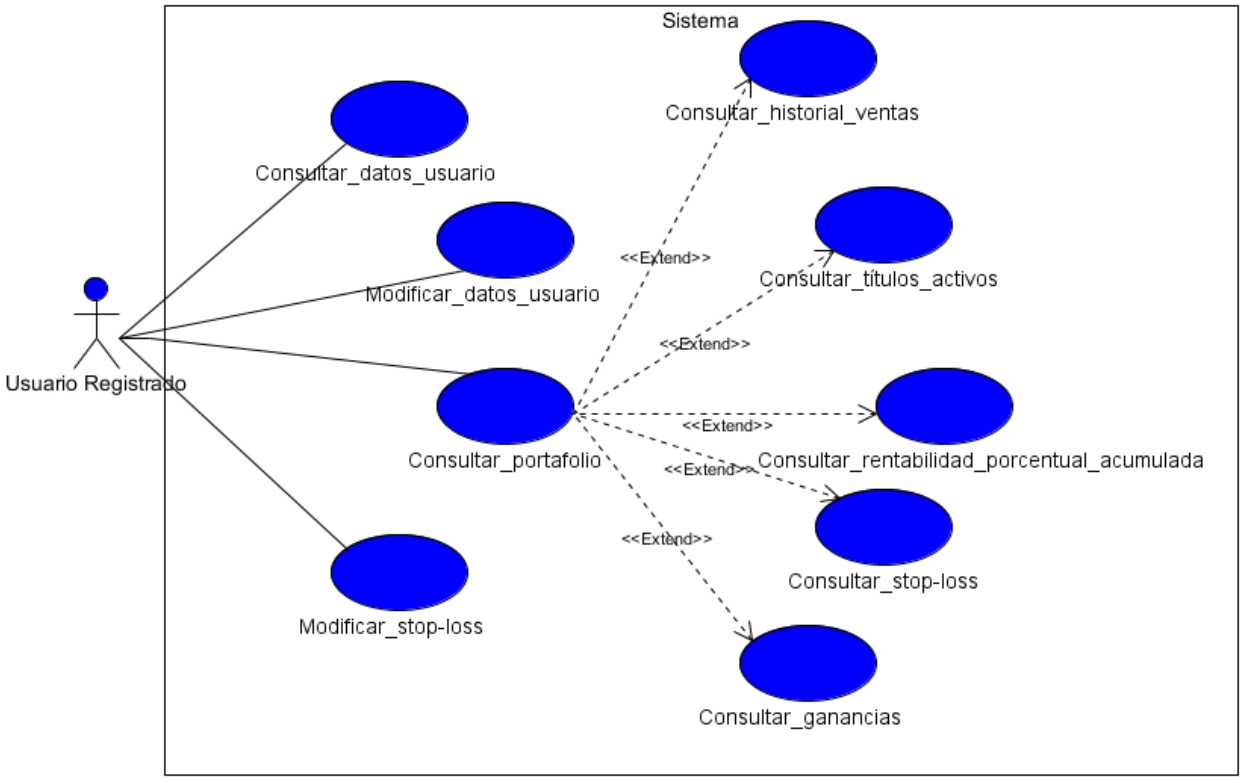


Ilustración C-4: Diagrama Casos de Uso - Usuario registrado (cliente)

3.4. Especificación

3.4.1. Consultar_datos_usuario

Caso de uso	
Consultar datos usuario	
Descripción	
Un usuario registrado quiere ver sus datos personales	
Actor principal	
Usuario registrado (cliente)	
Precondición	
El usuario está registrado y ha iniciado sesión.	
Pos condición	
Se muestran los datos del usuario registrado.	
Curso típico de acontecimientos	
Actor	**Sistema**
1. El usuario solicita ver sus datos.	2. El sistema muestra los datos del usuario.
Cursos alternativos	
No hay	

Tabla C-13: CU - Consultar_datos_usuario

3.4.2. Modificar_datos_usuario

Caso de uso	
Modificar datos usuario	
Descripción	
Un usuario registrado quiere modificar sus datos personales	
Actor principal	
Usuario registrado (cliente)	
Precondición	
El usuario está registrado y ha iniciado sesión.	
Pos condición	
Se modifican los datos del usuario registrado.	
Curso típico de acontecimientos	
Actor	**Sistema**
1. El usuario solicita modificar sus datos.	
	2. El sistema muestra los datos del usuario.
3. El usuario modifica los datos y confirma	
	4. El sistema registra los nuevos datos.
Cursos alternativos	
No hay	

Tabla C-14: CU - Modificar_datos_usuario

3.4.3. Consultar_portafolio

Caso de uso	
Consultar portafolio	
Descripción	
Un usuario registrado quiere consultar toda la información de su portafolio hasta el momento.	
Actor principal	
Usuario registrado (cliente)	
Precondición	
El usuario está registrado y ha iniciado sesión.	
Pos condición	
Se muestran todos los datos de la información financiera del usuario.	
Curso típico de acontecimientos	
Actor	**Sistema**
1. El usuario registrado pide acceder a toda su información financiera.	2. El sistema muestra todos los datos. Extiende a los casos de uso: - consultar_títulos_activos - consultar_historial_ventas - consultar_rentabilidad_porcentual_acumulada - consultar_ganancias - consultar_stop-loss
Cursos alternativos	
2a. No hay información financiera del usuario 2a1. El sistema notifica al usuario que no hay información financiera.	

Tabla C-15: CU - Consultar_portafolio

3.4.4. Consultar_títulos_comprados

Caso de uso	
Consultar títulos comprados	
Descripción	
Un usuario registrado quiere consultar sus títulos comprados (acciones activas).	
Actor principal	
Usuario registrado (cliente)	
Precondición	
El usuario está registrado y ha iniciado sesión.	
Pos condición	
Se muestran las acciones activas del usuario.	
Curso típico de acontecimientos	
Actor	**Sistema**
1. El usuario registrado pide que se le muestre la información de sus títulos comprados	2. El sistema muestra las acciones activas; son aquellas que se han comprado y aún no se han vendido en su totalidad. Los datos que muestra son: - Ticker de la compañía - Fecha de la compra - Precio de compra - Cantidad comprada
Cursos alternativos	
2a. Aún no se tienen acciones activas registradas. 2a1. El sistema notifica al usuario que no hay datos registrados	

Tabla C-16: CU - Consultar_títulos_comprados

3.4.5. Consultar_historial_ventas

Caso de uso	
Consultar historial de ventas	
Descripción	
Un usuario registrado quiere consultar la información histórica de sus acciones.	
Actor principal	
Usuario registrado (cliente)	
Precondición	
El usuario está registrado y ha iniciado sesión.	
Pos condición	
Se ha mostrado toda la información de los datos financieros históricos.	
Curso típico de acontecimientos	
Actor	**Sistema**
1. El usuario registrado pide que se le muestre la información histórica de sus acciones.	2. El sistema muestra las acciones históricas; son aquellas que se han vendido totalmente. Los datos que muestra son: - Ticker de la compañía - Fecha de la compra - Fecha de la venta - Precio de compra - Precio de la venta - Ganancia simple - Cantidad vendida
Cursos alternativos	
2a. Aún no se tiene información histórica registrada. 2a1. El sistema notifica al usuario que no hay datos registrados.	

Tabla C-17: CU - Consultar_historial_ventas

3.4.6. Consultar_rentabilidad_porcentual_acumulada

Caso de uso	
Consular rentabilidad simple acumulada	
Descripción	
Un usuario registrado quiere consultarla rentabilidad porcentual acumulada de sus acciones activas	
Actor principal	
Usuario registrado (cliente)	
Precondición	
El usuario está registrado y ha iniciado sesión.	
Pos condición	
Se muestra la rentabilidad porcentual acumulada de las acciones del usuario.	
Curso típico de acontecimientos	
Actor	**Sistema**
1. El usuario registrado pide que se le muestre la rentabilidad porcentual acumulada de sus acciones indicando la fecha a consultar.	2. El sistema muestra la rentabilidad porcentual acumulada del portafolio
Cursos alternativos	
1a. No hay información financiera en la fecha indicada. 1a1. El sistema indica que no hay valores en el mercado en la fecha indicada. 1a2. El caso de uso empieza de nuevo. 2a. No hay información de la rentabilidad porcentual acumulada del usuario 2a1. El sistema notifica al usuario que no hay información de la rentabilidad simple acumulada.	

Tabla C-18: CU - Consultar_rentabilidad_porcentual_acumulada

3.4.7. Consultar_ganancias

Caso de uso		
Consultar ganancias		
Descripción		
Un usuario registrado quiere consultar sus ganancias		
Actor principal		
Usuario registrado (cliente)		
Precondición		
El usuario está registrado y ha iniciado sesión.		
Pos condición		
Se muestran las ganancias del usuario.		
Curso típico de acontecimientos		
Actor	**Sistema**	
	1. El sistema muestra la ganancia total del usuario	
Cursos alternativos		
2a. No hay información de las ganancias del usuario 2a1. El sistema notifica al usuario que no hay información de sus ganancias.		

Tabla C-19: CU - Consultar_ganancias

3.4.8. Consultar_ stop-loss

Caso de uso		
Consular Stop-loss		
Descripción		
Un usuario registrado quiere consultar el stop-loss de cada una de sus acciones.		
Actor principal		
Usuario registrado (cliente)		
Precondición		
El usuario está registrado y ha iniciado sesión.		
Pos condición		
Se muestra el stop-loss de las acciones del usuario.		
Curso típico de acontecimientos		
Actor	**Sistema**	
1. El usuario registrado pide que se le muestre el stop-loss de sus acciones.	2. El sistema muestra el stop-loss de cada una de las acciones.	
Cursos alternativos		
2a. No hay información del stop-loss del usuario 2a1. El sistema notifica al usuario que no hay información del stop-loss.		

Tabla C-20: CU - Consultar_stop-loss

3.4.9. Modificar_ stop-loss

Caso de uso		
Modificar Stop-loss		
Descripción		
Un usuario registrado quiere consultar el stop-loss de sus acciones.		
Actor principal		
Usuario registrado (cliente)		
Precondición		
El usuario está registrado y ha iniciado sesión.		
Pos condición		
Se muestra el stop-loss de las acciones del usuario.		
Curso típico de acontecimientos		
Actor	**Sistema**	
1. El usuario registrado modifica el porcentaje para que el sistema calcule el stop-loss nuevo (por defecto 3,5%).	2. El sistema calcula y muestra el nuevo stop-loss	
Cursos alternativos		
2a. No hay información del stop-loss del usuario 2a1. El sistema notifica al usuario que no hay información del stop-loss.		

Tabla C-21: CU - Modificar_stop-loss

3.5. Diagrama

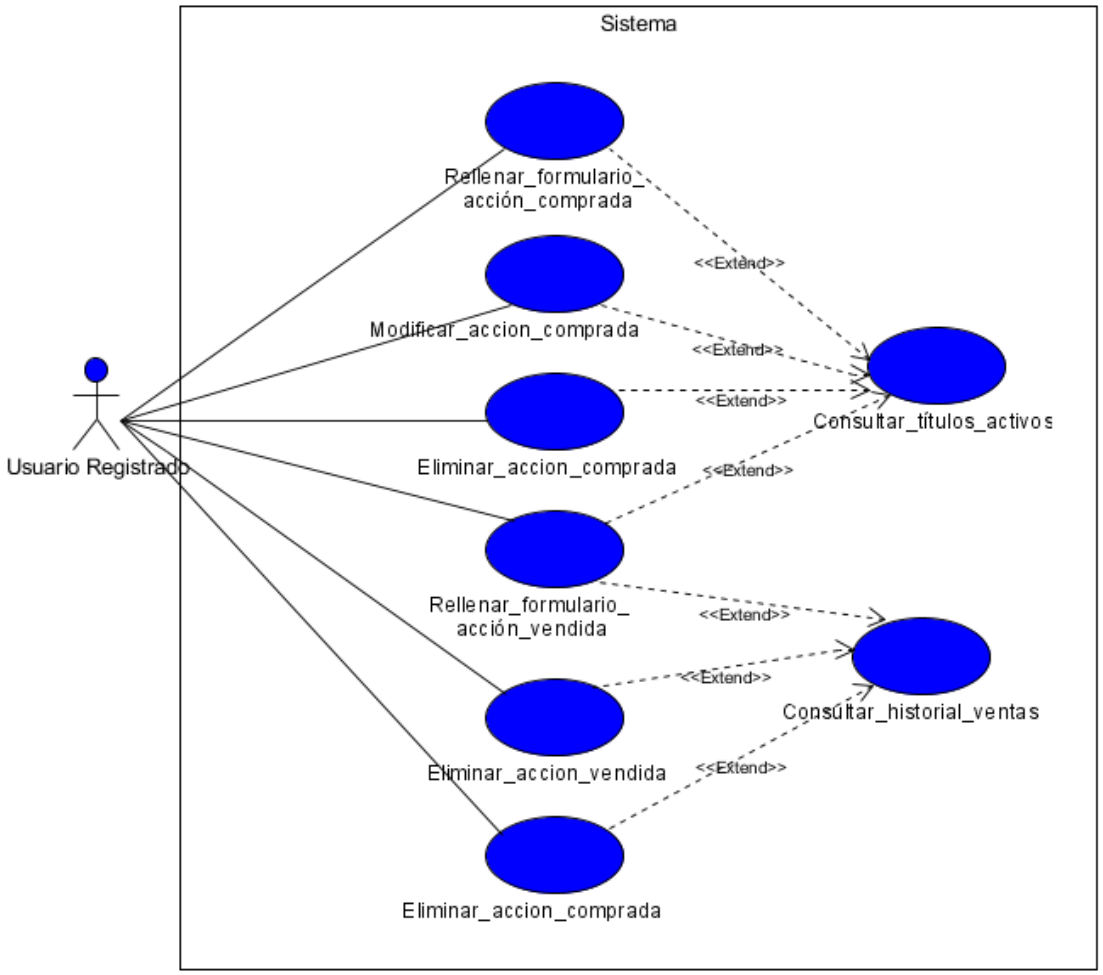


Ilustración C-5: Diagrama Casos de Uso - Usuario registrado (cliente)

3.6. Especificación

3.6.1. Rellenar_formulario_ acción_comprada

Caso de uso	
Rellenar formulario de acción comprada	
Descripción	
Un usuario registrado quiere registrar acciones compradas en el sistema.	
Actor principal	
Usuario registrado (cliente)	
Precondición	
El usuario está registrado y ha iniciado sesión.	
Pos condición	
Se han registrado las acciones de una compañía del IBEX 35 compradas por el usuario registrado.	
Curso típico de acontecimientos	
Actor	**Sistema**
1. El usuario registrado pide rellenar el formulario de las acciones compradas.	2. El sistema muestra un formulario para rellenar. En el formulario se pide: - La compañía IBEX - La fecha de compra - Precio de la compra - El número de acciones compradas.
3. El usuario rellena dichos datos.	4. El sistema comprueba que los datos estén correctos, es decir, que en la fecha introducida la compañía tenga acciones al precio indicado. Finalmente, registra la/s acciones comprada/s en la base de datos.
Cursos alternativos	
3a.El usuario pide al sistema que le muestre el intervalo de precios de las acciones de la compañía y fecha seleccionados. 3a1.El sistema muestra el máximo y mínimo de precios para ese día. 3a2.El caso de uso continúa en el paso 3. 4a. Los datos entrados son incorrectos. 4a1. El sistema notifica al usuario que los datos son incorrectos. 4a2. El usuario entra de nuevo los datos. 4a3. El caso de uso continúa en el paso 4.	

Tabla C-22: CU - Rellenar_formulario_acción_comprada

3.6.2. Modificar_acción_comprada

Caso de uso	
Modificar acción comprada	
Descripción	
Un usuario registrado quiere modificar los valores de una acción comprada registrada en el sistema como título activo.	
Actor principal	
Usuario registrado (cliente)	
Precondición	
El usuario está registrado y ha iniciado sesión.	
Pos condición	
Se han cambiado los valores del título activo con los nuevos valores	
Curso típico de acontecimientos	
Actor	**Sistema**
1. El usuario registrado pide modificar los datos de un título activo seleccionado.	
	2. El sistema muestra un formulario para rellenar. En el formulario se pide: - La fecha de compra - Precio de la compra - El número de acciones compradas.
3. El usuario rellena dichos datos.	
	4. El sistema comprueba que los datos estén correctos y registra la/s los cambios del título activo..
Cursos alternativos	
3a.El usuario pide al sistema que le muestre el intervalo de precios de las acciones de la compañía y fecha seleccionados. 3a1.El sistema muestra el máximo y mínimo de precios para ese día. 3a2.El caso de uso continúa en el paso 3. 4a. Los datos entrados son incorrectos. 4a1. El sistema notifica al usuario que los datos son incorrectos. 4a2. El usuario entra de nuevo los datos. 4a3. El caso de uso continúa en el paso 4.	

Tabla C-23: CU - Modificar_acción_comprada

3.6.3. Eliminar_acción_comprada

Caso de uso		
Eliminar acción comprada		
Descripción		
Un usuario registrado quiere eliminar un registro de una acción comprada (título activo) del sistema		
Actor principal		
Usuario registrado (cliente)		
Precondición		
El usuario está registrado y ha iniciado sesión.		
Pos condición		
Se ha eliminado el título activo seleccionado.		
Curso típico de acontecimientos		
Actor		**Sistema**
1. El usuario selecciona un título activo y pide al sistema que lo elimine.		
		2. El sistema muestra una ventana de confirmación.
3. El usuario confirma la eliminación.		
		4. El sistema elimina el título activo seleccionado.
Cursos alternativos		
3a. El usuario decide cancelar la eliminación. 4a1. El caso de uso empieza de nuevo.		

Tabla C-24: CU - Eliminar_acción_comprada

3.6.4. Rellenar_formulario_acción_vendida

Caso de uso	
Rellenar formulario de acción vendida	
Descripción	
Un usuario registrado quiere registrar acciones vendidas en el sistema.	
Actor principal	
Usuario registrado (cliente)	
Precondición	
El usuario está registrado y ha iniciado sesión.	
Pos condición	
Se han registrado las acciones vendidas de una compañía del IBEX 35 por el usuario registrado.	
Curso típico de acontecimientos	
Actor	**Sistema**
1. El usuario registrado pide rellenar el formulario de las acciones vendidas.	
	2. El sistema muestra un formulario para rellenar. En el formulario se pide: - La compañía IBEX - El título activo a vender - La fecha de la venta - Precio de la venta - El número de acciones vendidas.
3. El usuario rellena dichos datos.	
	4. El sistema comprueba que los datos estén correctos, es decir, que en la fecha introducida la compañía tenga acciones al precio indicado. Finalmente, registra la/s acciones vendida/s en la base de datos.
Cursos alternativos	
3a.El usuario pide al sistema que le muestre el intervalo de precios de las acciones de la compañía y fecha seleccionados. 3a1.El sistema muestra el máximo y mínimo de precios para ese día. 3a2.El caso de uso continúa en el paso 3. 4a. Los datos entrados son incorrectos. 4a1. El sistema notifica al usuario que los datos son incorrectos. 4a2. El usuario entra de nuevo los datos. 4a3. El caso de uso continúa en el paso 4.	

Tabla C-25: CU - Rellenar_formulario_acción_vendida

3.6.5. Modificar_acción_vendida

Caso de uso	
Modificar acción vendida	
Descripción	
Un usuario registrado quiere modificar los valores de una acción vendida registrada en el sistema como título histórico.	
Actor principal	
Usuario registrado (cliente)	
Precondición	
El usuario está registrado y ha iniciado sesión.	
Pos condición	
Se han cambiado los valores del título histórico con los nuevos valores	
Curso típico de acontecimientos	
Actor	**Sistema**
1. El usuario registrado pide modificar los datos de un título histórico.	
	2. El sistema muestra un formulario para rellenar. En el formulario se pide: - La fecha de la venta - Precio de la venta - El número de acciones vendidas.
3. El usuario rellena dichos datos.	
	4. El sistema comprueba que los datos estén correctos y registra la/s los cambios del título histórico.
Cursos alternativos	
3a.El usuario pide al sistema que le muestre el intervalo de precios de las acciones de la compañía y fecha seleccionados. 3a1.El sistema muestra el máximo y mínimo de precios para ese día. 3a2.El caso de uso continúa en el paso 3. 4a. Los datos entrados son incorrectos. 4a1. El sistema notifica al usuario que los datos son incorrectos. 4a2. El usuario entra de nuevo los datos. 4a3. El caso de uso continúa en el paso 4.	

Tabla C-26: CU - Modificar_acción_vendida

3.6.6. Eliminar_acción_vendida

Caso de uso	
Eliminar acción vendida	
Descripción	
Un usuario registrado quiere eliminar un registro de una acción vendida (título histórico) del sistema	
Actor principal	
Usuario registrado (cliente)	
Precondición	
El usuario está registrado y ha iniciado sesión.	
Pos condición	
Se ha eliminado el título histórico seleccionado.	
Curso típico de acontecimientos	
Actor	**Sistema**
1. El usuario selecciona un título histórico y pide al sistema que lo elimine.	
	2. El sistema muestra una ventana de confirmación.
3. El usuario confirma la eliminación.	
	4. El sistema elimina el título activo seleccionado.
Cursos alternativos	
3a. El usuario decide cancelar la eliminación. 3a1. El caso de uso empieza de nuevo.	

Tabla C-27: CU - Eliminar_acción_vendida

4. Casos de uso del usuario registrado - administrador

4.1. Diagrama

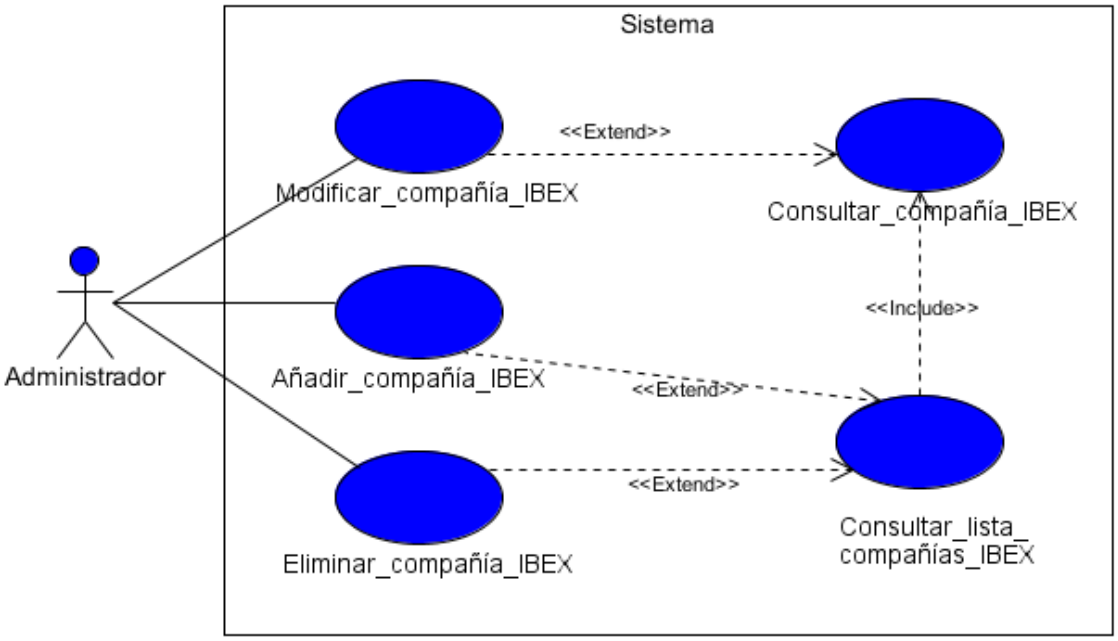


Ilustración C-6: Diagrama Casos de Uso - Usuario registrado (administrador)

4.2. Especificación

4.2.1. Añadir_compañía_IBEX

Caso de uso	
Añadir compañía del IBEX 35	
Descripción	
Un administrador quiere añadir una compañía del IBEX 35.	
Actor principal	
Usuario registrado (administrador)	
Precondición	
El administrador está registrado y ha iniciado sesión.	
Pos condición	
Se registra una compañía del IBEX 35 en el sistema.	
Curso típico de acontecimientos	
Actor	**Sistema**
1. El administrador pide añadir una nueva compañía del IBEX 35 en el sistema.	2. El sistema muestra un formulario con la siguiente información: - Ticker - ISIN - Nombre - Descripción - Domicilio - Teléfono - Fecha_inicio - URL - Capital permitido - Número de acciones - País - Uptoday - Publicado
3. El administrador introduce los datos y pide que se guarden	4. El sistema comprueba los datos, registra la compañía y vuelve al caso de uso consultar_lista_compañías_IBEX.
Cursos alternativos	
3a. El administrador cancela la inserción de una nueva compañía del IBEX 35 3a1. Se vuelve al caso de uso consultar_lista_compañías_IBEX.	

3b. El administrador pide que se apliquen los resultados. 3b1. El sistema comprueba los datos, registra la compañía. 3b2. El caso de uso continúa en el paso 3. 4a. Los datos introducidos son incorrectos 4a1. El caso de uso continúa en el paso 3.

Tabla C-28: CU - Añadir_compañía_IBEX

4.2.2. Consultar_compañía_IBEX

Caso de uso	
Consultar información de una compañía del IBEX 35	
Descripción	
Un administrador quiere obtener información sobre una compañía de IBEX 35.	
Actor principal	
Usuario registrado (administrador)	
Precondición	
El administrador está registrado y ha iniciado sesión.	
Pos condición	
El sistema muestra la información sobre la compañía del IBEX 35 seleccionada.	
Curso típico de acontecimientos	
Actor	**Sistema**
1. El administrador pide que se muestre información sobre la compañía del IBEX 35.	2. El sistema muestra el perfil de la compañía del IBEX seleccionada.
Cursos alternativos	
No hay	

Tabla C-29: CU - Consultar_compañía_IBEX

4.2.3. Modificar_compañía_IBEX

Caso de uso	
Modificar información de una compañía del IBEX 35	
Descripción	
Un administrador quiere modificar la información de una compañía del IBEX 35.	
Actor principal	
Usuario registrado (administrador)	
Precondición	
El administrador está registrado y ha iniciado sesión.	
Pos condición	
El sistema guarda los nuevos valores de la compañía del IBEX 35 seleccionada.	
Curso típico de acontecimientos	
Actor	**Sistema**
1. El administrador selecciona una compañía y pide modificar sus datos.	
	2. El sistema muestra un formulario con la siguiente información de la compañía: - Ticker - ISIN - Nombre - Descripción - Domicilio - Teléfono - Fecha_inicio - URL - Capital permitido - Número de acciones - País - Uptoday - Publicado
3. El administrador modifica la información.	
	4. El sistema comprueba los datos, modifica la compañía y vuelve al caso de uso consultar_lista_compañías_IBEX
Cursos alternativos	
3a. El administrador cancela la modificación de la información de la compañía del IBEX 35	

3a1. Se vuelve al caso de uso consultar_lista_compañías_IBEX. 3b. El administrador pide que se apliquen los resultados. 3b1. El sistema comprueba los datos, registra la compañía. 3b2. El caso de uso continúa en el paso 3. 4a. Los datos introducidos son incorrectos 4a1. El caso de uso continúa en el paso 3.

Tabla C-30: CU - Modificar_compañía_IBEX

4.2.4. Eliminar_compañía_IBEX

Caso de uso	
Eliminar una compañía del IBEX 35	
Descripción	
Un administrador quiere eliminar una compañía del IBEX 35 del sistema.	
Actor principal	
Usuario registrado (administrador)	
Precondición	
El administrador está registrado y ha iniciado sesión.	
Pos condición	
El sistema elimina la compañía del IBEX 35 seleccionada.	
Curso típico de acontecimientos	
Actor	**Sistema**
1. El administrador selecciona una compañía del IBEX 35 y pide al sistema que lo elimine.	
	2. El sistema muestra una ventana de confirmación.
3. El administrador usuario confirma la eliminación.	
	4. El sistema elimina el título activo seleccionado.
Cursos alternativos	
3a. El administrador decide cancelar la eliminación. 3a1. El caso de uso empieza de nuevo.	

Tabla C-31: CU - Eliminar_compañía_IBEX

4.2.5. Consultar_lista_compañías_IBEX

Caso de uso		
Consultar lista de compañías del IBEX 35		
Descripción		
Un administrador quiere consultar la lista de las compañías del IBEX 35 registradas en sistema.		
Actor principal		
Usuario registrado (administrador)		
Precondición		
El administrador está registrado y ha iniciado sesión.		
Pos condición		
El sistema muestra la lista de compañías del IBEX 35.		
Curso típico de acontecimientos		
Actor		**Sistema**
1. El administrador pide que se muestre una lista con las compañías del IBEX 35 registradas.		2. El sistema muestra un listado con todas las compañías del IBEX 35.
Cursos alternativos		
No hay		

Tabla C-32: CU - consultar_lista_compañías_IBEX

5. Casos de uso del usuario registrado – cliente *(Ampliación)*

A continuación se muestra un caso de uso que se podría aplicar en la ampliación del proyecto.

5.1. Diagrama

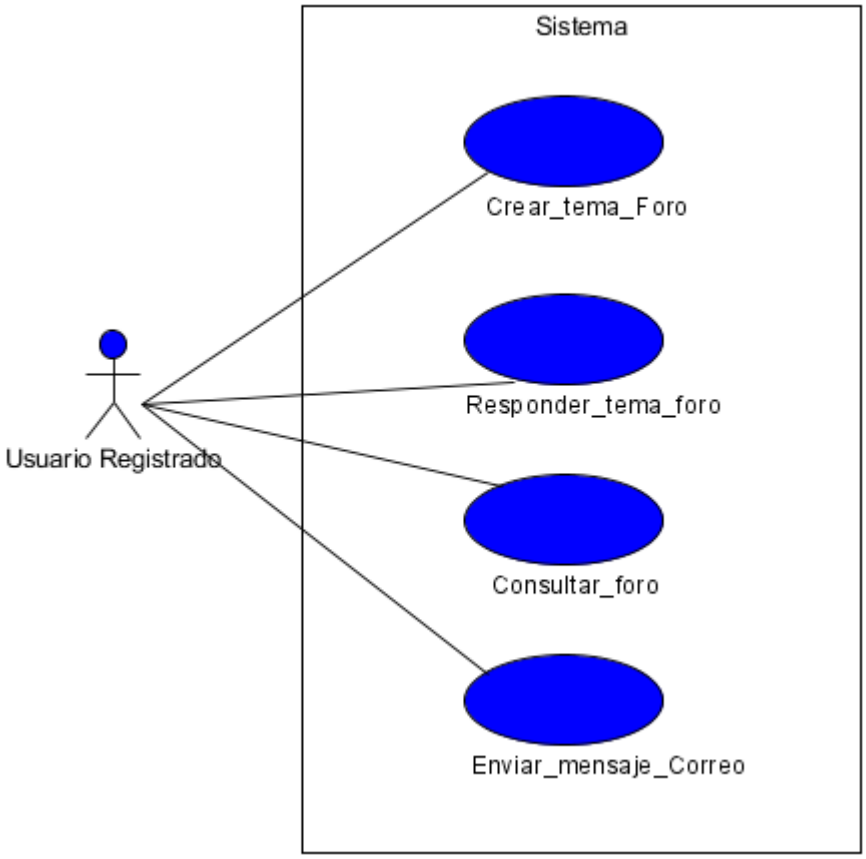


Ilustración C-7: Diagrama Casos de Uso - Usuario registrado (cliente)

5.2. Especificación

5.2.1. Crear_tema_foro

Caso de uso	
Crear tema en foro	
Descripción	
Un usuario registrado quiere crear un nuevo tema en el foro	
Actor principal	
Usuario registrado (cliente)	
Precondición	
El usuario está registrado y ha iniciado sesión.	
Pos condición	
Se crea un nuevo tema en el foro.	
Curso típico de acontecimientos	
Actor	**Sistema**
1. El usuario introduce el título y el comentario del nuevo tema del foro.	2. El sistema actualiza el foro con el nuevo tema.
Cursos alternativos	
No hay	

Tabla C-33: CU - Crear_tema_foro

5.2.2. Responder_tema_foro

Caso de uso		
Responder tema en foro		
Descripción		
Un usuario registrado quiere responder un mensaje de un tema del foro		
Actor principal		
Usuario registrado (cliente)		
Precondición		
El usuario está registrado y ha iniciado sesión.		
Pos condición		
Se ha respondido un mensaje del foro.		
Curso típico de acontecimientos		
Actor		**Sistema**
1. El usuario selecciona el tema del foro.		2. El sistema muestra todos los comentarios del tema.
3. El usuario selecciona un comentario y lo responde.		4. El sistema registra el comentario
Cursos alternativos		
No hay		

Tabla C-34: CU - Responder_tema_foro

5.2.3. Consultar_foro

Caso de uso		
Consultar foro		
Descripción		
Un usuario registrado quiere consultar el foro		
Actor principal		
Usuario registrado (cliente)		
Precondición		
El usuario está registrado y ha iniciado sesión.		
Pos condición		
Se muestra un listad con todos los temas del foro.		
Curso típico de acontecimientos		
Actor	**Sistema**	
1. El usuario pide que s ele muestre el foro.	2. El sistema muestra un listado con todos los temas del foro.	
Cursos alternativos		
No hay		

Tabla C-35: CU - Consultar_Foro

5.2.4. Enviar mensaje por correo

Caso de uso	
Enviar mensaje por correo	
Descripción	
Un usuario registrado quiere enviar un correo electrónico a otro usuario registrado	
Actor principal	
Usuario registrado (cliente)	
Precondición	
El usuario está registrado y ha iniciado sesión.	
Pos condición	
Se ha enviado el correo electrónico.	
Curso típico de acontecimientos	
Actor	**Sistema**
1. El usuario escribe el correo electrónico al usuario seleccionado.	2. El sistema envía el mensaje al correo electrónico del usuario seleccionado.
Cursos alternativos	
No hay	

Tabla C-36: CU - Enviar_mensaje_correo

Apéndice D

Estructura de la base de datos Joomla!

1. Bloques de tablas

La base de datos Joomla! contiene 34 tablas estructuradas en en siete grandes bloques.

1.1. Bloque para el contenido de un sitio web Jooma!

Joomla! proporciona una jerarquía para la organización del contenido web. Ofrece tres niveles: el más alto llamado secciones, el intermedio categorías y el situado debajo, los artículos. Una sección puede tener cero o varias categorías. Mientras que una categoría tiene cero o varios artículos. En la base de datos se indica en qué orden se muestran los artículos mediante la clase *jos_content_frontpage*. Además, para cada artículo se indica el número de visitas y la última visita de éste.

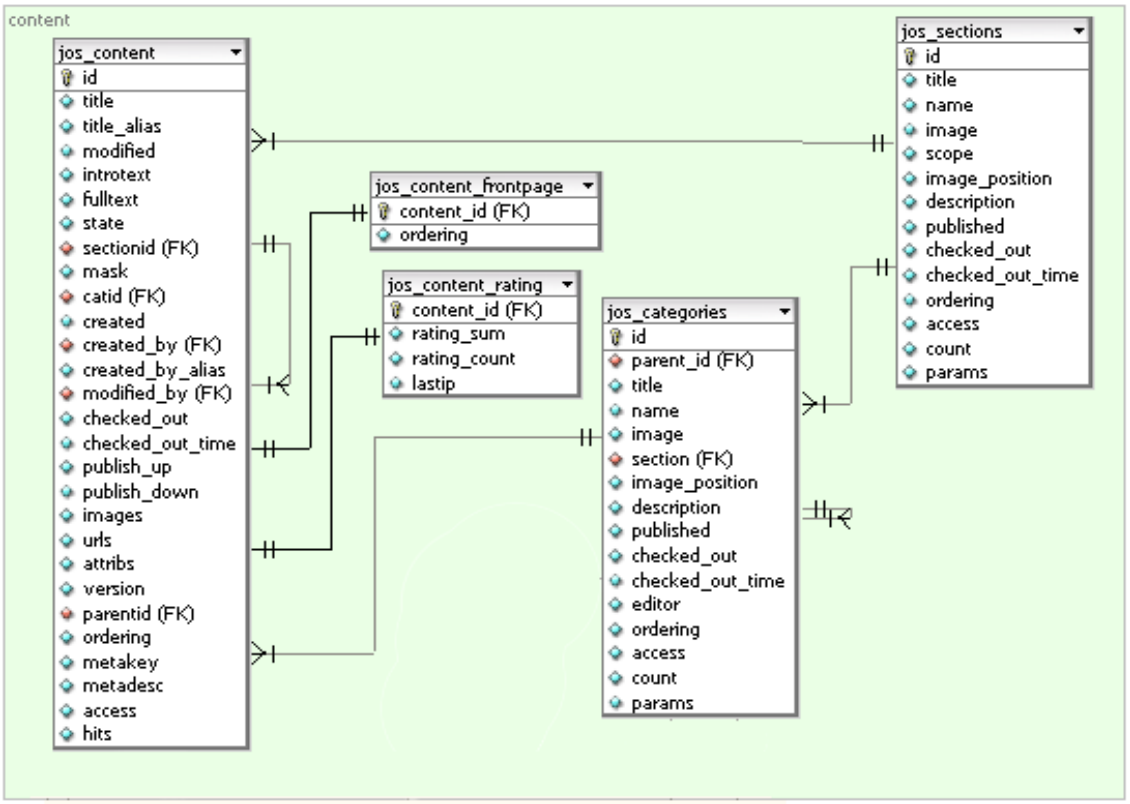


Ilustración D-1: Bloque Contenido

1.2. Bloque para las extensiones

Joomla! almacena en la base de datos las extensiones que hay instaladas en el sitio web. Mediante las clases jos_modules, jos_plugin y jos_components sabe en qué sitio de la página web se encuentran los módulos, plugins y componentes respectivamente. Hay atributos adicionales como *access* que limitan el acceso según los privilegios de usuario o *published* que indica si la extensión está publicada o no.

En mi proyecto se ha tenido que añadir una entrada en la tabla *jos_components* para indicar que se crea un nuevo componente.

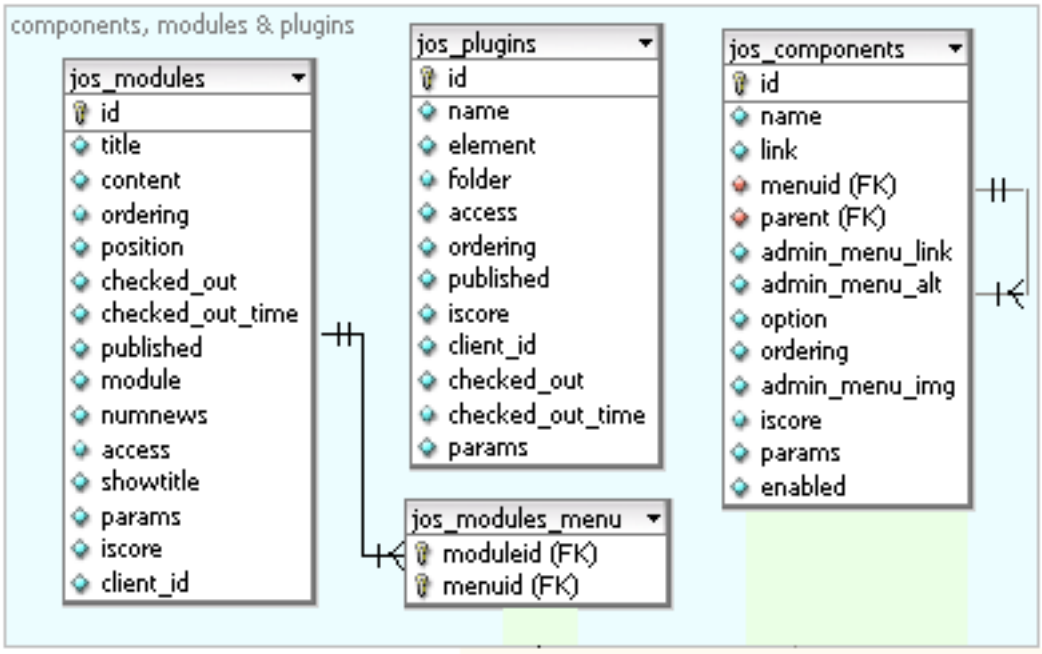


Ilustración D-2: Bloque Extensiones

1.3. Bloque para componentes instalados por defecto

Al instalar Joomla! se crean distintos componentes en la base de datos. Esto no quiere decir que se publiquen en la página web, sino que para hacer uso de ellos el atributo *published* tiene que estar a 1. Cada vez que se instala un componente, se crean en la base de datos tablas que se sitúan en este gran bloque. Éste es el ejemplo del presente proyecto: las 6 tablas se añaden aquí.

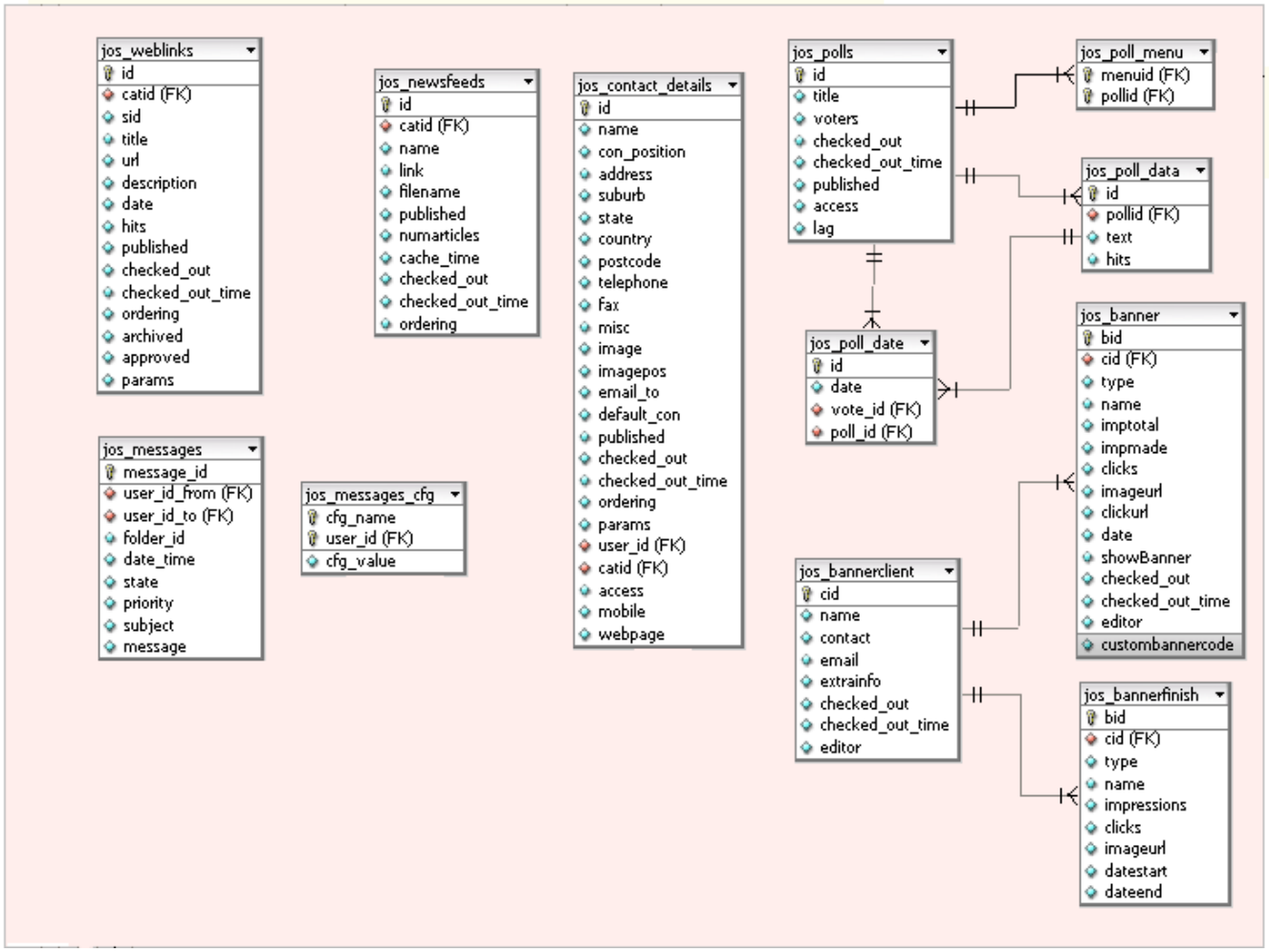


Ilustración D-3: Bloque Componentes por defecto

1.4. Bloque para las plantillas almacenadas

Estas dos tablas le permiten a Jooma! Saber dónde hay que ir a buscar en la estructura de fichero para encontrar las diferentes plantillas instaladas en el sitio web.

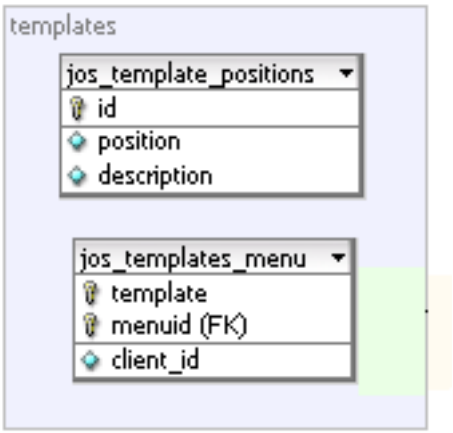
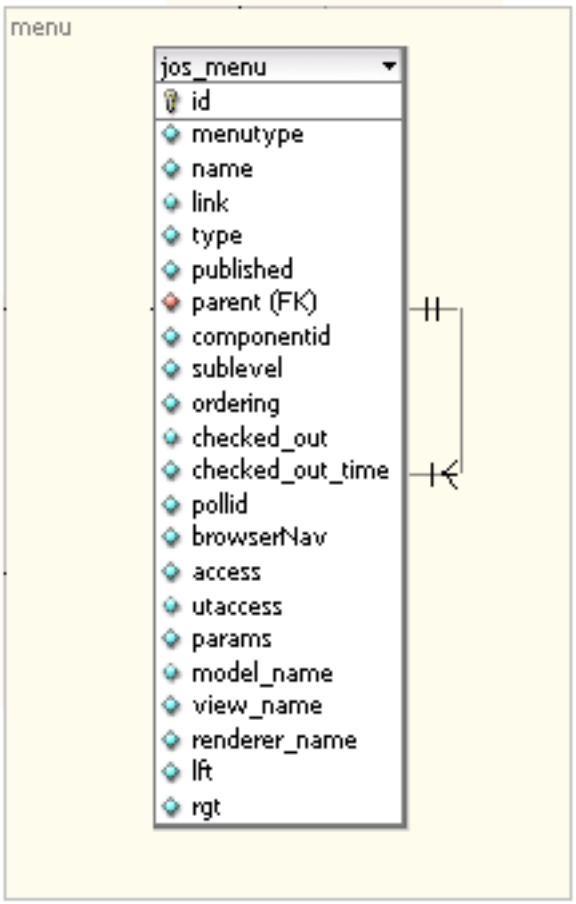


Ilustración D-4: Bloque Plantillas

1.5. Bloque para el contenido de los menús



Este bloque contiene todas las entradas que hay en los menús del sitio web y el orden en el qué se encuentran. Además indica a qué tipo de menú se añadirá. Joomla! tiene distintos tipos de menús según donde se encuentren situados en la página web. En el presente proyecto, todos los casos de uso se transforman en entradas de los menús, y por lo tanto, para cada caso de uso hay que introducir una entrada en la tabla *jos_menu* indicando el *link* donde se encuentra.

Ilustración D-5: Bloque Contenido

1.6. Bloque para los usuarios y control de acceso

Este bloque indica los usuarios que están registrados en el sistema junto con sus privilegios. En la tabla *jos_session* se almacena la última sesión del usuario en el sitio web. Las tablas más pequeñas indican a Joomla! el control de acceso de usuarios.

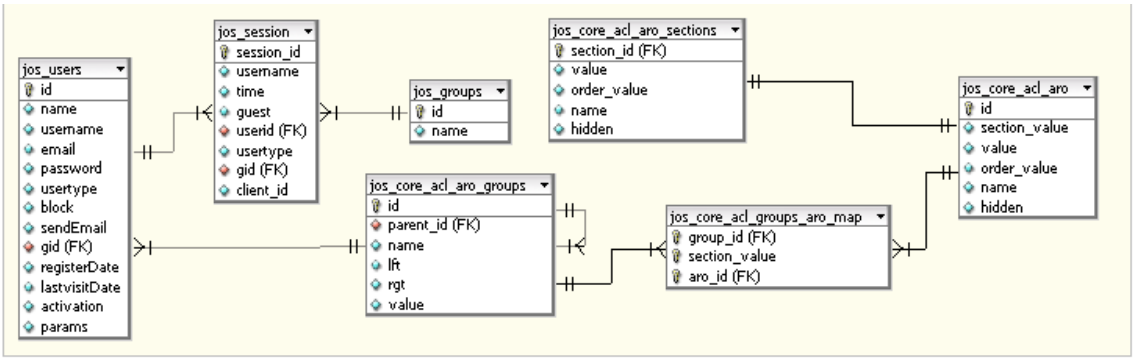


Ilustración D-6: Bloque Usuarios y Control de acceso

1.7. Bloque para los logs y estados

Este bloque simplemente sirve para controlar los errores y avisos que pueden ocurrir en el sitio web.

Ilustración D-7: Bloque Logs y Estados

2. Relación entre los bloques

En la siguiente imagen se muestra como están relacionadas todas las tablas del sitio web Joomla!. Se ilustra los enlaces que hay entre las llaves foranas de una tabla (FK) y las llaves primarias de otras (PK).

Nótese que el atributo *checked_out* se encuentra en la mayoría de las tablas y es en realidad una llave forana a user.id de la tabla *jos_users*. Para hacer más legible la imagen, se han suprimido todas las mencionadas relaciones.

Tal como se puede leer en la leyenda, existen dos tipos de relaciones. Por un lado, las *many to one* hacen referencia a las relaciones de tipo *varias-uno.* Por ejemplo, las tablas *jos_users* y *jos_messages* tienen una de estas relaciones. Esto significa que un usuario puede mandar cero o varios mensajes, mientras que un mensaje solamente puede ser escrito por un usuario. Por otro lado, las relaciones *one to one*, uno-uno, indican que una instancia de una tabla solamente se corresponde con una instancia de la otra tabla de la relación. Por ejemplo, la tabla *jos_content* que hace referencia a los artículos de un sitio web, tiene una relación de este tipo con *jos_content_rating* que indica el número de vísitas del artículo y el último usuario que lo ha consultado.

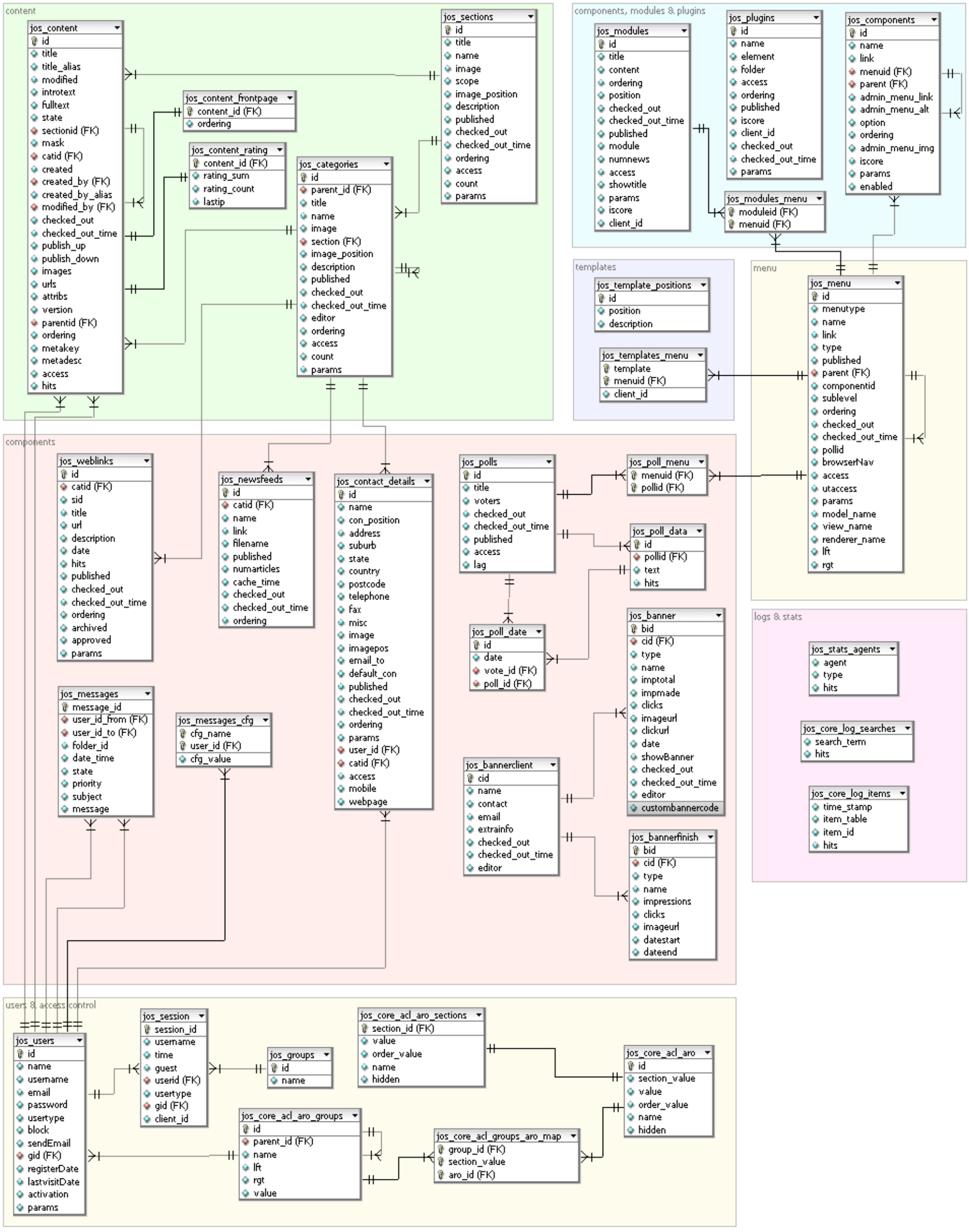


Ilustración D-8: Mapa completo de la BD Joomla!