Licencia 2: (Creative Commons)

Esta obra está bajo una licencia Reconocimiento-No comercial-Sin obras derivadas 2.5 España de Creative Commons. Puede copiarlo, distribuirlo y transmitirlo públicamente siempre que cite al autor y la obra, no se haga un uso comercial y no se hagan copias derivadas. La licencia completa se puede consultar en http://creativecommons.org/licenses/by-nc-nd/2.5/es/deed.es.

Roberto Rodríguez Cernadas
Ingeniería informática de gestión.

1 Memoria. 6

1.1 Introducción. 6
1.1.1 Motivación del proyecto. 7
1.1.2 Descripción del proyecto. 7
1.1.3 Tecnología de desarrollo. 8
1.1.4 Objetivos del proyecto. 8

1.2 Análisis de requerimientos informal 9
1.2.1 Usuarios 9
1.2.2 Servidor de aplicaciones 10
1.2.3 Actualizador de valores 10

1.3 Planificación de tareas 10
1.3.1 Diagrama de Grantt 10
1.3.2 Calendario 11

2 Especificación de requerimientos 12

2.1 Definición 12
2.1.1 Objetivos 12

2.1.2 Composición del software 12
2.1.2.1 Interfaz de usuario 12
2.1.2.2 Servidor de aplicaciones 12
2.1.2.3 Software de actualización 13

2.2 Requisitos funcionales 13
2.2.1 Interfaz de usuario 13
2.2.1.1 Usuario anónimo 13
2.2.1.2 Usuario registrado: 13
2.2.1.3 Software de actualización 14

2.3 Casos de uso 15
2.3.1 Diagrama de casos de uso. 15
2.3.1.1 Usuario no registrado. 15
2.3.1.2 Usuario registrado. 16
2.3.1.3 Software de actualización 17

2.4 Descripción textual de casos de uso 17
2.4.1 Caso de uso: Usuario registro 17
2.4.2 Caso de uso: Usuario identificación 18
2.4.3 Caso de uso: Mercados listar 18
2.4.4 Caso de uso: Mercados mostrar 19
2.4.5 Caso de uso: Entidades listar 19
2.4.6 Caso de uso: Entidades mostrar 20
2.4.7 Caso de uso: Tarjetas añadir 20
2.4.8 Caso de uso: Tarjetas eliminar 21

2.4.9 Caso de uso: Tarjetas mostrar 22
2.4.10 Caso de uso: Tarjetas listar 22
2.4.11 Caso de uso: Entidad comprar 23
2.4.12 Caso de uso: Cartera vender 24
2.4.13 Caso de uso: Cartera mostrar 24
2.4.14 Caso de uso: Cartera listar 24
2.4.15 Caso de uso: Crédito mostrar 25
2.4.16 Caso de uso: Crédito comprar 25
2.4.17 Caso de uso: Usuario modificar 26
2.4.18 Caso de uso: Usuario desconectar 27
2.4.19 Caso de uso: Mercados añadir 27
2.4.20 Caso de uso: Mercados eliminar 28
2.4.21 Caso de uso: Mercados actualizar 28
2.4.22 Caso de uso: Entidades añadir 29
2.4.23 Caso de uso: Entidades eliminar 29
2.4.24 Caso de uso: Entidades actualizar 30
2.4.25 Caso de uso: Realizar actualización 30

3 Análisis 31

3.1 Especificación de las clases de análisis. 31
3.1.1 Identificación de casos de entidades. 31
3.1.2 Especificación de atributos y clases de entidad: Diagrama 32

3.2 Diagramas de colaboración, clases frontera, clases de control.. 33
3.2.1 Diagrama de colaboración registro 33
3.2.2 Diagrama de colaboración identificación 33
3.2.3 Diagrama de colaboración listar entidades 33
3.2.4 Diagrama de colaboración compra de acciones. 34
3.2.5 Diagrama de colaboración listar cartera de acciones 34
3.2.6 Diagrama de colaboración venta de acciones 34
3.2.7 Diagrama de colaboración consulta de datos de usuario 35
3.2.8 Diagrama de colaboración modificar datos de usuario 35
3.2.9 Diagrama de colaboración actualizar entidades 35

3.3 Diseño de persistencia 36

3.4 Diseño 36
3.4.1 Diseño arquitectónico 36
3.4.2 Identificación de subsistemas 36
3.4.2.1 Subsistema de consulta 37
3.4.2.2 Subsistema de actualización: 37
3.4.2.3 Subsistema de usuarios: 37
3.4.2.4 Subsistema de compra venta 37

3.5 Diseño de la interfaz de la aplicación 37
3.5.1 Actualizador de la aplicación: 37
3.5.2 Usuario registro: 38

3.5.3 Tarjetas eliminar, Tarjetas mostrar, Tarjetas listar: 39
3.5.4 Mercados listar, Mercados mostrar, Entidades listar.. 40
3.5.5 Entidad comprar: 41
3.5.6 Cartera mostrar, Cartera listar: 42

4 Desarrollo técnico 43

4.1 Introducción a la plataforma J2EE 43

4.2 MVC 44
4.2.1 Patrón MVC 45
4.2.1.1 Estructura 45
4.2.1.2 Flujo de funcionamiento 46
4.2.2 MVC en Struts2

4.3 Struts2 46
4.3.1 Funcionamiento general de Struts2 47
4.3.1.1 Los interceptores de Struts2 48
4.3.1.2 El archivo de configuración struts.xml 48
4.3.1.3 Administración dinámica del mapping 49
4.3.1.4 Arquitectura de Struts2 49
4.3.2 MVC en Struts2 50
4.3.2.1 Controlador 50
4.3.2.2 Modelo 51
4.3.2.3 Vista 52

4.4 Hibernate 53
4.4.1 Mapeo de clases 53
4.4.2 HibernateUtil.java 54
4.4.3 El patrón de diseño singleton 55

5 Implementación 55
5.1 Instalación de nuestro proyecto. 55

5.2.1 Instalación del archivo WAR en el servidor GlassFish. 55
5.2.1 Importar proyecto 55
5.2.2 Instalación del frameworks Struts2 56
5.2.3 Cartografía Struts2 en web.xml 56
5.2.4 Instalación del framework Hibernate 57
5.2.5 Configuración hibernate.cfg.xml 58
5.2.6 Instalación del servidor Glassfish en eclipse 58
5.3 Instalación alternativa 60
5.4 Actualizador de datos (Muy importante) 61
5.5 Visualización del proyecto 61
5.6 Test del proyecto – Pruebas unitarias 62

6 Conclusiones 62
7 Bibliografía 62

1 Memoria.

1.1 Introducción.

Este proyecto, tratara de mostrar los distintos conocimientos adquiridos a lo largo de la carrera, intentando profundizar en la tecnología J2EE y en el uso de Frameworks en la implementación y el desarrollo del proyecto. Para lo cual se mostrara el funcionamiento de algunos patrones, entre ellos el MVC muy importante para el desarrollo de aplicaciones.

Para llevarlo a cabo realizaremos una aplicación web en el que los usuarios podrán simular un mercado de valores. Esta aplicación tendrá como base un elemento muy usado en las aplicaciones del mundo de Internet, un carro de compra. A partir d esta idea iremos desarrollando otras funcionalidades que completan nuestro modelo de aplicación.

1.1.1 Motivación del proyecto.

En estos últimos años han surgido multitud de juegos online, diseñados para jugar desde el navegador. En estos juegos los usuarios compiten entre si y los acontecimientos se suceden estando o no conectados los usuarios.

Sus principales ventajas son la ausencia de cualquier tipo de instalación, solo es necesario que el dispositivo que utilicemos posea un navegador web. Esto hace que podamos acceder a la aplicación desde casi cualquier plataforma.

Su temática es muy amplia y llega a todos los públicos, desde el famoso http://www.ogame.com.es en el que tienes que gobernar tu imperio intergalactico gestionando los recursos naturales de tus planetas e intentando avanzar tecnológica y militarmente mientras luchas contra otros usuarios. A otro igualmente muy conocido como el http://www.hattrick.tk/ en el que has de gestionar un equipo de fútbol, realizando fichajes y diseñando estrategias con el propósito de ganar la liga.

Este proyecto tiene como propósito realizar un juego online en el que los usuarios tengan que gestionar una cartera de valores, realizando ventas y compras de acciones del IBEX-35 compitiendo para ser el que mayores beneficios obtenga.

En este TFC, se utilizaran los conocimientos previos en Java, y realizara una investigación en aplicaciones de sistemas distribuidos, utilizando la tecnológica J2EE e implementando distintos patrones de diseño para su realización.

1.1.2 Descripción del proyecto.

Se trata de desarrollar una aplicación para la gestión de una cartera de valores, donde los usuarios podrán comprar y vender de acciones del IBEX 35 de manera ficticia y comparar la rentabilidad de sus inversiones con la de otros participantes.

El usuario dispondrá de un presupuesto inicial al hacer la cuenta, con el que podrá comprar sus primeras acciones, de este modo el usuario podrá consultar su saldo actual y el contenido de su cartera de valores. Esta le indicara la fecha de compra de el paquete de acciones, el precio que tenían en el momento de su compra, el precio actual de las acciones y el beneficio o perdida obtenido con su compra.

También se plantea la posibilidad de que el usuario pueda comprar crédito virtual utilizando dinero real, de tal modo que si el usuario ve una buena oportunidad de realizar alguna operación y no tiene liquidez suficiente pueda obtener capital. De este modo ademas obtendríamos ganancias gracias a nuestra aplicación. El cambio de dinero virtual a dinero real seria con una proporción aun por determinar a favor de el dinero virtual y este cambio de divisas solo funcionaria en un sentido: Dinero Real → Dinero Virtual.

La aplicación utilizaría los valores reales del IBEX 35 que se irían actualizando constantemente.

El proyecto pretende ser un simulador de mercado de valores lo mas fiel posible al real. De manera que los usuarios puedan probar su habilidad para poder hacer inversiones, sin asumir ningún riesgo y ademas divertirse mientras lo hacen.

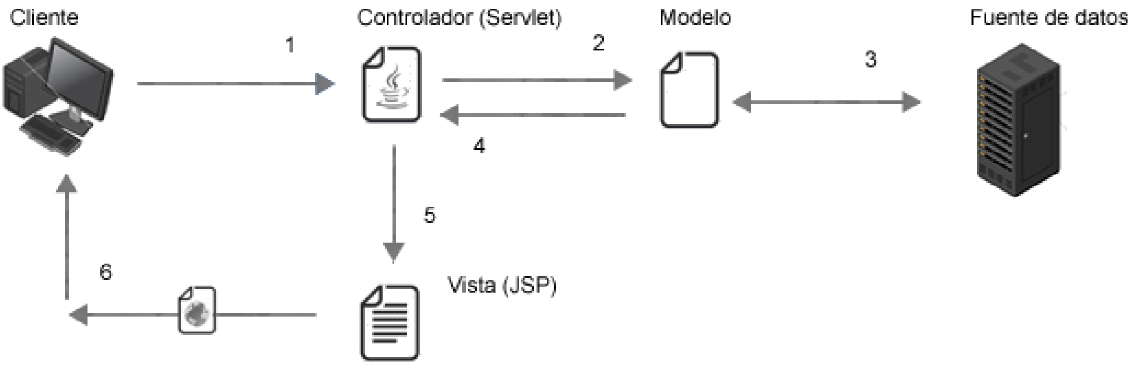
1.1.3 Tecnología de desarrollo.

El proyecto tiene mucha similitud con un comercio electrónico, con lo que gran parte de este seguiría ese modelo.

El desarrollo de esta aplicación sera realizado en J2EE siguiendo el patrón de arquitectura MVC gracias al Framework Struts 2, que nos permitirá mantener por separado el diseño, los datos, y la lógica de control. Este patrón se usa muy comúnmente en las aplicaciones web, donde la vista es la página HTML y el código que provee de datos dinámicos a la página. El modelo es el Sistema de Gestión de Base de Datos y la Lógica de negocio, y el controlador es el responsable de recibir los eventos de entrada desde la vista.

Para la capa de persistencia utilizaremos Hibernate que es una herramienta de Mapeo objeto-relacional (ORM) que permite comunicar con la base de datos de manera orientada a objetos, representando cada entidad de la base de datos como un objeto al que puede modificarse sus atributos y así modificar la base de datos real.

Modelo MVC



1.1.4 Objetivos del proyecto.

El programa dispone de dos partes bien diferenciadas, por una parte tenemos la interfaz de usuario. Esta es accesible desde el navegador, dependiendo del tipo de usuario que tengamos podremos hacer distintas operaciones.

Usuario sin registrar:

Este usuario podrá obtener información acerca del juego y las condiciones de uso, ademas también podrá consultar la cotización actual de valores del IBEX 35.

- Registrar nuevo usuario
- Identificarse como usuario
- Consultar información general del juego (reglas, condiciones de uso, etc)
- Listar valores del IBEX 35 actuales

Usuario registrado:

Este usuario, dispone de todas las opciones para poder operar con la aplicación de manera completa, y realizando operaciones de compra y venta de acciones, consultar todo tipo de información estadística acerca de sus operaciones etc.

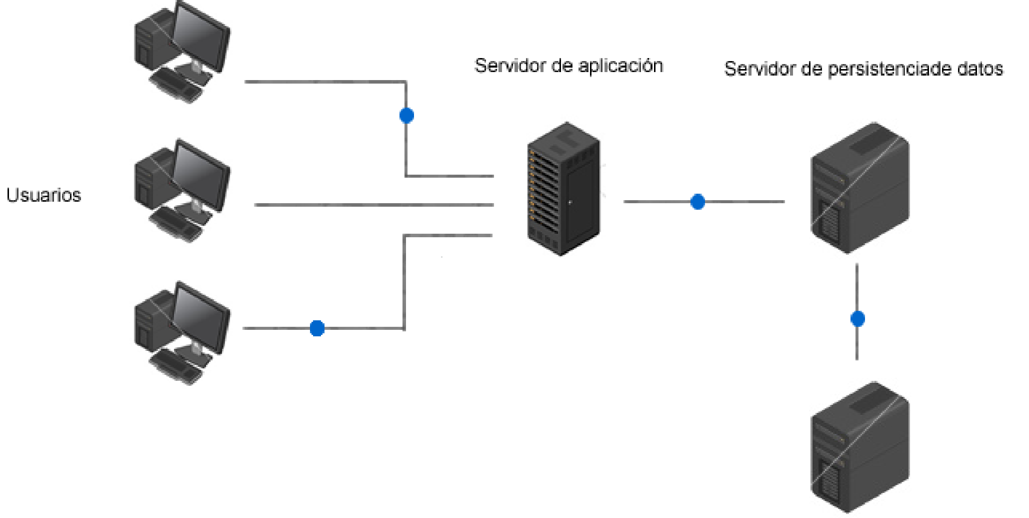
- Modificar datos de usuario
- Consultar datos de usuario
- Eliminar usuario propio
- Añadir tarjeta de crédito
- Modificar tarjeta de crédito
- Eliminar tarjeta de crédito
- Consultar información general del juego (reglas, condiciones de uso, etc)
- Consultar saldo
- Comprar saldo
- Comprar acciones
- Vender acciones
- Consultar cartera propia de acciones
- Consultar registro de operaciones realizadas
- Listar valores del IBEX 35 actuales
- Consultar registro histórico del IBEX 35
- Listar mejores jugadores

Por otra parte la aplicación dispone de un servidor de se encarga de actualizar automáticamente los datos del IBEX 35 y de compactar las estadísticas y operaciones una vez acabada la jornada.

Debido a que no disponemos de un servidor que nos informe de los valores del IBEX 35 en tiempo real, para nuestra practica obtendremos los valores de alguna web en donde estén disponibles.

El sistema automático de la aplicación se encargara de:

- Actualizar lista de IBEX 35
- Almacenar un registro de valores del IBEX 35
- Compactar las operaciones antiguas de los usuarios.



1.2 Análisis de requerimientos informal

1.2.1 Usuarios

El usuario tendrá acceso a la aplicación mediante un navegador web que soporte los estándares HTML y una conexión a Internet para conectar con el servidor de la aplicación.

1.2.2 Servidor de aplicaciones

La aplicación correrá en un servidor de aplicaciones para este caso utilizaremos el Jboss ya que ademas de ser de los mas utilizados es muy recomendado en la comunidad de Internet, aunque disponemos de otras posibilidades como GlassFish, PowerBuilder, EAServer etc.

La aplicación utiliza el modelo MVC mediante un Framework. Nosotros vamos a utilizar Struts 2 ya que es uno de los mas recomendados, ademas este Framework tiene una gran cantidad de usuarios y nos resultara más fácil obtener ejemplos e información para su manejo.

Para el modelo de persistencia utilizaremos otro Framework de uso muy común hoy en día, que viene siendo Hibernate. Este nos permitirá guardar atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos,
Este Framework también ofrece la posibilidad de ser usado como Framework MVC, pero no nos ofrece tanta flexibilidad como Struts 2. Así que ambos se complementaran para darnos mayor potencia.

La base de datos, almacenara todas las operaciones, de compra venta de nuestros usuarios, ademas de sus datos, valores de las acciones y unas cuantas cosas mas. Para almacenar estos datos utilizaremos MySQL que es un gestor de base de datos muy usado para aplicaciones de Internet, ademas proporciona un fácil manejo y una seguridad muy aceptable.

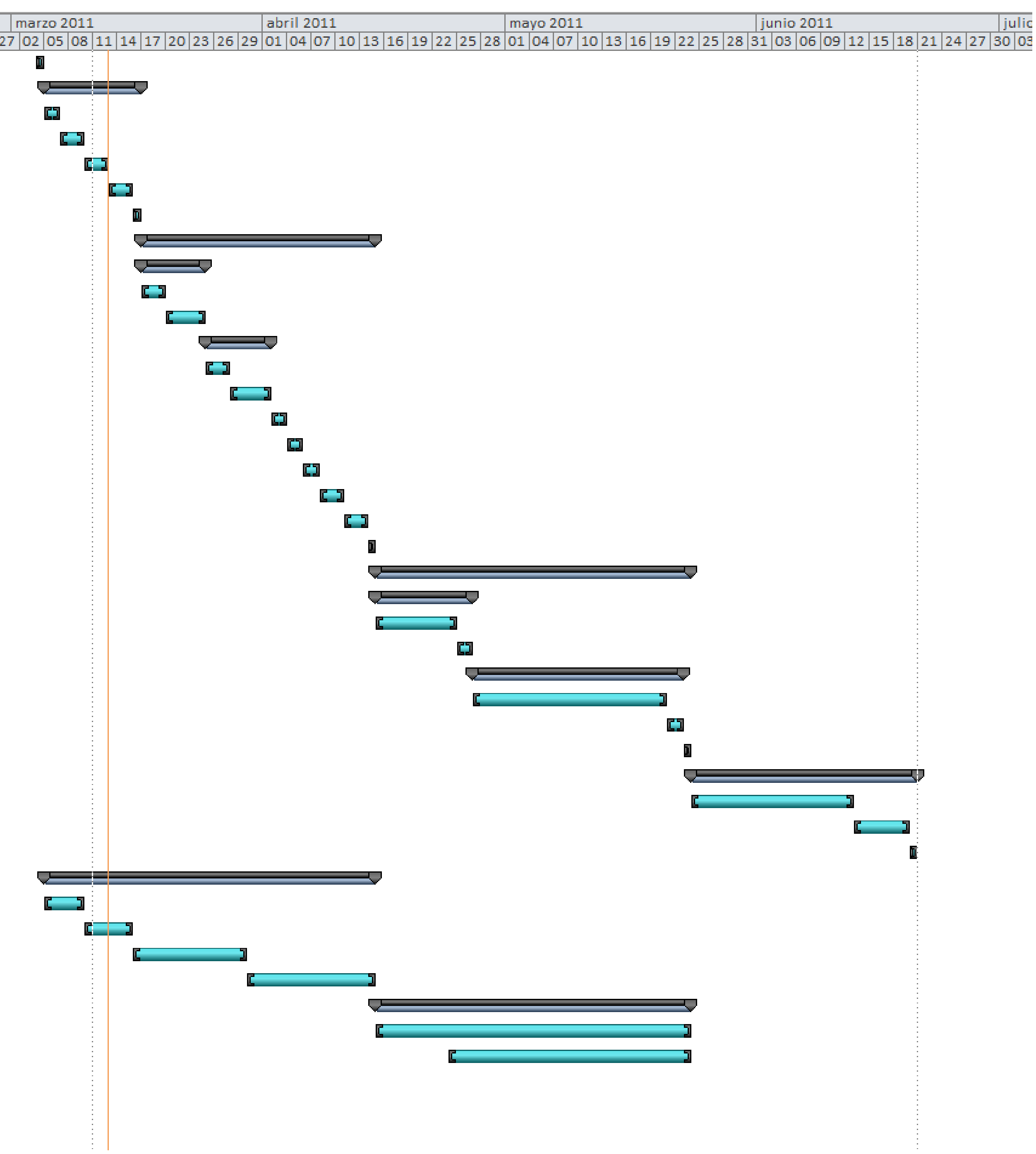
1.2.3 Actualizador de valores

Esta otra parte de la aplicación. Esta se encargara de actualizar los valores del IBEX 35 para que los usuarios puedan visualizar los últimos valores disponibles. Ya que no disponemos de ningún servidor dedicado, estos valores los iremos tomando de alguna Web mediante un socket y después de ser tratados, se almacenaran en la base de datos.

Ademas también estará encargado de compactar las estadísticas de valores y usuarios para su posterior consulta.

1.3 Planificación de tareas

1.3.1 Diagrama de Grantt



1.3.2 Calendario

Modo de tarea	Nombre de tarea	Duración	Comienzo	Fin
	Inicio del proyecto	1 día	vie 04/03/11	vie 04/03/11
	Definición y planificacion del proyecto	**12 días**	**sáb 05/03/11**	**mié 16/03/11**
	Descripción general del proyecto	2 días	sáb 05/03/11	dom 06/03/11
	Establecimiento de objetivos	3 días	lun 07/03/11	mié 09/03/11
	Definición de tareas principales	3 días	jue 10/03/11	sáb 12/03/11
	Planificacion de tareas	3 días	dom 13/03/11	mar 15/03/11
	Entrega del plan de trabajo	1 día	mié 16/03/11	mié 16/03/11
	Analisis y diseño	**29 días**	**jue 17/03/11**	**jue 14/04/11**
	Definición de subsistemas	**8 días**	**jue 17/03/11**	**jue 24/03/11**
	Gestor de actualización	3 días	jue 17/03/11	sáb 19/03/11
	Aplicación web	5 días	dom 20/03/11	jue 24/03/11
	Diagrama de clases	**8 días**	**vie 25/03/11**	**vie 01/04/11**
	Gestor de actualización	3 días	vie 25/03/11	dom 27/03/11
	Aplicación web	5 días	lun 28/03/11	vie 01/04/11
	Diagrama de casos de uso	2 días	sáb 02/04/11	dom 03/04/11
	Diagrama de colaboración	2 días	lun 04/04/11	mar 05/04/11
	Diagrama de secuencia	2 días	mié 06/04/11	jue 07/04/11
	Diseño de base de datos	3 días	vie 08/04/11	dom 10/04/11
	Diseño de Interfaces de la aplicación	3 días	lun 11/04/11	mié 13/04/11
	Entrega del plan de trabajo	1 día	jue 14/04/11	jue 14/04/11
	Implementación	**39 días**	**vie 15/04/11**	**lun 23/05/11**
	Gestor de actualización	**12 días**	**vie 15/04/11**	**mar 26/04/11**
	Implementación	10 días	vie 15/04/11	dom 24/04/11
	Prueba de funcionamiento	2 días	lun 25/04/11	mar 26/04/11
	Aplicación web	**26 días**	**mié 27/04/11**	**dom 22/05/11**
	Implementación	24 días	mié 27/04/11	vie 20/05/11
	Prueba de funcionamiento	2 días	sáb 21/05/11	dom 22/05/11
	Entrega del plan de trabajo	1 día	lun 23/05/11	lun 23/05/11
	Memoria final + Practica	**28 días**	**mar 24/05/11**	**lun 20/06/11**
	Realización de memoria final	20 días	mar 24/05/11	dom 12/06/11
	Puesta a punto de la practica y los juegos de pruebas	7 días	lun 13/06/11	dom 19/06/11
	Entrega de la memoria final	1 día	lun 20/06/11	lun 20/06/11
	Puesta a punto del entorno de trabajo	**41 días**	**sáb 05/03/11**	**jue 14/04/11**
	Investigación acerca de las tecnologias existentes	5 días	sáb 05/03/11	mié 09/03/11
	Selección de tecnologias que formaran parte	6 días	jue 10/03/11	mar 15/03/11
	Instalación del entorno de trabajo	14 días	mié 16/03/11	mar 29/03/11
	Realización de pruebas de funcionamiento	16 días	mié 30/03/11	jue 14/04/11
	Documentación de las tecnologias a utilizar	**39 días**	**vie 15/04/11**	**lun 23/05/11**
	Estudio de las tecnologias seleccionadas	39 días	vie 15/04/11	lun 23/05/11
	Practicas con las tecnologias seleccionadas	30 días	dom 24/04/11	lun 23/05/11

2 Especificación de requerimientos

2.1 Definición

2.1.1 Objetivos

La software UbrOkerC tiene como objetivo realizar un simulador de mercado y cartera de valores donde el precio de las acciones de las empresas del IBEX 35 son actualizados en "tiempo real" permitiendo a los usuarios registrados simular la compra de estas con el objetivo de comprobar su pericia en este tipo de inversiones.

Para tener pleno acceso a la aplicación el usuario deberá registrarse en esta. Tanto el registro de usuario como el manejo de la aplicación se realizara mediante un navegador web que soporte los estándares HTML y una conexión a Internet para conectar con el servidor de la aplicación.

2.1.2 Composición del software

El software de la aplicaciones esta dividido en tres partes que interactuan entre si.

2.1.2.1 Interfaz de usuario

Incluye las aplicaciones necesarias para que un cliente pueda conectarse a la aplicación e interactuar con ella. En este caso, este software es un navegador web que soporte los estándares HTML y una conexión a Internet.

2.1.2.2 Servidor de aplicaciones

La aplicación correrá en un servidor de aplicaciones para este caso utilizaremos el Jboss ya que ademas de ser de los mas utilizados es muy recomendado en la comunidad de Internet, aunque disponemos de otras posibilidades como GlassFish, PowerBuilder, EAServer etc.

La aplicación utiliza el modelo MVC mediante un Framework. Nosotros vamos a utilizar Struts 2 ya que es uno de los mas recomendados, ademas este Framework tiene una gran cantidad de usuarios y nos resultara más fácil obtener ejemplos e información para su manejo.

Para el modelo de persistencia utilizaremos otro Framework de uso muy común hoy en día, que viene siendo Hibernate. Este nos permitirá guardar atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos,
Este Framework también ofrece la posibilidad de ser usado como Framework MVC, pero no nos ofrece tanta flexibilidad como Struts 2. Así que ambos se complementaran para darnos mayor potencia.

La base de datos, almacenara todas las operaciones, de compra venta de nuestros usuarios, ademas de sus datos, valores de las acciones y unas cuantas cosas mas. Para almacenar estos datos utilizaremos MySQL que es un gestor de base de datos muy usado para aplicaciones de Internet, ademas proporciona un fácil manejo y una seguridad muy aceptable.

2.1.2.3 Software de actualización

Este software puede estar instalado junto al servidor de aplicaciones o en otro equipo que tenga acceso a la base de datos. Este software se encargara de actualizar los valores del IBEX 35 en la base de datos para que el servidor de aplicaciones disponga de los valores mas recientes y crear un registro de los valores de cada entidad de manera que podamos consultar la evolución de cada entidad en un determinado espacio de tiempo.

Como en este caso no disponemos de ningún servidor dedicado estos valores los iremos tomando de alguna Web que disponga de dicha información.

2.2 Requisitos funcionales

2.2.1 Interfaz de usuario

El programa dispone de dos partes bien diferenciadas, por una parte tenemos la interfaz de usuario. Esta es accesible desde el navegador, dependiendo del tipo de usuario que tengamos podremos hacer distintas operaciones.

2.2.1.1 Usuario anónimo

Este usuario podrá obtener información acerca del juego y las condiciones de uso, ademas también podrá consultar la cotización actual de valores del IBEX 35.

- **Usuario registro:** Permite a un usuario anónimo obtener una cuenta de usuario registrado con la que tener total acceso a la aplicación.
- **Usuario identificación:** Permite a un usuario anónimo identificarse en el sistema y tener acceso con una cuenta de usuario registrado.
- **Mercados listar:** Permite listar los mercados disponibles en la aplicación. La aplicación solo tiene como mercado de valores el IBEX 35, pero dejamos así la opción de ampliar el numero de mercados en el futuro sin realizar apenas cambios.
- **Mercados mostrar:** Permite obtener información del mercado seleccionado.
- **Entidades listar:** Permite listar las entidades pertenecientes al mercado seleccionado. Como se indico antes antes, la aplicación solo tiene como mercado de valores el IBEX 35, pero dejamos así la opción de ampliar el numero de mercados en el futuro sin realizar apenas cambios.
- **Entidades mostrar:** Permite o obtener información de la entidad seleccionada.

2.2.1.2 Usuario registrado

Un usuario registrado dispone de todas las opciones para poder operar con la aplicación de manera completa, y realizando operaciones de compra y venta de acciones, consultar todo tipo de información estadística acerca de sus operaciones etc.

- **Tarjetas añadir:** Un usuario registrado puede añadir una tarjeta de crédito a su lista de tarjetas disponibles para comprar crédito.
- **Tarjetas eliminar:** Un usuario registrado puede eliminar una tarjeta de crédito de su lista de tarjetas disponibles para comprar crédito.

- **Tarjetas mostrar:** Muestra a un usuario registrado los datos de una de sus tarjetas de crédito.
- **Tarjetas listar:** Muestra a un usuario el listado de las tarjetas de crédito que tiene asociadas a su cuenta.
- **Entidad comprar:** Añade acciones de una determinada entidad a la cartera de acciones de un usuario a cambio de el valor en crédito de las acciones
- **Cartera vender:** Añade el valor actual de las acciones vendidas al crédito del usuario.
- **Cartera mostrar:** Muestra los datos de la compra de acciones seleccionada de nuestra cartera de valores.
- **Cartera listar:** Muestra un listado de las acciones que un usuario posee en su cartera de valores.
- **Crédito mostrar:** Muestra al usuario cuanto dinero virtual posee.
- **Crédito comprar:** El usuario añade a su crédito una cantidad de dinero virtual a cambio de dinero real.
- **Usuario modificar:** El usuario modifica sus datos personales.
- **Usuario desconectar:** El usuario registrado desconecta la sesión y pasa a ser un usuario anónimo hasta que se vuelva a identificar.

2.2.1.3 Software de actualización

El software de actualización mantiene los valores de las distintas entidades actualizados y realiza un registro histórico de estas.

- **Mercados añadir:** Se añade un mercado de valores a la aplicación.
- **Mercados eliminar:** Se elimina un mercado de valores de la aplicación.
- **Mercados listar:** Permite listar los mercados disponibles en la aplicación. La aplicación solo tiene como mercado de valores el IBEX 35, pero dejamos así la opción de ampliar el numero de mercados en el futuro sin realizar apenas cambios.
- **Mercados mostrar:** Permite obtener información del mercado seleccionado.
- **Mercados actualizar**: Actualiza los datos del mercado seleccionado.
- **Entidades añadir:** Se añade una entidad al mercado seleccionado y realiza un registro histórico del valor.
- **Entidades eliminar:** Elimina una entidad del mercado seleccionado.
- **Entidades listar:** Permite listar las entidades pertenecientes al mercado seleccionado. Como se indico antes antes, la aplicación solo tiene como mercado de valores el IBEX 35, pero dejamos así la opción de ampliar el numero de mercados en el futuro sin realizar apenas cambios.
- **Entidades mostrar:** Permite obtener información de la entidad seleccionada.
- **Entidades actualizar:** Actualiza los datos de la entidad seleccionada y realiza un registro histórico del valor.
- **Realizar actualización:** Actualiza los datos de los datos de mercados y entidades del sistema.

2.3 Casos de uso

Dependiendo del tipo de usuario que acceda al sistema se disponen de un determinado grupo de acciones.

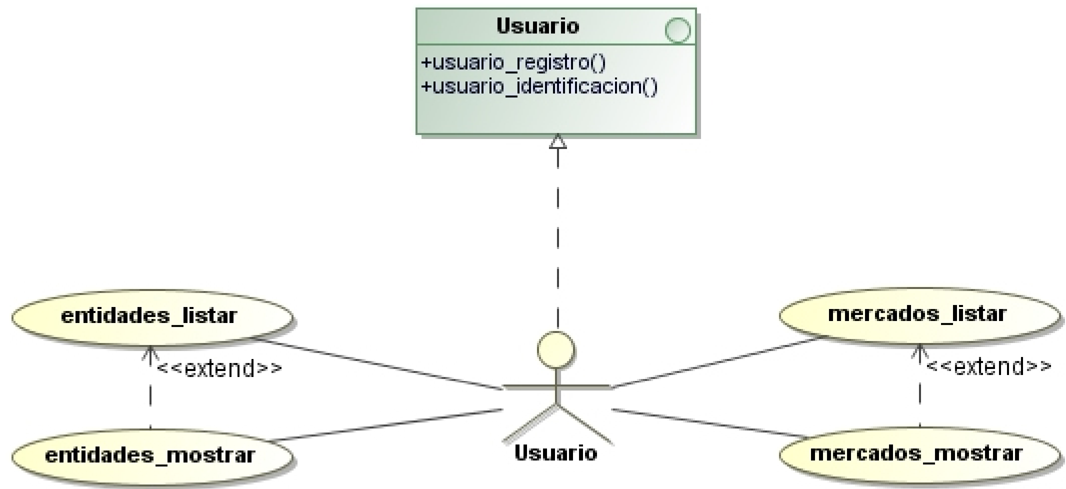
2.3.1 Diagrama de casos de uso.

2.3.1.1 Usuario no registrado.

El usuario no registrado puede ser cualquier persona que se conecte a la pagina web y no se identifique.

Las funciones que ofrece el sistema para este tipo de usuario es limitada, ya que para poder tener un seguimiento de la aplicación por parte del usuario sus acciones realizadas han de ir asociadas a una cuenta.

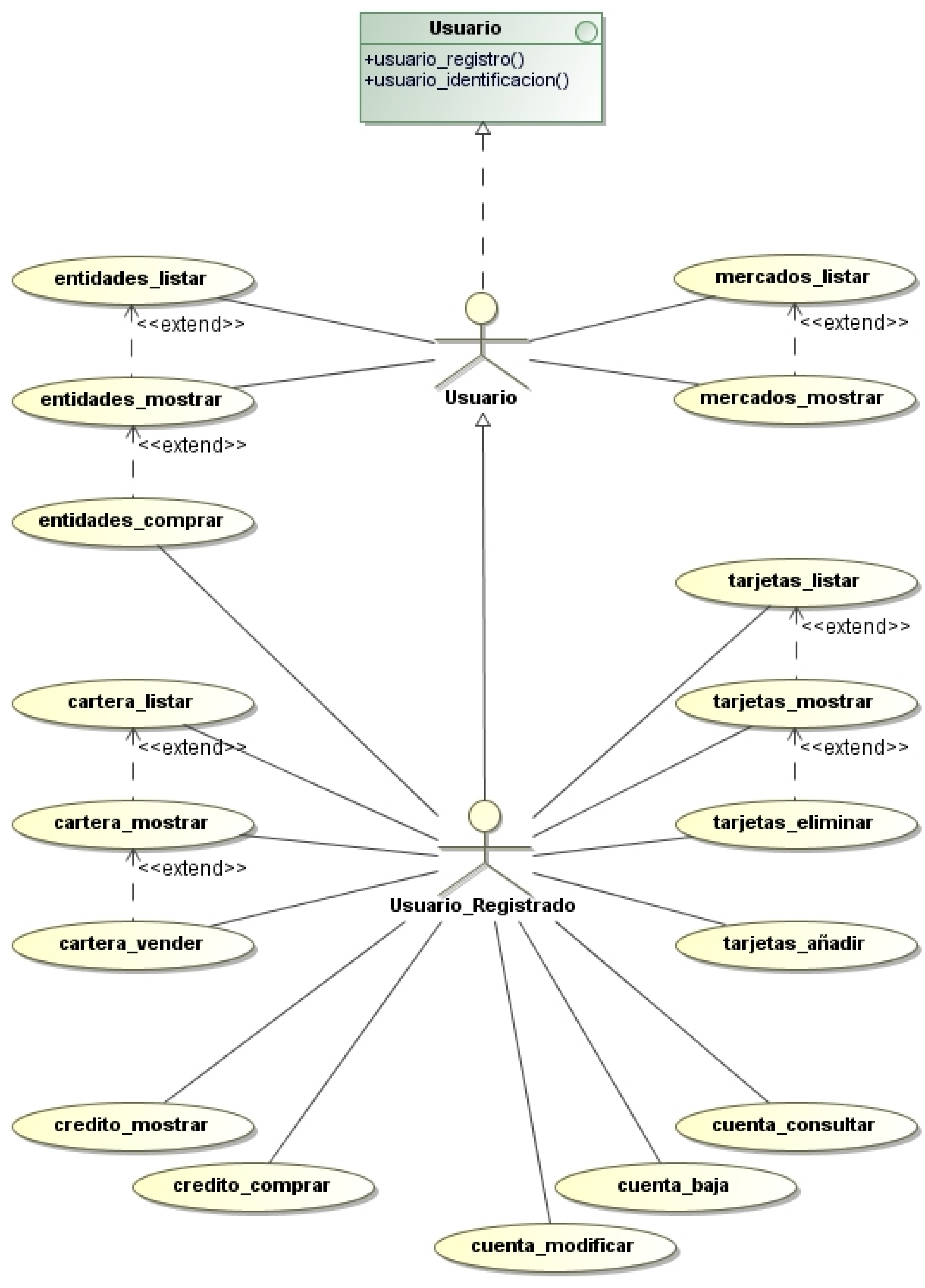
Existen un par de acciones que están solo disponibles para este tipo de usuario, usuario registro, y usuario identificación.



2.3.1.2 Usuario registrado.

Un usuario registrado es todo usuario que haya realizado un registro de usuario y se haya identificado en la actual sesión.

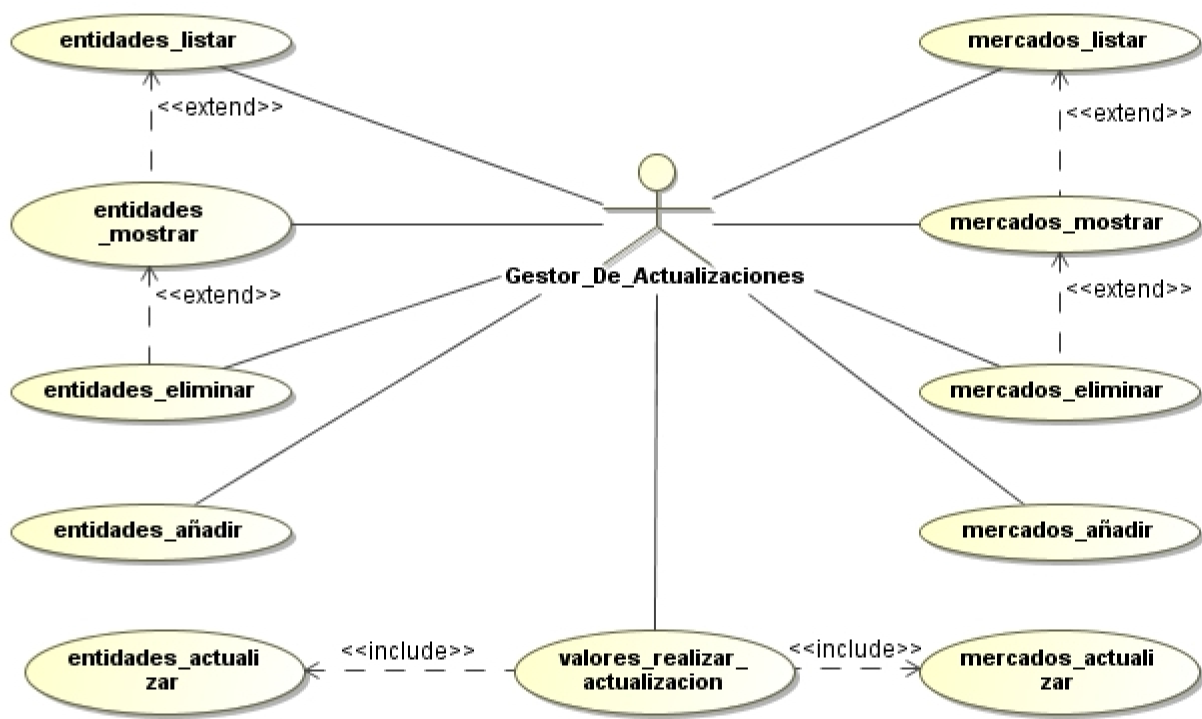
La aplicación ofrece a este tipo de usuarios un conjunto de acciones que no son ofrecidas a un usuario anónimo.



2.3.1.3 Software de actualización

Este tipo de usuario es el encargado de mantener el sistema actualizado y mantener un registro histórico de los valores. Se implementa en un programa que realiza frecuentes actualizaciones en la base de datos.

El software de actualización es usado solamente por un único servidor de actualizaciones.



2.4 Descripción textual de casos de uso

2.4.1 Caso de uso: Usuario registro

Resumen de la funcionalidad: Permite a un usuario anónimo obtener una cuenta de usuario registrado con la que tener total acceso a la aplicación.

Papel dentro del trabajo del usuario: Habitual.

Actores: Usuario anónimo.

Casos de uso relacionados: Usuario identificación, Usuario baja, Usuario desconexión, Usuario modificación.

Precondición: El alias seleccionado no puede existir en la base de datos.
Poscondición: Se crea el usuario indicado, en el sistema.

Flujo de eventos principal:

- El usuario escoge selecciona la opción de registro de usuario.
- El sistema muestra un formulario al usuario.
- El usuario rellena los datos del formulario y pulsa aceptar
- El sistema almacena los datos en el la base de datos y le asigna al usuario un saldo inicial gratuito.

Flujos alternativos:

- Si el sistema detecta que el alias ya esta siendo usado lanza un mensaje de error.
- Si alguno de los datos obligatorios no es introducido, el sistema lanza un mensaje de error.

2.4.2 Caso de uso: Usuario identificación

Resumen de la funcionalidad: Permite a un usuario anónimo identificarse en el sistema y tener acceso con una cuenta de usuario registrado.

Papel dentro del trabajo del usuario: Habitual.

Actores: Usuario anónimo.

Casos de uso relacionados: Usuario baja, Usuario desconexión, Usuario modificación, Usuario registro.

Precondición: El usuario tiene que estar dado de alta en el sistema.
Poscondición: El usuario pasa a tener una cuenta de usuario registrado.

Flujo de eventos principal:

- El usuario selecciona la opción de identificación de usuario
- El sistema muestra un formulario donde el usuario ha de meter su alias y contraseña
- El usuario introduce los datos y presiona aceptar.
- El usuario pasa a tener una cuenta de usuario registrado.

Flujos alternativos:

- El sistema mostrara un mensaje de error si la contraseña no coincide con la que tiene el usuario en la base de datos,.
- El sistema mostrara un mensaje de error si el usuario no introduce alguno de los dos campos.

2.4.3 Caso de uso: Mercados listar

Resumen de la funcionalidad: Permite listar los mercados disponibles en la aplicación. La aplicación solo tiene como mercado de valores el IBEX 35, pero dejamos así la opción de ampliar el numero de mercados en el futuro sin realizar apenas cambios.
Papel dentro del trabajo del usuario: Habitual.

Actores: Usuario anónimo, Usuario registrado, Software de actualización.

Casos de uso relacionados: Mercados mostrar.

Precondición:
Poscondición: El sistema mostrara un listado de los mercados que tiene almacenados

Flujo de eventos principal:

- El usuario selecciona la opción de listar mercado.
- El sistema muestra un listado de mercados almacenados en la base de datos.

Flujos alternativos:

- El sistema no tiene ningún mercado almacenado en la base de datos.

2.4.4 Caso de uso: Mercados mostrar

Resumen de la funcionalidad: Permite obtener información del mercado seleccionado.

Papel dentro del trabajo del usuario: Habitual.

Actores: Usuario anónimo, Usuario registrado, Software de actualización.

Casos de uso relacionados: Mercados listar.

Precondición: El mercado seleccionado ha de estar almacenado en la base de datos.
Poscondición: El sistema muestra al usuario la información del mercado seleccionado.

Flujo de eventos principal:

- El usuario selecciona el mercado que desea consultar de la lista de mercados facilitada por el ordenador y presiona aceptar.
- El sistema devuelve al usuario los datos del mercado seleccionado.

Flujos alternativos:

- El sistema mostrara error si el mercado no se encuentra en la base de datos.

2.4.5 Caso de uso: Entidades listar

Resumen de la funcionalidad: Permite listar las entidades pertenecientes al mercado seleccionado. Como se indico antes antes, la aplicación solo tiene como mercado de valores el IBEX 35, pero dejamos así la opción de ampliar el numero de mercados en el futuro sin realizar apenas cambios.

Papel dentro del trabajo del usuario: Habitual.

Actores: Usuario anónimo, Usuario registrado, Software de actualización.

Casos de uso relacionados: Mercados listar, Mercados Mostrar, Entidades mostrar.

Precondición: El usuario ha de seleccionar un mercado existente
Poscondición: El sistema muestra al usuario el listado de entidades pertenecientes a el mercado seleccionado.

Flujo de eventos principal:

- El usuario selecciona un mercado de los disponibles en la base de datos e indica al sistema que quiere ver las entidades pertenecientes a este.
- El sistema muestra al usuario el listado de las entidades pertenecientes al mercado seleccionado

Flujos alternativos:

- El sistema mostrara error si el mercado indicado no se encuentra en el sistema
- El sistema mostrara error si el mercado indicado no contiene entidades

2.4.6 Caso de uso: Entidades mostrar

Resumen de la funcionalidad: Permite obtener información de la entidad seleccionada.

Papel dentro del trabajo del usuario: Habitual.

Actores: Usuario anónimo, Usuario registrado, Software de actualización.

Casos de uso relacionados: Mercados listar, Mercados Mostrar, Entidades listar.

Precondición: La entidad seleccionada debe estar almacenada en el sistema.

Poscondición: El sistema muestra al usuario los datos de la entidad seleccionada.

Flujo de eventos principal:

- El usuario selecciona una entidad de la lista e indica al sistema que quiere obtener información acerca de esta.
- El sistema muestra al usuario la información de la entidad seleccionada.

Flujos alternativos:

- El sistema muestra error si la entidad seleccionada no se encuentra en el sistema.

2.4.7 Caso de uso: Tarjetas añadir

Resumen de la funcionalidad: Un usuario registrado puede añadir una tarjeta de crédito a su lista de tarjetas disponibles para comprar crédito.

Papel dentro del trabajo del usuario: Habitual.

Actores: Usuario registrado.

Casos de uso relacionados:

Precondición: El usuario no puede tener añadida esta tarjeta previamente

Poscondición: El usuario añade la tarjeta indicada a la lista de sus tarjetas de crédito disponibles.

Flujo de eventos principal: Usuario modificación, Tarjetas eliminar, Tarjetas mostrar, Tarjetas listar.

- El usuario selecciona la opción de añadir tarjeta
- El sistema muestra al usuario un formulario para que indique los datos de su tarjeta.
- El usuario introduce los datos de su tarjeta y presiona aceptar.
- El sistema añade la tarjeta indicada al listado de tarjetas del usuario

Flujos alternativos:

- El sistema indica error si el usuario ya tenia la tarjeta indicada en su lista de tarjetas de crédito.
- El sistema dará error si los datos de la tarjeta no han sido introducidos correctamente.

2.4.8 Caso de uso: Tarjetas eliminar

Resumen de la funcionalidad: Un usuario registrado puede eliminar una tarjeta de crédito de su lista de tarjetas disponibles para comprar crédito.

Papel dentro del trabajo del usuario: Habitual.

Actores: Usuario registrado.

Casos de uso relacionados: Usuario modificación, Tarjetas añadir, Tarjetas mostrar, Tarjetas listar.

Precondición: La tarjeta indicada ha de estar en la lista de tarjetas de crédito del usuario.

Poscondición: La tarjeta indicada se elimina del listado de tarjetas de crédito que el usuario tenia disponible para la compra de crédito.

Flujo de eventos principal:

- El usuario selecciona del listado de sus tarjetas la tarjeta que desea eliminar.
- El sistema pregunta al usuario si esta seguro de que desea eliminar esa tarjeta.
- El usuario confirma su acción.
- El sistema elimina la tarjeta del listado de tarjetas de crédito de el usuario y confirma el resultado de la operación.

Flujos alternativos:

- El sistema mostrara error si la tarjeta que el usuario indica no se encuentra entre el listado de sus tarjetas de crédito.
- El sistema mostrara un mensaje de operación cancelada si el usuario no confirma la acción de eliminar la tarjeta previamente indicada.

2.4.9 Caso de uso: Tarjetas mostrar

Resumen de la funcionalidad: Muestra a un usuario registrado los datos de una de sus tarjetas de crédito.

Papel dentro del trabajo del usuario: Habitual.

Actores: Usuario registrado.

Casos de uso relacionados: Usuario modificación, Tarjetas añadir, Tarjetas eliminar, Tarjetas listar.

Precondición: El usuario debe disponer de la tarjeta indicada en el listado de sus tarjetas de crédito disponible.
Poscondición: El sistema muestra al usuario los datos de la tarjeta seleccionada.

Flujo de eventos principal:

- El usuario selecciona una tarjeta del listado ofrecido por el sistema y le indica a este que muestre la información de la tarjeta seleccionada.
- El sistema muestra al usuario la información de la tarjeta seleccionada.

Flujos alternativos:

- El sistema mostrara error si la tarjeta indicada por el usuario no se encuentra en el listado de las tarjetas de crédito disponibles de este.

2.4.10 Caso de uso: Tarjetas listar

Resumen de la funcionalidad: Muestra a un usuario el listado de las tarjetas de crédito que tiene asociadas a su cuenta.

Papel dentro del trabajo del usuario: Habitual.

Actores: Usuario registrado.

Casos de uso relacionados: Usuario modificación, Tarjetas añadir, Tarjetas eliminar, Tarjetas mostrar.

Precondición:

Poscondición: El sistema muestra al usuario el listado de tarjetas de crédito asociadas a su cuenta.

Flujo de eventos principal:

- El usuario selecciona la opción de mostrar las tarjetas de crédito asociadas a su cuenta.
- El sistema muestra al usuario el listado de tarjetas de crédito asociadas a su cuenta.

Flujos alternativos:

- El sistema muestra un mensaje de no hay ninguna tarjeta disponible si el usuario no tiene ninguna tarjeta añadida.

2.4.11 Caso de uso: Entidad comprar

Resumen de la funcionalidad: Añade acciones de una determinada entidad a la cartera de acciones de un usuario a cambio de el valor en crédito de las acciones

Papel dentro del trabajo del usuario: Habitual.

Actores: Usuario registrado.

Casos de uso relacionados: Crédito mostrar, Crédito comprar, Mercados listar, Mercados Mostrar, Entidades listar, Entidades mostrar.

Precondición: El usuario debe tener suficiente crédito para comprar las acciones que solicita.

Poscondición: El usuario resta el dinero gastado a su crédito y añade a su cartera de acciones las acciones compradas.

Flujo de eventos principal:

- El usuario selecciona la opción comprar para una de las entidades del listado de entidades disponibles.
- El sistema muestra una pantalla indicando los datos de la entidad y un formulario para que el usuario indique el numero de acciones que desea comprar.
- El usuario introduce el numero de acciones que desea comprar y pulsa aceptar.
- El sistema resta el dinero gastado a la cuenta del usuario y añade a la cartera de acciones del usuario las acciones adquiridas en la compra.
- El sistema indica al usuario que su compra ha sido realizada correctamente.

Flujos alternativos:

- El sistema muestra una pantalla de error, si la entidad seleccionada no se encuentra en el sistema.
- El sistema muestra una pantalla de error, si el usuario no dispone de dinero suficiente para comprar las acciones solicitadas.

2.4.12 Caso de uso: Cartera vender

Resumen de la funcionalidad: Añade el valor actual de las acciones vendidas al crédito del usuario.

Papel dentro del trabajo del usuario: Habitual.

Actores: Usuario registrado.

Casos de uso relacionados: Entidad mostrar, Entidad listar, Entidad comprar, Cartera mostrar, Cartera listar.

Precondición: El usuario debe tener alguna acción en su cartera. El usuario no podrá vender mas acciones que las que posee.

Poscondición: El usuario suma a su saldo el valor de las acciones vendidas.

Flujo de eventos principal:

- El usuario selecciona una entidad de la cual posee acciones.
- El sistema muestra un menú al usuario informándole de el precio actual de venta y donde puede indicar el número de acciones que puede vender.
- El usuario selecciona el número de acciones que quiere vender.
- El sistema pide al usuario confirmación sobre la acción.
- El usuario confirma su operación.
- El sistema añade al crédito el precio de las acciones vendidas teniendo en cuenta el precio actual d estas y elimina las acciones vendidas de la cartera de valores del usuario.
- El sistema confirma la venta

Flujos alternativos:

- El sistema muestra una pantalla de error, si las acciones seleccionadas no se encuentran en la cartera del usuario.
- El sistema muestra error, intenta vender mas acciones de las que posee.
- El sistema mostrara un mensaje de operación cancelada si el usuario no confirma la acción de vender las acciones indicadas.

2.4.13 Caso de uso: Cartera mostrar

Resumen de la funcionalidad: Muestra los datos de la compra de acciones seleccionada de nuestra cartera de valores.

Papel dentro del trabajo del usuario: Habitual.

Actores: Usuario registrado.

Casos de uso relacionados: Cartera listar, Cartera vender.

Precondición: El usuario ha de tener alguna acción en su cartera de inversiones.
Poscondición: El sistema muestra al usuario la información sobre el las acciones seleccionadas.

Flujo de eventos principal:

- El usuario selecciona del listado de acciones de su cartera algún grupo de acciones y presiona aceptar.
- El sistema muestra al usuario los datos del grupo de acciones seleccionadas.

Flujos alternativos:

- El sistema muestra una pantalla de error, si las acciones seleccionadas no se encuentran en la cartera del usuario.

2.4.14 Caso de uso: Cartera listar

Resumen de la funcionalidad: Muestra un listado de las acciones que un usuario posee en su cartera de valores.

Papel dentro del trabajo del usuario: Habitual.

Actores: Usuario registrado.

Casos de uso relacionados: Cartera mostrar, Cartera vender.

Precondición:
Poscondición: El sistema muestra el listado de acciones que posee el usuario.

Flujo de eventos principal:

- El usuario selecciona la opción de listar su cartera de inversiones.
- El sistema muestra al usuario un listado con las acciones que posee.

Flujos alternativos:

- El sistema mostrara un aviso indicando que el usuario no posee acciones si este no las tiene.

2.4.15 Caso de uso: Crédito mostrar

Resumen de la funcionalidad: Muestra al usuario cuanto dinero virtual posee.

Papel dentro del trabajo del usuario: Habitual.

Actores: Usuario registrado.

Casos de uso relacionados: Crédito comprar.

Precondición:
Poscondición: El sistema muestra al usuario el crédito que tiene disponible.

Flujo de eventos principal:

- El usuario selecciona la opción de mostrar crédito disponible.
- El sistema consulta el crédito de usuario en la base de datos y se lo muestra al usuario.

Flujos alternativos:

2.4.16 Caso de uso: Crédito comprar

Resumen de la funcionalidad: El usuario añade a su crédito una cantidad de dinero virtual a cambio de dinero real.

Papel dentro del trabajo del usuario: Habitual.

Actores: Usuario registrado.

Casos de uso relacionados: Tarjetas añadir, Tarjetas listar, Tarjetas mostrar, Crédito mostrar.

Precondición: El usuario ha de tener alguna tarjeta de crédito valida en su lista de tarjetas de crédito.

Poscondición: El usuario cambia crédito real de su tarjeta de crédito por crédito virtual multiplicado por cien.

Flujo de eventos principal:

- El usuario selecciona la opción de comprar crédito virtual.
- El sistema muestra un listado de las tarjetas que el usuario tiene adjuntas a su cuenta.
- El usuario selecciona una de las tarjetas de la lista y presiona aceptar.
- El sistema muestra un aviso de las condiciones de la compra y un formulario donde el usuario pueda meter la cantidad de crédito que quiere comprar y los datos necesarios para completar la transacción.
- El sistema pregunta al usuario si desea realizar la operación.
- El usuario confirma su operación.
- El sistema carga en la tarjeta de crédito la cantidad de dinero solicitada y añade en la cuenta del usuario una cantidad de dinero igual a esa multiplicada por cien.
- El sistema muestra al usuario que la operación ha sido realizada correctamente.

Flujos alternativos:

- El sistema mostrara un error si la tarjeta indicada por el usuario no acepta la operación de cobro.
- El sistema mostrara un mensaje de operación cancelada si el usuario no confirma la acción de comprar crédito.

2.4.17 Caso de uso: Usuario modificar

Resumen de la funcionalidad: El usuario modifica sus datos personales.

Papel dentro del trabajo del usuario: Habitual.

Actores: Usuario registrado.

Casos de uso relacionados: Tarjetas añadir, Tarjetas eliminar, Tarjetas mostrar, Tarjetas listar.

Precondición: El usuario no puede cambiar su alias.

Poscondición: El usuario modifica sus datos personales.

Flujo de eventos principal:

- El usuario selecciona la opción de modificar los datos de su cuenta.
- El sistema muestra al usuario un formulario con los datos de su cuenta.
- El usuario modifica los datos de este formulario según desee.
- El sistema muestra al usuario los nuevos datos.

Flujos alternativos:

- El sistema lanza un mensaje de error si alguno de los datos obligatorios no es introducido.

2.4.18 Caso de uso: Usuario desconectar

Resumen de la funcionalidad: El usuario registrado desconecta la sesión y pasa a ser un usuario anónimo hasta que se vuelva a identificar.

Papel dentro del trabajo del usuario: Habitual.

Actores: Usuario registrado.

Casos de uso relacionados: Usuario identificación, Usuario baja, Usuario modificación, Usuario registro.

Precondición:

Poscondición: El usuario registrado pasa a ser un usuario anonimo.

Flujo de eventos principal:

- El usuario selecciona la opción de cerrar sesión.
- El sistema desconecta la sesión poniendo al usuario como un usuario anónimo y muestra la pagina principal.

Flujos alternativos:

2.4.19 Caso de uso: Mercados añadir

Resumen de la funcionalidad: Se añade un mercado de valores a la aplicación.

Papel dentro del trabajo del usuario: Habitual.

Actores: Software de actualización.

Casos de uso relacionados: Mercados listar, Mercados Mostrar, Mercados eliminar, Mercados actualizar.

Precondición: El mercado introducido no puede estar ya en el sistema.

Poscondición: Se añade al sistema el mercado indicado.

Flujo de eventos principal:

- El software de actualización lee de un archivo el listado de mercados y la dirección de donde ha de obtener los datos
- El software de actualización realiza una petición a la dirección solicitada.
- El software de actualización introduce en el sistema el mercado indicado.

Flujos alternativos:

- El sistema realiza un reporte si el mercado ya existe en el sistema.
- El sistema realiza un reporte si la conexión a la fuente no puede ser realizada.

2.4.20 Caso de uso: Mercados eliminar

Resumen de la funcionalidad: Se elimina un mercado de valores de la aplicación.

Papel dentro del trabajo del usuario: Habitual.

Actores: Software de actualización.

Casos de uso relacionados: Mercados listar, Mercados Mostrar, Mercados actualizar, Mercados añadir.

Precondición: El mercado debe estar almacenado en el sistema.

Poscondición: Se elimina el mercado indicado del sistema.

Flujo de eventos principal:

- El software de actualización selecciona el mercado que desea eliminar.
- El software de actualización paga a los usuario el dinero de las acciones de las entidades pertenecientes a el mercado indicado con el ultimo valor que tenga en el historial y luego esas acciones.
- El software de actualización elimina las entidades pertenecientes al mercado indicado
- El software de actualización elimina el mercado

Flujos alternativos:

2.4.21 Caso de uso: Mercados actualizar

Resumen de la funcionalidad: Actualiza los datos del mercado seleccionado.

Papel dentro del trabajo del usuario: Habitual.

Actores: Software de actualización.

Casos de uso relacionados: Mercados listar, Mercados Mostrar, Mercados eliminar, Mercados añadir.

Precondición: El sistema indicado ha de estar almacenado en el sistema.

Poscondición: Los datos del mercado son actualizados.

Flujo de eventos principal:

- El software de actualización realiza una petición a la dirección del mercado y obtiene los datos actualizados.
- El software de actualización actualiza los datos del sistema.

Flujos alternativos:

- El sistema realiza un reporte si la conexión a la fuente no puede ser realizada.

2.4.22 Caso de uso: Entidades añadir

Resumen de la funcionalidad: Se añade una entidad al mercado seleccionado y realiza un registro histórico del valor.

Papel dentro del trabajo del usuario: Habitual.

Actores: Software de actualización.

Casos de uso relacionados:

Precondición: La entidad indicada no esta en el sistema.

Poscondición: La entidad indicada se añade al sistema.

Flujo de eventos principal:

- El software de actualización realiza una petición a la dirección del mercado y obtiene los datos actualizados.
- El software de actualización inserta los datos de la entidad en el sistema.

Flujos alternativos:

- El sistema realiza un reporte si la entidad ya existe en el sistema.
- El sistema realiza un reporte si la conexión a la fuente no puede ser realizada.

2.4.23 Caso de uso: Entidades eliminar

Resumen de la funcionalidad: Elimina una entidad del mercado seleccionado.

Papel dentro del trabajo del usuario: Habitual.

Actores: Software de actualización.

Casos de uso relacionados: Mercados listar, Mercados Mostrar, Mercados eliminar, Mercados actualizar, Mercados añadir, Entidades listar, Entidades mostrar, Entidades añadir, Entidades actualizar.

Precondición: La entidad seleccionada ha de estar almacenada en el sistema.
Poscondición: La entidad seleccionada se elimina del sistema.

Flujo de eventos principal:

- El software de actualización selecciona la entidad que desea eliminar.
- El software de actualización paga a los usuario el dinero de las acciones de las entidades pertenecientes a la entidad indicada con el ultimo valor que tenga en el historial y luego esas acciones.
- El software de actualización elimina la entidad indicada

Flujos alternativos:

2.4.24 Caso de uso: Entidades actualizar

Resumen de la funcionalidad: Actualiza los datos de la entidad seleccionada y realiza un registro histórico del valor.

Papel dentro del trabajo del usuario: Habitual.

Actores: Software de actualización.

Casos de uso relacionados: Mercados listar, Mercados Mostrar, Mercados eliminar, Mercados actualizar, Mercados añadir, Entidades listar, Entidades mostrar, Entidades eliminar, Entidades añadir.

Precondición: La entidad indicada ha de estar almacenada en el sistema.

Poscondición: Los datos de la entidad indicada son actualizados

Flujo de eventos principal:

- El software de actualización realiza una petición a la dirección del mercado y obtiene los datos actualizados.
- El software de actualización actualiza los datos del sistema.

Flujos alternativos:

- El sistema realiza un reporte si la conexión a la fuente no puede ser realizada.

2.4.25 Caso de uso: Realizar actualización

Resumen de la funcionalidad: Actualiza los datos de los datos de mercados y entidades del sistema.

Papel dentro del trabajo del usuario: Habitual.

Actores: Software de actualización.

Casos de uso relacionados: Mercados listar, Mercados Mostrar, Mercados eliminar, Mercados actualizar, Mercados añadir, Entidades listar, Entidades mostrar, Entidades eliminar, Entidades añadir, Entidades actualizar.

Precondición:
Poscondición: El sistema actualiza todos los datos de mercados y entidades de la lista de actualización.

Flujo de eventos principal:

- El software de actualización lee el archivo donde se encuentra la lista de mercados que ha de actualizar.
- El software de actualización realiza las peticiones a las fuentes indicadas para cada mercado.
- El software de actualización introduce en el sistema las actualizaciones de los datos obtenidos.

Flujos alternativos:

- El sistema realiza un reporte si la conexión a la fuente no puede ser realizada.

3 Análisis

3.1 Especificación de las clases de análisis.

El diagrama de clases muestra la estructura estática de las clases en un dominio (porción del mundo real considerada por una aplicación); se muestran las clases y las relaciones entre ambas, que pueden ser de herencia, asociación, agregación o uso.

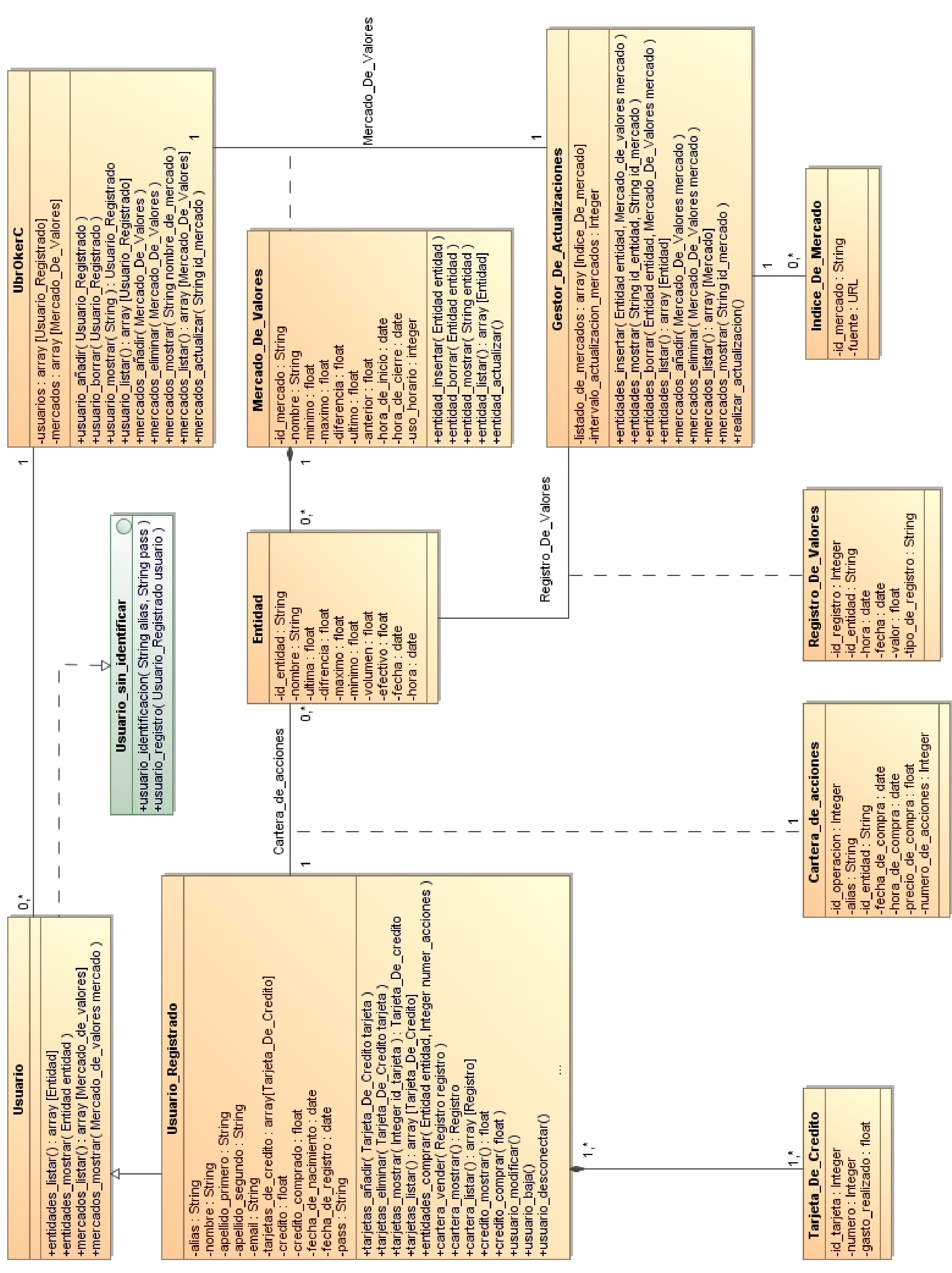
3.1.1 Identificación de casos de entidades.

La identificación de las clases de entidad consiste en hacer un esquema de las clases que necesitaremos y mediante las cuales se puedan especificar los casos de uso.

- Caso de uso “tarjetas_añadir”: Usuario registrado.
- Caso de uso “tarjetas_eliminar”: Usuario registrado.
- Caso de uso “tarjetas_mostrar”: Usuario registrado.
- Caso de uso “tarjetas_listar”: Usuario registrado.
- Caso de uso “entidades_comprar”: Usuario registrado.
- Caso de uso “entidades_listar”: Usuario registrado, Usuario anónimo.
- Caso de uso “entidades_mostrar”: Usuario registrado, Usuario anónimo.
- Caso de uso “mercados_listar”: Usuario registrado, Usuario anónimo.
- Caso de uso “mercados_mostrar”: Usuario registrado, Usuario anónimo.
- Caso de uso “cartera_vender”: Usuario registrado.
- Caso de uso “cartera_mostrar”:Usuario registrado.
- Caso de uso “cartera_listar”:Usuario registrado.
- Caso de uso “credito_mostrar”:Usuario registrado.
- Caso de uso “credito_comprar”:Usuario registrado.
- Caso de uso “usuario_modificar”: Usuario registrado.
- Caso de uso “usuario_desconectar”: Usuario registrado.
- Caso de uso “usuario_identificacion”: Usuario anónimo.
- Caso de uso “usuario_registro”: Usuario anónimo.
- Caso de uso “entidades_insertar”: Software de actualizaciones.
- Caso de uso “entidades_mostrar”: Software de actualizaciones.
- Caso de uso “entidades_borrar”: Software de actualizaciones.
- Caso de uso “entidades_listar”: Software de actualizaciones.
- Caso de uso “mercados_añadir”: Software de actualizaciones.
- Caso de uso “mercados_eliminar”: Software de actualizaciones.
- Caso de uso “mercados_listar”: Software de actualizaciones.
- Caso de uso “mercados_mostrar”: Software de actualizaciones.
- Caso de uso “realizar_actualizacion”: Software de actualizaciones.

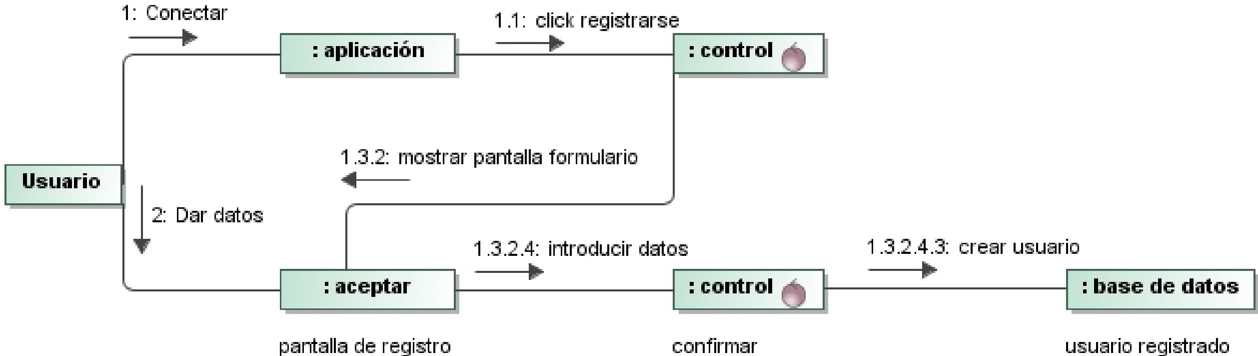
3.1.2 Especificación de atributos y clases de entidad: Diagrama

Muestra el conjunto de las clases que forman el modelo de datos y las relaciones que presenta estas entre si.

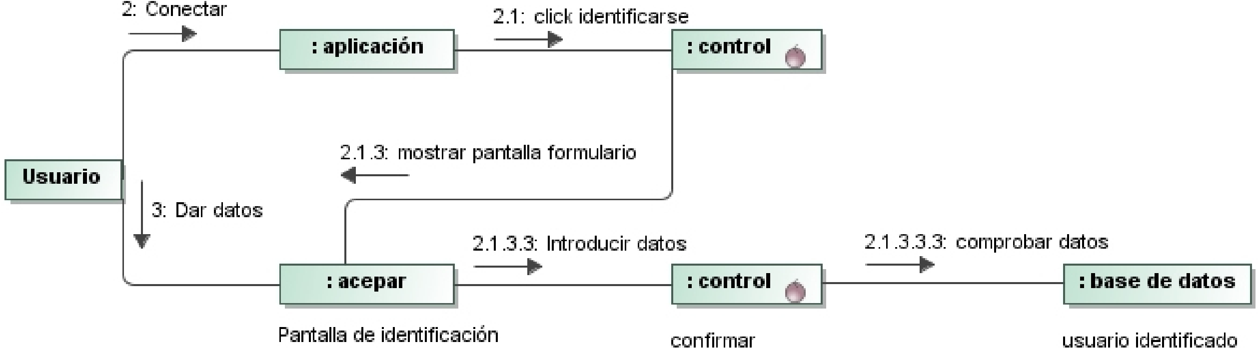


3.2 Diagramas de colaboración, clases frontera, clases de control y operaciones.

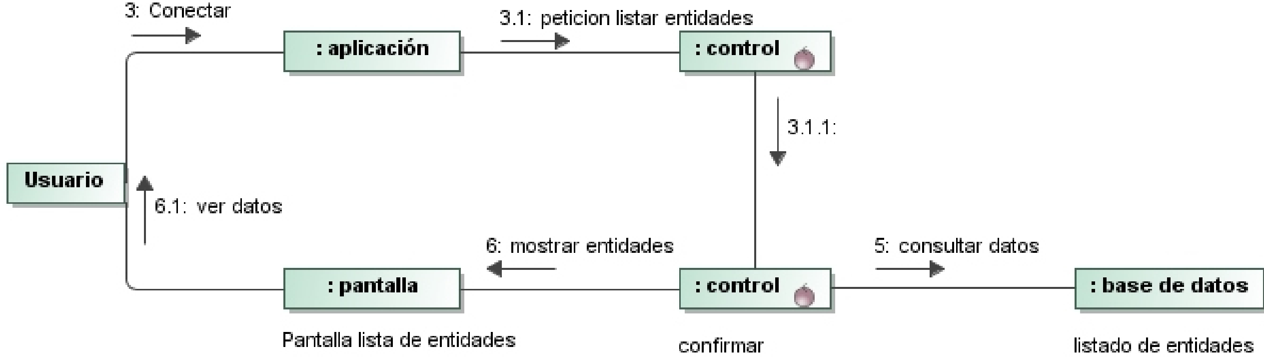
3.2.1 Diagrama de colaboración registro



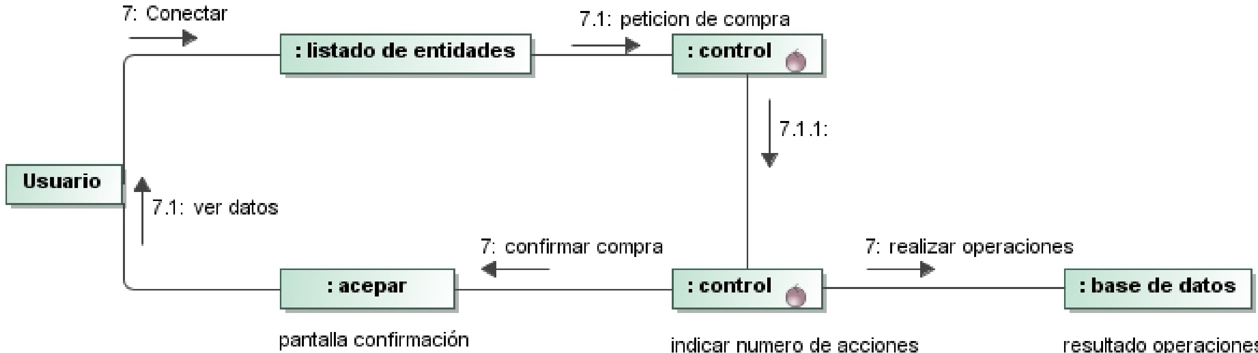
3.2.2 Diagrama de colaboración identificación



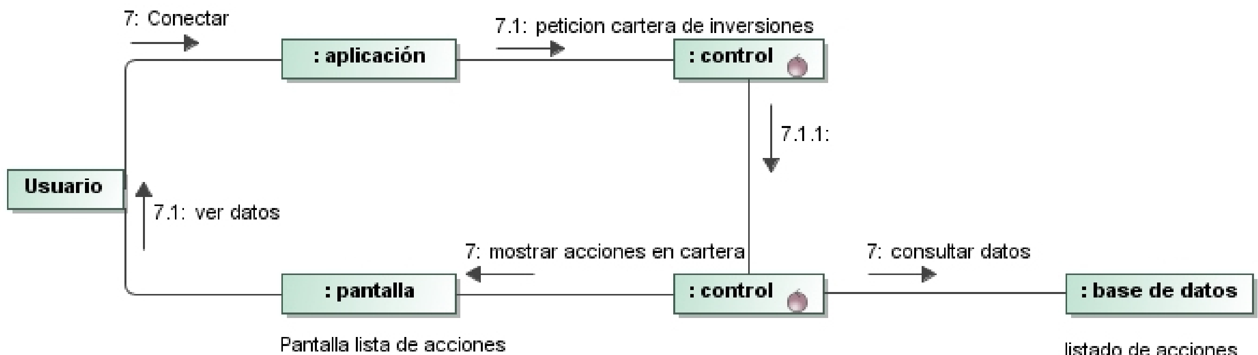
3.2.3 Diagrama de colaboración listar entidades



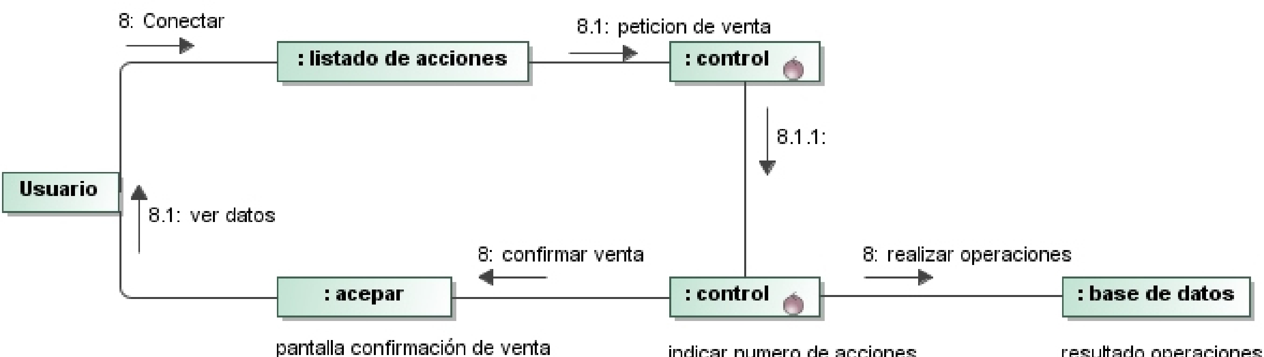
3.2.4 Diagrama de colaboración compra de acciones.



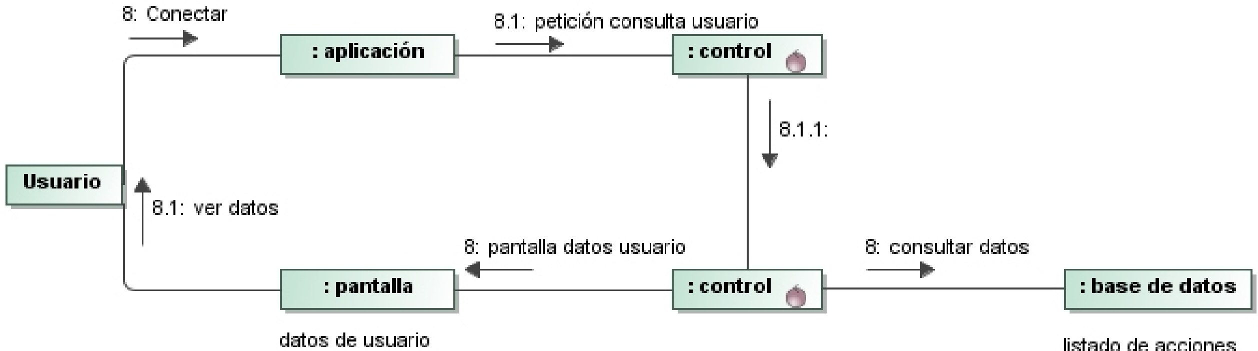
3.2.5 Diagrama de colaboración listar cartera de acciones



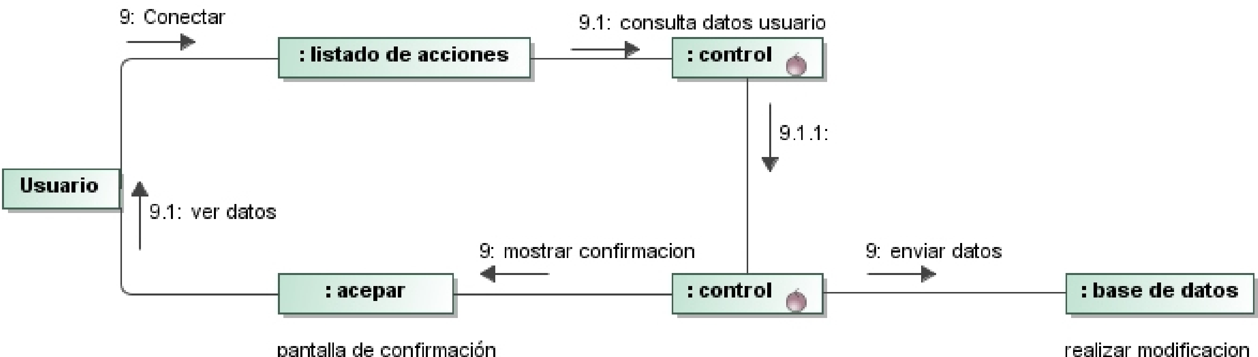
3.2.6 Diagrama de colaboración venta de acciones



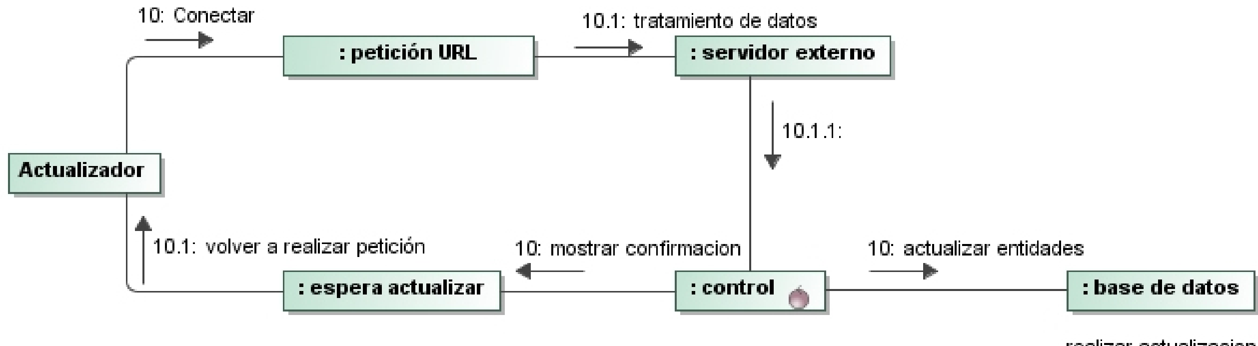
3.2.7 Diagrama de colaboración consulta de datos de usuario



3.2.8 Diagrama de colaboración modificar datos de usuario

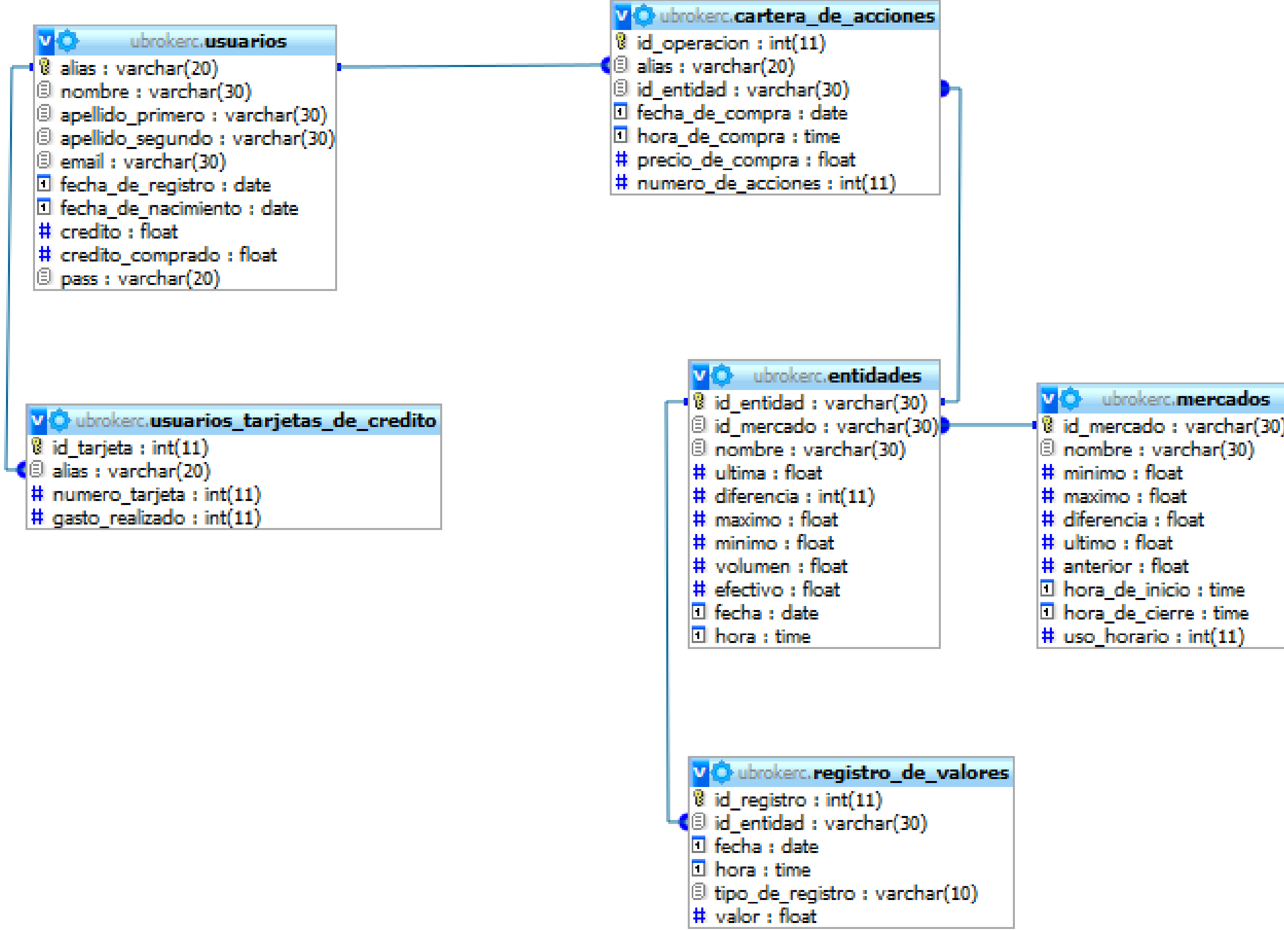


3.2.9 Diagrama de colaboración actualizar entidades



3.3 Diseño de persistencia

Para almacenar la información generada por la aplicación usaremos la base de datos MySql que nos ofrece una gran flexibilidad y una gran fiabilidad. Utilizaremos para su manejo Hibernate que como ya se indico es una herramienta de Mapeo objeto-relacional (ORM) para la plataforma Java que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML) o anotaciones en los beans de las entidades que permiten establecer estas relaciones.



3.4 Diseño

3.4.1 Diseño arquitectónico

En el esta parte se establecerán los subsistemas y módulos que formaran la aplicación.
Al dividir la aplicación en subsistemas y módulos su desarrollo resulta mas claro y fácil de implementar.

3.4.2 Identificación de subsistemas

Dividiremos nuestra aplicación en cuatro subsistemas.

3.4.2.1 Subsistema de consulta

Se encarga de mostrar tanto a los usuarios como al sistema de actualizaciones, los datos de los mercados y entidades que solicitan.

Ademas se encarga de proporcionar información sobre el registro histórico de los valores de las entidades.

3.4.2.2 Subsistema de actualización:

Se encarga de la actualización, inserción y eliminación la información correspondiente los mercados y entidades, así como los registros históricos de cada una.

3.4.2.3 Subsistema de usuarios:

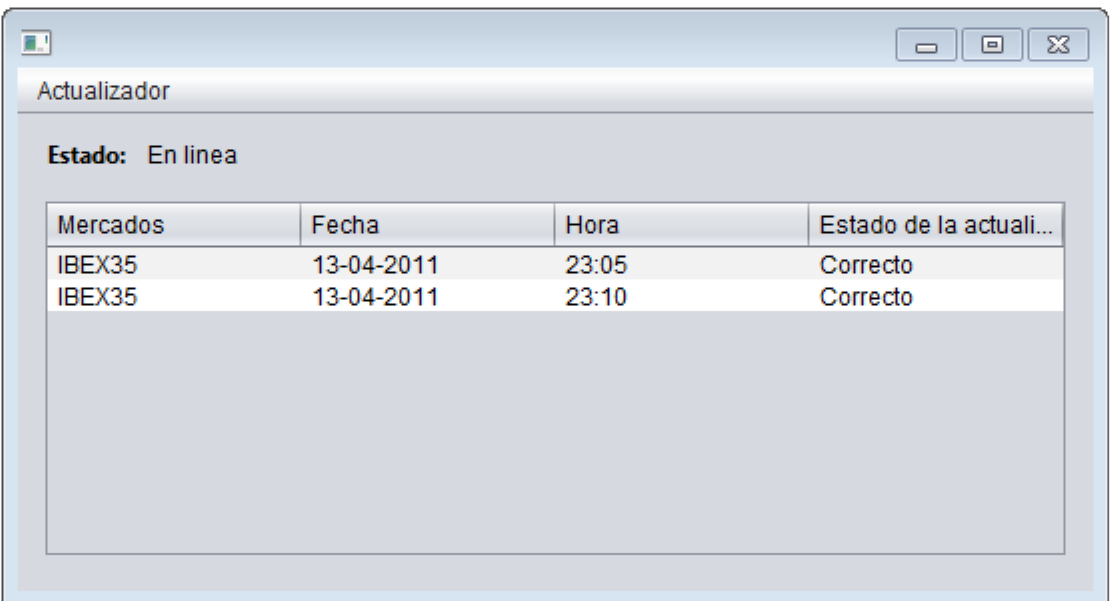
Se encarga de todas las acciones relacionadas con el manejo de los usuarios. Identificación, registro, modificación de datos, baja, gestión de tarjetas de crédito etc.

3.4.2.4 Subsistema de compra venta

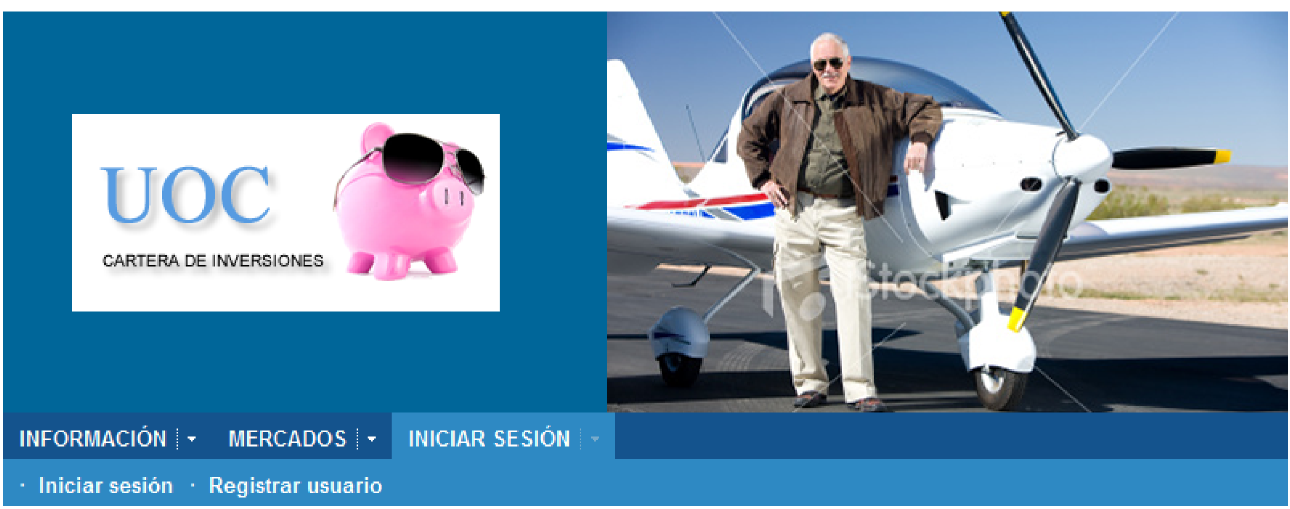
Este subsistema se encarga de realizar las operaciones de compra y venta de las acciones.

3.5 Diseño de la interfaz de la aplicación

3.5.1 Actualizador de la aplicación:



3.5.2 Usuario registro: Permite a un usuario anónimo obtener una cuenta de usuario registrado con la que tener total acceso a la aplicación.



Condiciones de registro

El acceso y navegación en esta web y/o el uso de los servicios ofrecidos en este lugar implica que el usuario acepta la política de privacidad, el aviso legal y las condiciones de uso de este servicio. En caso de no aceptarlos, no debe usar la aplicación.

Para poder completar el registro es necesario que el usuario solicitante introduzca todos los datos que le son requeridos.

Registro de usuario

Identificador :

Contraseña :

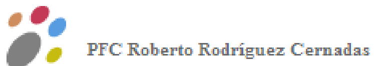
Nombre :

Primer apellido :

Segundo apellido :

Correo Electrónico :

Crear usuario Cancelar



3.5.3 Tarjetas eliminar, Tarjetas mostrar, Tarjetas listar:

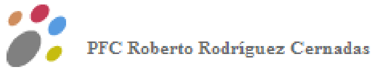
Tarjetas de credito

A continuación se muestra el listado de tarjetas de credito que el usuario ha indicado para comprar credito virtual para esta aplicación. El cambio es actualmente de 1 € real a 100 € virtuales.

Si desea añadir alguna tarjeta a su lista puede hacerlo pinchando aqui..

Nombre	Número de tarjeta	Saldo cargado	Eliminar tarjeta	Comprar credito
BBVA	**** **** **** *123	50,00		
Banco Pastor	**** **** **** *553	40,00		

Saldo del usuario: 600.000 €



3.5.4 Mercados listar, Mercados mostrar, Entidades listar, Entidades mostrar : (Registrado)

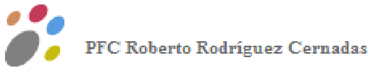
IBEX 35

Índice	Anterior	Último	Dif %	Max.	Min.
IBEX 35	23,5250	-1,98	24,0400	23,5200	270.090

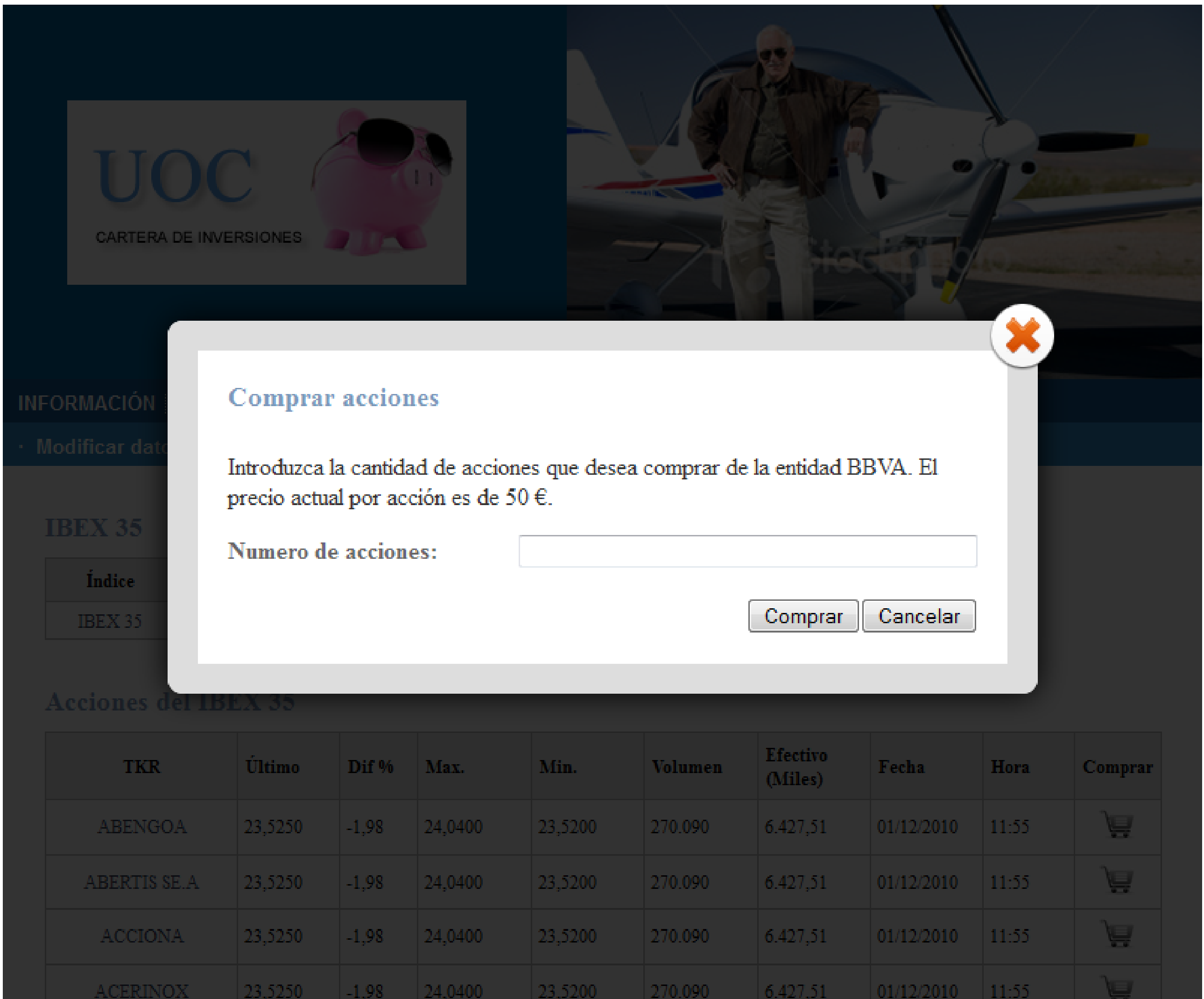
Acciones del IBEX 35

TKR	Último	Dif %	Max.	Min.	Volumen	Efectivo (Miles)	Fecha	Hora	Comprar
ABENGOA	23,5250	-1,98	24,0400	23,5200	270.090	6.427,51	01/12/2010	11:55	
ABERTIS SE.A	23,5250	-1,98	24,0400	23,5200	270.090	6.427,51	01/12/2010	11:55	
ACCIONA	23,5250	-1,98	24,0400	23,5200	270.090	6.427,51	01/12/2010	11:55	
ACERINOX	23,5250	-1,98	24,0400	23,5200	270.090	6.427,51	01/12/2010	11:55	
ACS	23,5250	-1,98	24,0400	23,5200	270.090	6.427,51	01/12/2010	11:55	
AMADEUS	23,5250	-1,98	24,0400	23,5200	270.090	6.427,51	01/12/2010	11:55	
ARTCELOMIT	23,5250	-1,98	24,0400	23,5200	270.090	6.427,51	01/12/2010	11:55	
BA. POPULAR	23,5250	-1,98	24,0400	23,5200	270.090	6.427,51	01/12/2010	11:55	
BA. SABADELL	23,5250	-1,98	24,0400	23,5200	270.090	6.427,51	01/12/2010	11:55	
BANKINTER	23,5250	-1,98	24,0400	23,5200	270.090	6.427,51	01/12/2010	11:55	
BBVA	23,5250	-1,98	24,0400	23,5200	270.090	6.427,51	01/12/2010	11:55	

Saldo del usuario: 600.000 €



3.5.5 Entidad comprar: Añade acciones de una determinada entidad a la cartera de acciones de un usuario a cambio de el valor en crédito de las acciones

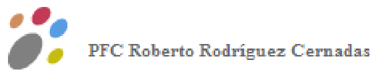


3.5.6 Cartera mostrar, Cartera listar:

Gestionar Inversiones

TKR	Acciones en cartera	Precio medio de compra	Precio de venta	Beneficio	Beneficio %	Vender
ABENGOA	300	40,00	50,00	10,00	25%	
ABERTIS SE.A	100	70,00	40,00	-30,00	-43%	
ACCIONA	50	90,00	20,00	-70,00	-88%	
ACERINOX	70	10,00	10,00	0	0%	
ACS	10.000	20,00	20,00	0	0%	

Saldo del usuario: 600.000 €



4. Desarrollo técnico

4.1 Introducción a la plataforma J2EE

* Información obtenida del libro de "Ingeniería del software de componentes y de sistemas distribuidos de la UOC".

J2EE es una plataforma de desarrollo empresarial (propuesta por Sun Microsystems en el año 1997) que define un estándar para el desarrollo de aplicaciones empresariales multicapa.

J2EE simplifica el desarrollo de estas aplicaciones basándolas en componentes modulares estandarizados, proporcionando un conjunto muy completo de servicios a estos componentes y gestionando automáticamente muchas de las funcionalidades o características complejas que requiere cualquier aplicación empresarial (seguridad, transacciones, etc.), sin necesidad de una programación compleja.

Se puede definir J2EE de esta manera:
J2EE es una plataforma abierta y estándar para desarrollar y desplegar aplicaciones empresariales multicapa con n-niveles, basadas en servidor, distribuidas y basadas en componentes.

Así pues, J2EE se enmarca dentro de un estilo arquitectónico heterogéneo, y aglutina distintas características correspondientes a estilos arquitectónicos en capas o niveles, cliente-servidor, orientada a objetos distribuidos e, incluso, orientada a servicios (tenéis la definición de todos estos estilos arquitectónicos en el módulo 2).

De esta definición, podemos extraer los conceptos básicos y los puntos clave que hay detrás de la plataforma J2EE:

- J2EE es una plataforma abierta y estándar.
- No es un producto, sino que define un conjunto de estándares que todos los contenedores deben cumplir para comunicarse con los componentes.
- En algunos ámbitos, se dice que J2EE es una especificación de especificaciones.
- J2EE define un modelo de aplicaciones distribuido y multicapa con n-niveles.
- Con este modelo, podemos dividir las aplicaciones en partes y cada una de estas partes se puede ejecutar en distintos servidores.
- La arquitectura J2EE define un mínimo de tres capas: la capa cliente, la capa intermedia y la capa de sistemas de información de la empresa (EIS - Enterprise Information Systems).
- J2EE basa las aplicaciones empresariales en el concepto de componentes modulares y estandarizados.
- Este concepto está muy vinculado al concepto contenedor.
- Los contenedores son entornos estándar de ejecución que proporcionan un conjunto de servicios a los componentes que ahorran mucho trabajo a la hora de desarrollar los componentes.

Para realizar una aplicación empresarial y que su coste de mantenimiento sea lo mas bajo posible es necesario aplicar ciertas técnicas de programación. Estas técnicas son conocidas como patrones y de entre ellas el patrón MVC que nos facilitara enormemente el desarrollo y posteriores ampliaciones que vayamos a hacer sobre la aplicación.

4.2 MVC * Información obtenida del libro de “Struts 2 El framework de desarrollo de aplicaciones J2EE de Jérôme LAFOSE”.

Como ya se ha explicado, es recomendable utilizar el modelo de diseño Modelo Vista Controlador (MVC) para el desarrollo de aplicaciones Web en Java.

Antes de comenzar el diseño, es importante comprender el funcionamiento de este modelo de desarrollo.

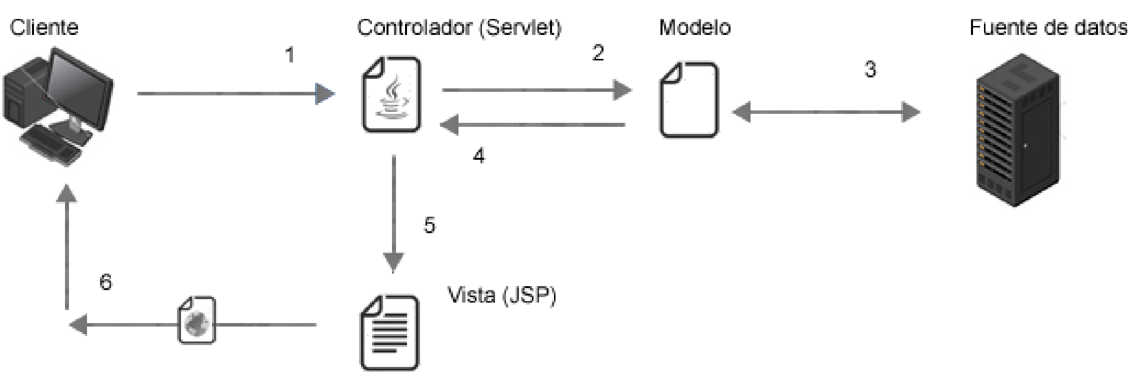
La arquitectura MVC que ofrece Sun es la solución de desarrollo Web del lado del servidor que permite separar la parte lógica de la presentación en una aplicación de Internet. Este es un punto esencial del desarrollo de proyectos ya que permite a todo el equipo trabajar por separado (cada usuario gestiona sus propios archivos, sus programas de desarrollo y sus componentes).

Esta arquitectura tiene su origen en el lenguaje SmallTalk a principios de la década de 1980, por lo tanto no es un nuevo modelo (design pattern) únicamente vinculado a Java EE. El objetivo principal es el de dividir la aplicación en tres partes distintas: el modelo, la vista y el controlador.
En la arquitectura MVC nos encontramos con:

El modelo representado por los EJB y/o JavaBeans y/o sistemas de persistencia (Hibernate, objetos serializados en XML, almacenamiento de datos por medio de JDBC...).

La vista representada por los JSP o clases SWING.

El controlador representado por los Servlets o clases Java.



Principio de funcionamiento de la arquitectura MVC

- El cliente envía una consulta HTTP al servidor. En general, esta consulta es un Servlet (o un programa ejecutable del lado del servidor) que procesa la solicitud.
- El Servlet recupera la información transmitida por el cliente y delega el procesamiento a un componente adaptado.
- Los componentes del modelo manipulan o no los datos del sistema de información (lectura, escritura, actualización, eliminación).
- Una vez finalizados los procesamientos, los componentes le devuelven el resultado al

Servlet. El Servlet entonces almacena el resultado en el contexto adaptado (sesión, consulta, respuesta...).
- El Servlet llama a la página JSP adecuada que puede acceder al resultado.
- El JSP se ejecuta, utiliza los datos transmitidos por el Servlet y genera la respuesta al cliente.

En proyectos simples, las consultas HTTP las administran los componentes Web que reciben las consultas, crean las respuestas y las devuelven a los clientes. En este caso tenemos un único componente responsable de la lógica de visualización, de la lógica empresarial y de la lógica de persistencia. En la arquitectura anterior, la visualización y la manipulación de los datos se mezclan en un único componente de Servlet. Esto puede ser en gran medida adecuado para un servicio específico no evolutivo y simple, pero puede convertirse en un problema cuando el sistema se desarrolla. En esta arquitectura se introduce código Java y código HTML en los Servlets o JSP. Existen varias soluciones a este problema. La más sencilla tiene que ver con la aparición de las páginas JSP y consiste en crear archivos de encabezado, de pie de página, de procesamiento... e incluir todo en una página general.
La arquitectura MVC separa la lógica empresarial de la visualización. En este modelo, un componente se encarga de recibir las consultas (Servlets), otro procesa los datos (Clases) y un tercero administra la visualización (JSP). Si la interfaz entre estos tres componentes está claramente definida, será más fácil modificar un componente sin afectar a los otros dos.

En una aplicación Web evolucionada, la lógica MVC es la siguiente:

El cliente emite consultas al servidor. Cada acción precisa corresponde a un Servlet que redirige las consultas a una página JSP adecuada, realiza un procesamiento o accede a los datos y, en este caso, inicia otro programa que se encargará de responder a la solicitud del usuario actual.

4.2.1 Patron MVC

El patrón Modelo-Vista-Controlador de arquitectura de software se encarga de separar los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos.

4.2.1.1 Estructura

- **Modelo**: Esta es la representación específica de la información con la cual el sistema opera. El modelo se limita a lo relativo de la vista y su controlador facilitando las presentaciones visuales complejas. El sistema también puede operar con más datos no relativos a la presentación, haciendo uso integrado de otras lógicas de negocio y de datos afines con el sistema modelado.
- **Vista:** Este presenta el modelo en un formato adecuado para interactuar, usualmente la interfaz de usuario.
- **Controlador:** Este responde a eventos, usualmente acciones del usuario, e invoca peticiones al modelo y a la vista.

4.2.1.2 Flujo de funcionamiento

- El usuario interactúa con la interfaz de usuario de alguna forma (por ejemplo, el usuario pulsa un botón, enlace, etc.)
- El controlador recibe (por parte de los objetos de la interfaz-vista) la notificación de la acción solicitada por el usuario. El controlador gestiona el evento que llega, frecuentemente a través de un gestor de eventos (handler) o callback.
- El controlador accede al modelo, actualizándolo, posiblemente modificándolo de forma adecuada a la acción solicitada por el usuario (por ejemplo, el controlador actualiza el carro de la compra del usuario). Los controladores complejos están a menudo estructurados usando un patrón de comando que encapsula las acciones y simplifica su extensión.
- El controlador delega a los objetos de la vista la tarea de desplegar la interfaz de usuario. La vista obtiene sus datos del modelo para generar la interfaz apropiada para el usuario donde se reflejan los cambios en el modelo (por ejemplo, produce un listado del contenido del carro de la compra). El modelo no debe tener conocimiento directo sobre la vista. Sin embargo, se podría utilizar el patrón Observador para proveer cierta indirección entre el modelo y la vista, permitiendo al modelo notificar a los interesados de cualquier cambio. Un objeto vista puede registrarse con el modelo y esperar a los cambios, pero aun así el modelo en sí mismo sigue sin saber nada de la vista. El controlador no pasa objetos de dominio (el modelo) a la vista aunque puede dar la orden a la vista para que se actualice. Nota: En algunas implementaciones la vista no tiene acceso directo al modelo, dejando que el controlador envíe los datos del modelo a la vista.
- La interfaz de usuario espera nuevas interacciones del usuario, comenzando el ciclo nuevamente.

4.3 Struts 2

* Información obtenida del libro de “Struts 2 El framework de desarrollo de aplicaciones J2EE de Jérôme LAFOSE y Wikipedia”.

Struts2 proporciona características para reducirlo a configuración de acciones a través de XML inteligentes, utiliza anotaciones y proporciona más de los convenios de configuración.

Las acciones son ahora POJOs que aumenta y reduce la comprobabilidad en el marco de acoplamiento, y los campos de datos de formulario HTML son convertidos al tipo correcto de la acción a utilizar.

Aún más, la disminución de acoplamiento por petición de procesamiento se ha hecho más modular, permitiendo una serie de interceptores (personalizados o provistos por Struts2) para proporcionar tratamiento previo y posterior a las funcionalidades del sistema.
Modularidad es un tema muy común, un Mecanismo de plug-in proporciona una forma de aumentar el las funciones del framework; Clases clave dadas por el framework puede ser sustituida por clases personalizadas para proporcionar características requeridas que no se dan por defecto en los paquetes de struts; los “Tags” pueden utilizar una gran variedad de renderizado Temas (incluyendo temas personalizados), y hay muchos diferentes

Tipos de resultados disponibles para la realización de la acción después de la ejecución de tareas Que incluyen, pero no están limitadas a, hacer las páginas JSP, Velocity Freemarker y plantillas. Y, por último, la inyección de dependencia es ahora una función de primera clase, dada a través del Plug-in del framework Spring.Y un sinfín de etc’s.

Las aplicaciones de Struts tienen un archivo de configuración llamado struts.xml. Este archivo de configuración es el más importante y sustituye a la definición de los Servlets en el descriptor de implementación web.xml. Este archivo permite administrar la configuración de las acciones que se van a realizar.

Struts también tiene un archivo de propiedades presente por defecto en el archivo de la aplicación struts2-core-version.jar llamado default.properties. Este archivo utilizado por defecto contiene los textos de validación (mensajes de error y de confirmación) y los parámetros de configuración de Struts. Este archivo predeterminado es suficiente para comenzar con una aplicación simple.

Como ya hemos dicho en el capítulo anterior, Struts utiliza un filtro para realizar el enrutamiento hacia un solo controlador de administración correspondiente al modelo MVC II. El controlador de Struts es capaz de:

- Determinar el URI para la acción que se va a ejecutar.
- Utilizar una clase de acción.
- Ejecutar el método de acción de la clase si está asociada.
- Si se han introducido datos, crear un objeto y actualizarlo o posicionar los valores de los parámetros.
- Regresar a la vista (página JSP) para mostrar la respuesta.

Gracias a la utilización de un filtro y de un controlador global, no hemos escrito un controlador complejo para cada servicio (por ejemplo, administración de clientes, de artículos...) para administrar el enrutamiento de la aplicación. Lo más importante para el desarrollador es administrar las acciones asociadas a una demanda específica.

4.3.1 Funcionamiento general de Struts 2

Struts utiliza un filtro que debe estar declarado en el descriptor de implementación de nuestra aplicación web.xml. Este filtro se define en la clase org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter.

Con Struts, el método de acción de la clase se ejecuta después de que todas las propiedades hayan sido procesadas y afectadas. Por su parte, un método de acción devuelve una cadena de caracteres de tipo String. Esta cadena indica a Struts donde debe redirigirse al controlador. Por ejemplo, la cadena success indica a Struts que regrese a la página en caso de que el procesamiento haya sido correcto y la cadena error determina la página que se mostrará en caso de error.

En la mayoría de los casos, Struts redirige al usuario a una vista JSP con la ayuda de la interfaz RequestDispatcher de Java. En general, las vistas devueltas están en formato JSP, pero también pueden ser modelos de Velocity o FreeMarker. Además, Struts también puede devolver un flujo multimedia de tipo imagen, por ejemplo.

Para probar y analizar los URI, Struts utiliza el archivo de configuración llamado struts.xml. Este archivo de configuración debe colocarse en el directorio /WEB-INF/src (o /WEB-INF/classes después de la compilación) para que funcione por defecto. Todas las acciones se declararán en el archivo de enrutamiento y a cada declaración de acción le corresponde un nombre de URI. Este archivo de configuración struts.xml se lee al iniciar la aplicación. Podemos, para simplificar y para evitar volver a cargar el administrador, declarar la aplicación en modo de desarrollo. Con esta modalidad de diseño, el archivo se volverá a cargar cada vez que haya un cambio en la aplicación. Con esta etiqueta, no es necesario recargar el contenedor.

Cada declaración de acción esta asociada a una clase completamente cualificada

(nombrepaquete.nombreclase). En caso contrario, podemos definir una acción de Struts por defecto. Una clase de acción deberá tener al menos un tipo de cadena de caracteres, pero también puede tener varios (confirmación, error, consulta, lista...).

A la hora de acceder a un recurso o de ejecutar una acción, Struts utiliza este proceso de llamada:

- El cliente envía consultas desde URL adaptadas. También puede enviar parámetros dentro de la URL o mediante un formulario.
- Struts consulta su archivo de configuración struts.xml para encontrar la configuración de la acción.
- Se ejecuta cada interceptor asociado a la acción. Uno de estos interceptores es el encargado de asignar automáticamente los valores recibidos en la consulta con las propiedades de la clase de acción, en función de los nombres (por ejemplo, identificador, contrasena).
- Struts ejecuta el método en de acción asociada a la clase.
- Se devuelve el resultado adaptado al usuario solicitante.

4.3.1.1 Los interceptores de Struts 2

Un interceptor de Struts es un filtro que puede efectuar distintos procesamientos sobre una acción. El código presente en un interceptor es modulable y puede añadirse o suprimirse directamente en el archivo de configuración struts.xml. También es posible añadir código especifico a una aplicación sin recompilar el framework principal.

En este punto, es necesario comprender que un interceptor es un filtro que tiene una función específica y permite, por ejemplo, administrar las cookies, los parámetros HTTP, la depuración, la carga de archivos (upload) o incluso los alias de las acciones.

4.3.1.2 El archivo de configuración struts.xml

Una aplicación de Struts tiene un archivo de configuración struts.xml el formato XML y el archivo de propiedades predeterminado default.properties, pero también puede tener otros archivos de configuración.

En el archivo struts.xml vamos a definir la configuración general de la aplicación:

- Los parámetros de configuración de Struts.
- Las acciones.
- Los interceptores.
- Los resultados de las acciones.

El archivo de configuración struts.xml siguiente se incluye con la aplicación struts-blank.war.

4.3.1.3 Administración dinámica del mapping

Un proyecto final contiene varias declaraciones de acciones y puede limitar la legibilidad y la

facilidad de mantenimiento del archivo de configuración struts.xml. En la primera versión de Struts, el archivo de configuración podía contener muchas líneas y, en ocasiones, declaraciones prácticamente idénticas (por ejemplo, administración de los artículos, clientes, categorías...).

Para ofrecer soluciones a estos problemas, Struts ofrece ahora declaraciones de acciones en forma de expresiones regulares o modelos llamados wildcard.

El carácter * permite capturar las URL compuestas de cualquier cadena de caracteres. Así, podemos ejecutar la siguiente URL **http://localhost: 8080//TFCF01/nr_sesionIniciar_mostrar** para mostrar el formulario de creación. La parte de la URL que se captura por el carácter * está disponible con el término {1}. Si utilizamos varios caracteres de escape, existen tantos parámetros como caracteres de escape ({1}, {2}, …).

Así, podemos reducir considerablemente el código de nuestro ejemplo utilizando los escapes y la invocación dinámica de métodos.

```
<action name="nr_*_*" class="acciones.nr.{1}" method="{1}_{2}">
<result name="input" type="tiles">nr_{1}</result>
<result name="success" type="tiles">nr_{1}</result>
</action>
```

De este modo la petición **http://localhost: 8080//TFCF01/nr_sesionIniciar_mostrar** sera dirigida a la clase acciones.nr.sesionIniciar y ejecutara el método mostrar.

Como podemos comprobar las paginas cuyo acceso requiera de registro empezar por ur_ con lo que de este modo a pesar del direccionamiento dinámico podamos controlar el acceso de distinto tipo de usuarios.

4.3.1.4 Arquitectura de Struts2

El esquema arquitectónico siguiente presenta el funcionamiento del framework basado en la utilización de interceptores y de propiedades de JavaBeans.

- El cliente envía consultas hacia un servicio de la aplicación con los parámetros eventuales.
- Se consulta el archivo de configuración de la aplicación o las anotaciones de clases.
- Los interceptores asociados a la acción se ejecutan y realizan los servicios asociados (conservar los parámetros, administrar las sesiones, guardar los mensajes de error...). El interceptor params asignan los valores presentes en la consulta a la clase de acción asociada por medio de sus descriptores de acceso y ejecutar el método de procesamiento (execute() predeterminado).
- La vista que se mostrará se selecciona de acuerdo con el archivo de configuración struts.xml o la anotación correspondiente.
- La clase de acción transmite los datos necesarios a la vista.
- La vista muestra al cliente los resultados procesados.

4.3.2 MVC en Struts 2

Struts sigue el patrón Modelo-Vista-Controlador. En Struts 2 las responsabilidades de modelo, vista y controlador están implementadas por el action, result, FilterDispatcher y los archivos JSP

4.3.2.1 Controlador

En Struts 2 esta parte esta constituida por el FilterDispacher. Este es el encargado de mapear las peticiones realizadas por el usuario e indicarle que acciones debe realizar el sistema para obtener los datos. En una aplicación web las peticiones HTTP serian tomadas como comandos, el trabajo del controlador es tomar esos comandos e indicarles que accion ha de realizar cada comando.

Tenemos como ejemplo la peticion **ajax_confirmar_entidadComprar** que se encarga de realizar la compra de acciones de una entidad.

```
<action name="ajax_confirmar_entidadComprar" class="acciones.ur.listadoEntidades"
method="listadoEntidades_comprar">
<result name="input">/jsp/popup/formulario_entidadComprar.jsp</result>
<result name="success">/jsp/popup/formulario_resultado.jsp</result>
</action>
```

Tenemos por una parte el comando **ajax_confirmar_entidadComprar** el controlador tiene establecido que para este comando ha de realizar la funcion **listadoEntidades_comprar** que se encuentra en la clase **acciones.ur.listadoEntidades** y dependiendo cual sea el resultado que el modelo le devuelva mostrara una de las opciones que tiene establecidas.

4.3.2.2 Modelo

Una vez el controlador ha dictaminado cual es es la clase y la funcion que corresponden a la petición del usuario. Entra en acción el modelo.

En Struts 2 esta formado por los archivos Action y los subsistemas de consulta. Estos se encargan de realizar las operaciones precisas para mostrar el resultado soliditado.

Continuando con el ejemplo anterior veremos que una vez el controlador llama a la función **listadoEntidades_compra**r de la clase **acciones.ur.listadoEntidades** esta realiza la siguiente accion.

```
public String listadoEntidades_comprar()
      {
/*
 * Obtiene los datos de sesion del usuario
 */

Usuario usuario;
try{ usuario = (Usuario) session.get("usuario");}
catch(Exception e){e.printStackTrace();addActionError("Se ha producido un error de
identificacion en el proceso de compra de saldo");return "error";}

/*
 * LLama al controlador de compraventa para realizar la operacion de compra
 */

int correcto = controlador_compra_venta.comprarEntidad(usuario.getAlias(),
entidad.id_entidad, numeroAcciones);

/*
 * LLama al controlador de usuarios obteniendo los datos del usuario y actualizando la
sesion con los posibles cambios que se hayan podido realizar
 * En este caso seria un aumento del credito del usuario
 */
try{usuario = this.controlador_usuarios.usuarios_seleccionar(usuario.getAlias());}
catch(Exception e) {e.printStackTrace();addActionError("Ha ocurrido un error en el
proceso de consulta de datos del usuario");return "success";}
```

```
//Si el sistema devuelve los datos de usuario indicado actualizaremos la sesion
if(usuario!=null){session.put("usuario",usuario);}
//En caso contrario mostraremos un error indicando que no se han podido consultar los
datos del usuario
else{addActionError("Ha ocurrido un error en el proceso de consulta de datos del
usuario");}


/*
 * Comprueba la salida de la llamada al controlador para indicar si se ha producido algun
error
 */


//Si el usuario ha intentado comprar acciones por dinero superior al que posee se le
indicara con un mensaje de aviso y podra volver a introducir la cantidad de acciones que
desea comprar
//Ademas de los errores que el usuario pueda realizar al solicitar la operacion, tambien
se contempla que las acciones hayan subido de precio, y ya no disponga del dinero
suficiente para comprarlas.
if(correcto==2){addActionError("No dispone del dinero suficiente para realizar la
operación");detalle_de_entidad();return "input";}

//Si la operacion se realiza correctamente se indica con un mensaje
else if(correcto==1){addActionMessage("Se ha eliminado la tarjeta correctamente");return
"success";}

//Si se produce algun otro error, se mostrara un mensaje indicandolo
else{addActionError("No se ha podido realizar la operacion de compra de acciones");return
"success";}
```

Como podemos ver esta función junto con la ayuda del controlador de usuarios y compra_venta realizar las operaciones necesarias para obtener los datos solicitados.

Podemos ver también que dependiendo de unas determinadas circunstancias queramos que se muestre un resultado u otro. Para ello devolveremos al controlador una respuesta indicándole cual fue el estado de la peticion.

En este caso tenemos **input** y **success** que corresponden a si el usuario tenia dinero suficiente para realizar la operación o no.

Como podemos ver el resultado que envia al modelo al controlador sera redirigido a la vista que hayamos establecido en el controlador.

```
<action name="ajax_confirmar_entidadComprar" class="acciones.ur.listadoEntidades"
method="listadoEntidades_comprar">
<result name="input">/jsp/popup/formulario_entidadComprar.jsp</result>
<result name="success">/jsp/popup/formulario_resultado.jsp</result>
</action>
```

4.3.2.3 Vista

La vista es el componente de presentación, la interfaz de usuario, es el resultado que podemos ver en el navegador.

Esta parte esta formada por los JSP que son archivos que contienen el código HTML y algunas etiquetas de Struts 2 que se encargan de mostrar los datos indicados. Desde la vista es también donde el usuario realiza las peticiones HTTP que el controlador debe manejar.

En el ejemplo que estamos tratando el usuario realizaba una petición de compra de acciones.

Como podemos ver todo empezaría desde este formulario **formulario_entidadComprar**.

```
<%@ taglib prefix="s" uri="/struts-tags"%>
<script>$(document).ready(entidadComprar());</script>

<div class="noticiaTitulo">Comprar acciones</div>
<p>Introduzca la cantidad de acciones que desea comprar de la entidad <s:property
value='entidad.nombre'/>. El precio actual por acci&oacute;nes de <s:property
value='entidad.ultima'/> &euro;.</p>
<div class="margen5"></div>
<div class="margen20"></div>

<!-- Error -->
<s:if test="errors.size()>0"><div id="mensaje_error"><label>Se han producido los
siguientes errores: </label><ul><s:fielderror /></ul></div><div
class="margen30"></div></s:if>
<s:if test="errorMessages.size()>0"><div id="mensaje_error"><ul><s:actionerror
/></ul></div><div class="margen5"></div><div class="margen20"></div></s:if>
<!-- Fin Error -->
<s:form method="post" action="ajax_confirmar_entidadComprar.action" id="Formulario_ajax"
name="Formulario_ajax">
	<input type="hidden" name="entidad.id_entidad" value="<s:property
value='entidad.id_entidad'/>">
	<div class="datosDeRegistroPopup">N&uacute;mero de acciones:<input
class="formularioCampoPopup" name="numeroAcciones" id="numeroAcciones" type="text"></div>
	<div class="margen20"></div>
	<div class="margen20"></div>
	<div style="text-align: right">
		<input name="entidadComprar" value="Comprar" type="submit">
		<input onclick="JavaScript:cerrarPopup()" name="Cancelar" value="Cancelar"
type="reset">
	</div>
</s:form>
```

Como podemos ver el usuario establece el numero de acciones que desea comprar y al hacer clic en el botón comprar enviá la petición HTTP al servidor indicándole la acción que desea realizar. En este caso la acción seria **ajax_confirmar_entidadComprar.**

Una vez el servidor ejecute las operaciones precisas nos sera devuelto un resultado para obtener el resultado de la operación solicitada.

Si el resultado de la operación fuese **input** el servidor nos volvería a mostrar el formulario anterior, y nos indicaría que no disponemos de suficiente dinero para realizar la operación.

En el caso contrario el controlador ser encargaría de devolvernos la vista **formulario_resultado.**

```
<%@ taglib prefix="s" uri="/struts-tags"%>

<div class="noticiaTitulo">
<s:if test="errors.size()>0">La operac&oacute;n no ha podido realizarse</s:if>
<s:elseif test="actionMessages.size()>0">La operac&oacute;n ha sido realizada con
exito</s:elseif>
<s:else></s:else>
</div>
<p>
<s:if test="errors.size()>0"><s:actionerror /></s:if>
<s:elseif test="actionMessages.size()>0"><s:actionmessage/></s:elseif>
<s:else></s:else>
</p><div class="margen5"></div><div class="margen20"></div>
```

4.4 Hibernate * Información obtenida de " Wikipedia".

Es una herramienta de Mapeo objeto-relacional (ORM) para la plataforma Java que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML) o anotaciones en los beans de las entidades que permiten establecer estas relaciones.

Hibernate busca solucionar el problema de la diferencia entre los dos modelos de datos coexistentes en una aplicación: el usado en la memoria de la computadora (orientación a objetos) y el usado en las bases de datos (modelo relacional). Para lograr esto permite al desarrollador detallar cómo es su modelo de datos, qué relaciones existen y qué forma tienen. Con esta información Hibernate le permite a la aplicación manipular los datos de la base operando sobre objetos, con todas las características de la POO. Hibernate convertirá los datos entre los tipos utilizados por Java y los definidos por SQL. Hibernate genera las sentencias SQL y libera al desarrollador del manejo manual de los datos que resultan de la ejecución de dichas sentencias, manteniendo la portabilidad entre todos los motores de bases de datos con un ligero incremento en el tiempo de ejecución.

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

4.4.1 Mapeo de clases

Para que hibernate como implementar el modelo orientado a objetos con la persistencia de datos debemos indicarle que entidades debe manejar.

Estas entidades indican a hibernate con que tabla de la base de datos han de operar, que atributos corresponden a cada campo y que tipo de dato han de utilizar.

Tenemos por ejemplo la entidad usuario.

```
@Entity
@Table(name="usuarios")
public class Usuario implements Serializable{
	private static final long serialVersionUID = -8486593002740655790L;
	private String alias;
	private String nombre;
	private String apellido_primero;
	private String apellido_segundo;
	private String email;
	private Date fecha_de_registro;
	private Date fecha_de_nacimiento;
	private float credito;
	private float credito_comprado;
	private String pass;

	@Id
	@Column(name="alias")
	public String getAlias() {return alias;}
	@Column(name="nombre")
	public String getNombre() {return nombre;}
	@Column(name="apellido_primero")
	public String getApellido_primero() {return apellido_primero;}
	@Column(name="apellido_segundo")
	public String getApellido_segundo() {return apellido_segundo;}
	@Column(name="email")
	public String getEmail() {return email;}
	@Column(name="fecha_de_registro")
```

```
public Date getFecha_de_registro() {return fecha_de_registro;}
@Column(name="fecha_de_nacimiento")
public Date getFecha_de_nacimiento() {return fecha_de_nacimiento;}
@Column(name="credito")
public float getCredito() {return credito;}
@Column(name="credito_comprado")
public float getCredito_comprado() {return credito_comprado;}
@Column(name="pass")
public String getPass() {return pass;}

public void setAlias(String alias) {this.alias = alias;}
public void setNombre(String nombre) {this.nombre = nombre;}
public void setApellido_primero(String apellido_primero) {
this.apellido_primero = apellido_primero;}
public void setApellido_segundo(String apellido_segundo) {
this.apellido_segundo = apellido_segundo;}
public void setEmail(String email) {this.email = email;}
public void setFecha_de_registro(Date date) {this.fecha_de_registro = date;}
public void setFecha_de_nacimiento(Date fecha_de_nacimiento) {
this.fecha_de_nacimiento = fecha_de_nacimiento;     }
public void setCredito(float credito) {this.credito = credito;}
public void setCredito_comprado(float credito_comprado) {
this.credito_comprado = credito_comprado;     }
public void setPass(String pass) {this.pass = pass;}}
```

4.4.2 HibernateUtil.java

Este archivo es el encargado dre inicar el entorno hibernate y obtener un objeto Session utilizando la clase SessionFactory de Hibernate.

```
public class HibernateUtil {
private static final SessionFactory sessionFactory;

static {
try {sessionFactory= new AnnotationConfiguration().configure().buildSessionFactory();}
catch (Throwable ex) {System.err.println("Initial SessionFactory creation failed." + ex);
throw new ExceptionInInitializerError(ex);}
}
public static SessionFactory getSessionFactory() {return sessionFactory;}}
```

La llamada a Configuration().configure() carga el fichero de configuración hibernate.cfg.xml e inicializa el entorno de Hibernate.
El objeto sessionFactory solo se crea una vez, y se reutiliza las veces necesarias. Este funcionamiento es el designado por el patrón singleton

4.4.3 El patrón de diseño singleton * Información obtenida de “ Wikipedia”.

Singleton (instancia única) está diseñado para restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto.
Su intención consiste en garantizar que una clase sólo tenga una instancia y proporcionar un punto de acceso global a ella.
El patrón singleton se implementa creando en nuestra clase un método que crea una instancia del objeto sólo si todavía no existe alguna. Para asegurar que la clase no puede ser instanciada nuevamente se regula el alcance del constructor (con atributos como protegido o privado).
La instrumentación del patrón puede ser delicada en programas con múltiples hilos de ejecución. Si dos hilos de ejecución intentan crear la instancia al mismo tiempo y esta no existe todavía, sólo uno de ellos debe lograr crear el objeto. La solución clásica para este problema es utilizar exclusión mutua en el método de creación de la clase que implementa el patrón.
Las situaciones más habituales de aplicación de este patrón son aquellas en las que dicha clase

controla el acceso a un recurso físico único (como puede ser el ratón o un archivo abierto en modo exclusivo) o cuando cierto tipo de datos debe estar disponible para todos los demás objetos de la aplicación.

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

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

5 Implementación

5.1 Instalación de la base de datos.

Para la instalación de la base de datos MySQL utilizaremos el programa XAMPP que automatiza todo el proceso y ademas nos instala un servidor apache desde el que podremos consultar la base de datos si nos fuese necesario. Este programa lo podremos descargar desde http://www.apachefriends.org/en/xampp.html

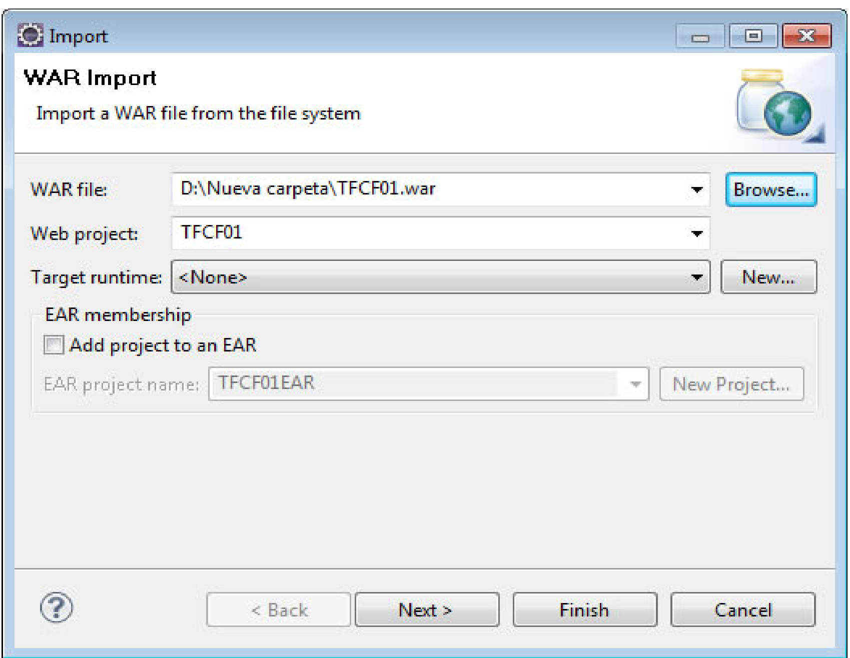
Lo mas importante en este paso es que la contraseña de **root** sea **roberto**, ya que es la que usaremos en el proyecto.

Posteriormente debemos crear la base de datos **ubrokerc** y ejecutar el archivo SQL que se adjunta en los documentos enviados, para crear las tablar y datos que nos serán necesarias

5.2.1 Instalación del archivo WAR en el servidor GlassFish.

5.2.1 Importar proyecto

En primer lugar debemos importar el archivo WAR que se adjunta en los documentos enviados.



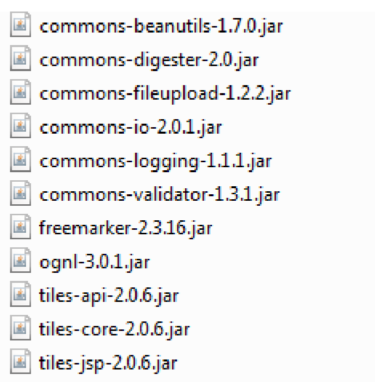
Con esto tendremos los archivos del programa. Como se nos ha indicado que no se incluyan las bibliotecas deberemos instalarlas manualmente. También debemos eliminar el archivo index.jsp que se crea automáticamente al importar un archivo.

5.2.2 Instalacion del frameworks Struts2

Para la instalación del framework Struts2 debemos realizar los siguientes pasos.

En primer lugar instalaremos el framework Struts 2 el cual podremos descargar de esta página. http://struts.apache.org/download.cgi#struts231-SNAPSHOT

Una vez descomprimido, pasaremos a importar las librerías a nuestro proyecto. Pare ello es necesario copiar los archivos JAR indicados en la siguiente imagen en WebContent -> WEB-INF -carpeta lib>. Crear esta carpeta si no existe.



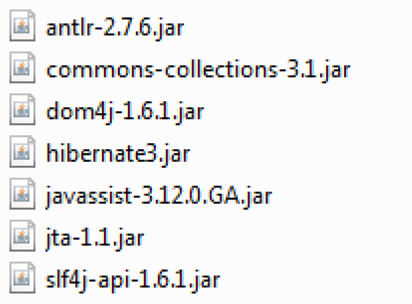
5.2.3 Cartografía Struts2 en web.xml (Paso ya configurado en el archivo WAR)

El punto de entrada de la solicitud Struts2 será el filtro definido en el descriptor de despliegue web.xml. Por lo tanto vamos a definir una entrada deorg.apache.struts2.dispatcher.FilterDispatcher clase en web.xml. Abrir el archivo web.xml que se encuentra bajo la carpeta WEB-INF y copiar y pegar el siguiente código.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="ejemplo35" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-
app_2_4.xsd">

<listener>
<listener-class>
org.apache.struts2.tiles.StrutsTilesListener
</listener-class>
</listener>
<filter>
<filter-name>struts2</filter-name>
<filter-class> org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter
</filter-class>
</filter>
<filter-mapping>
<filter-name>struts2</filter-name>
<url-pattern>/*</url-pattern>
</filter-mapping></web-app>
```

5.2.4 Instalación del framework Hibernate

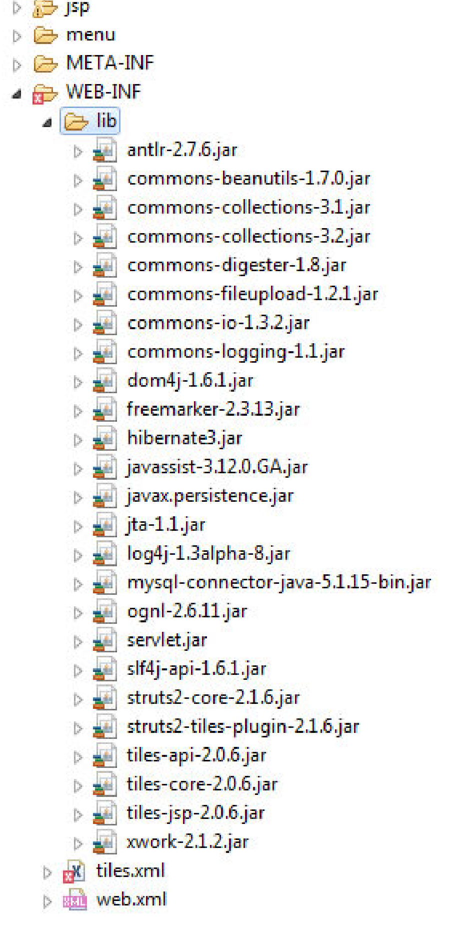
Para añadir el soporte de hibernate, debemos importar las bibliotecas indicadas en la siguiente imagen, las cuales podemos obtener en http://sourceforge.net/projects/hibernate/files/hibernate3/



Debemos añadir también el conector a la base de datos que vayamos a utilizar ya que es el encargado de realizar las peticiones realizadas por hibernate a la base de datos.

En nuestro caso como la base da datos es mysql nos descargaremos el conector desde http://www.mysql.com/products/connector/

Una vez hayamos descomprimido los archivos debemos importarlos a nuestro proyecto quedando así.



5.2.5 Configuración hibernate.cfg.xml (Paso ya configurado en el archivo WAR)

Después añadiremos el siguiente archivo a nuestro proyecto **hibernate.cfg.xml**
Este es el archivo de configuración de Hibernate y contendrá configuraciones tales como información de conexión de base de datos, la persistencia de la clase de información.

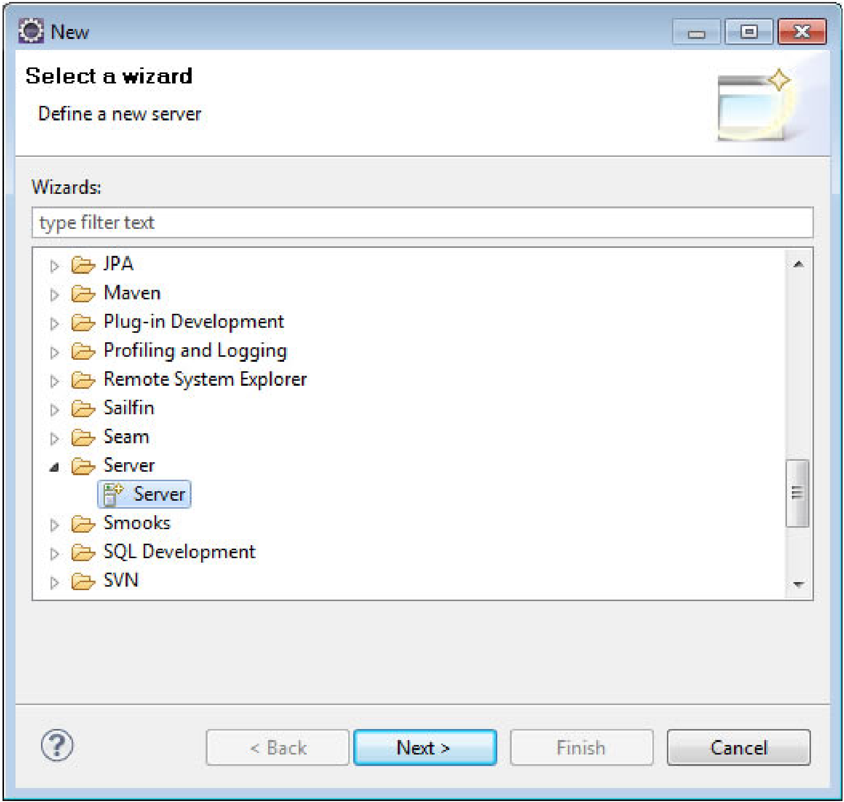
```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session-factory name="">
<property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
<property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/ubrokerc</property>
<property name="hibernate.connection.username">root</property>
<property name="hibernate.connection.password">roberto</property>
<!-- <property name="hibernate.hbm2ddl.auto">create</property> -->

<!-- Mezcla -->
<property name="current_session_context_class">thread</property>
<property name="cache.provider_class">org.hibernate.cache.NoCacheProvider</property>
<property name="show_sql">true</property>
<property name="hbm2ddl.auto">update</property>
<!-- Fin Mezcla -->

<mapping class="entidades.Usuario" />
<mapping class="entidades.Entidad" />
<mapping class="entidades.Mercado" />
<mapping class="entidades.Tarjeta_de_credito" />
<mapping class="entidades.Operacion_cartera" />
</session-factory>
</hibernate-configuration>
```

5.2.6 Instalación del servidor Glassfish en eclipse

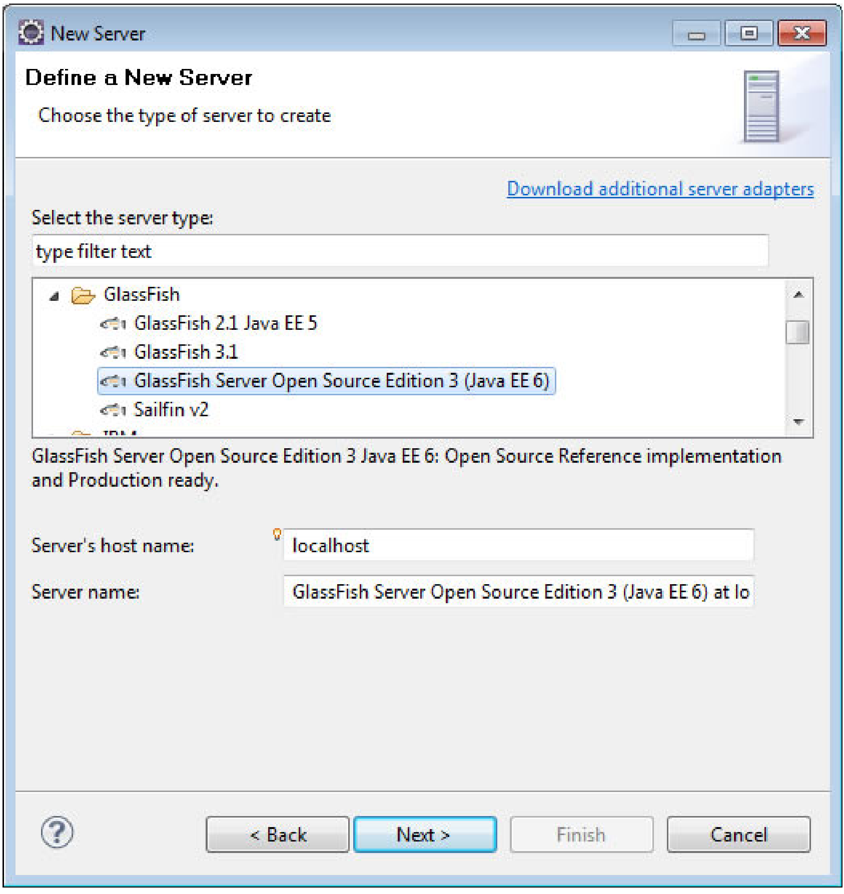
Para la realización de pruebas en el periodo de implementación instalaremos el servidor Glassfish de manera que sea ejecutado por el eclipse. Para lo cual seguiremos los siguientes pasos.

En primero lugar debemos descargarnos el servidor GlassFish http://download.oracle.com/otn-pub/java/java_ee_sdk/6u2-wjdk-6u26/java_ee_sdk-6u2-jdk-windows-ml.exe desde http://www.oracle.com/technetwork/java/javase/downloads/java-ee-6u2-wjdk-6u26-409231.html

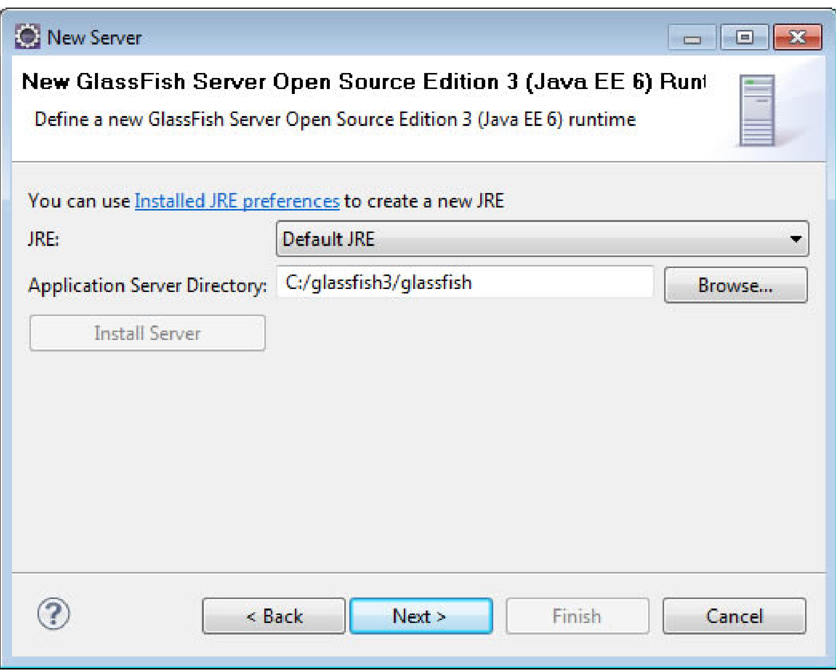
Una vez hayamos descargado y descomprimido el archivo pasaremos a la implementación en Eclipse.



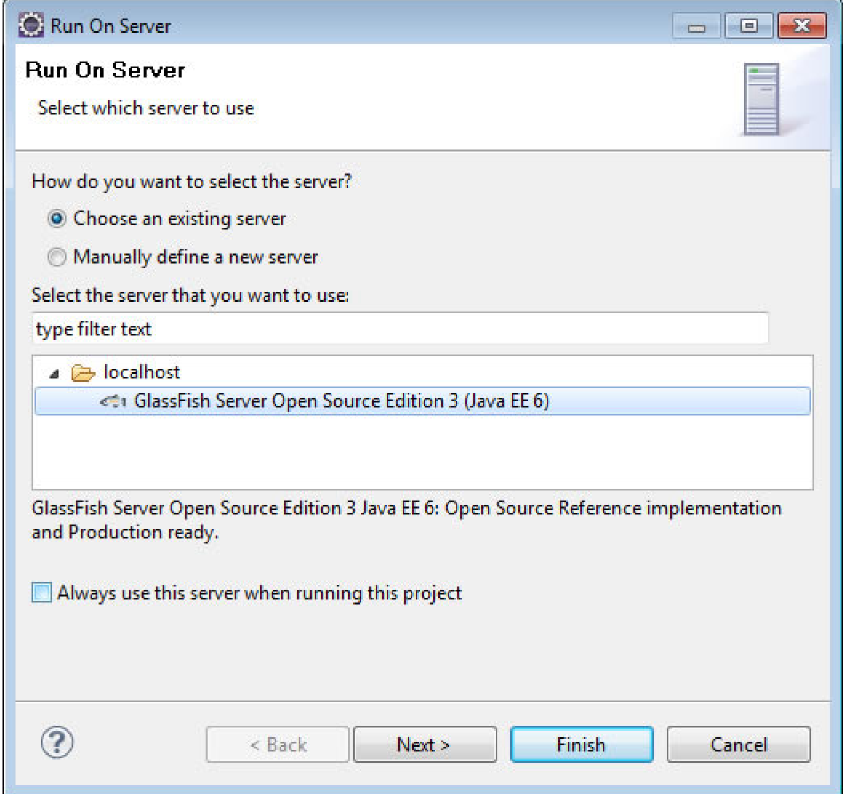
Debemos indicar que servidor deseamos utilizar.



Una vez seleccionado el servidor debemos indicar donde hemos descomprimido el archivo, para que el programa pueda llamarlo cuando necesite ejecutarlo.



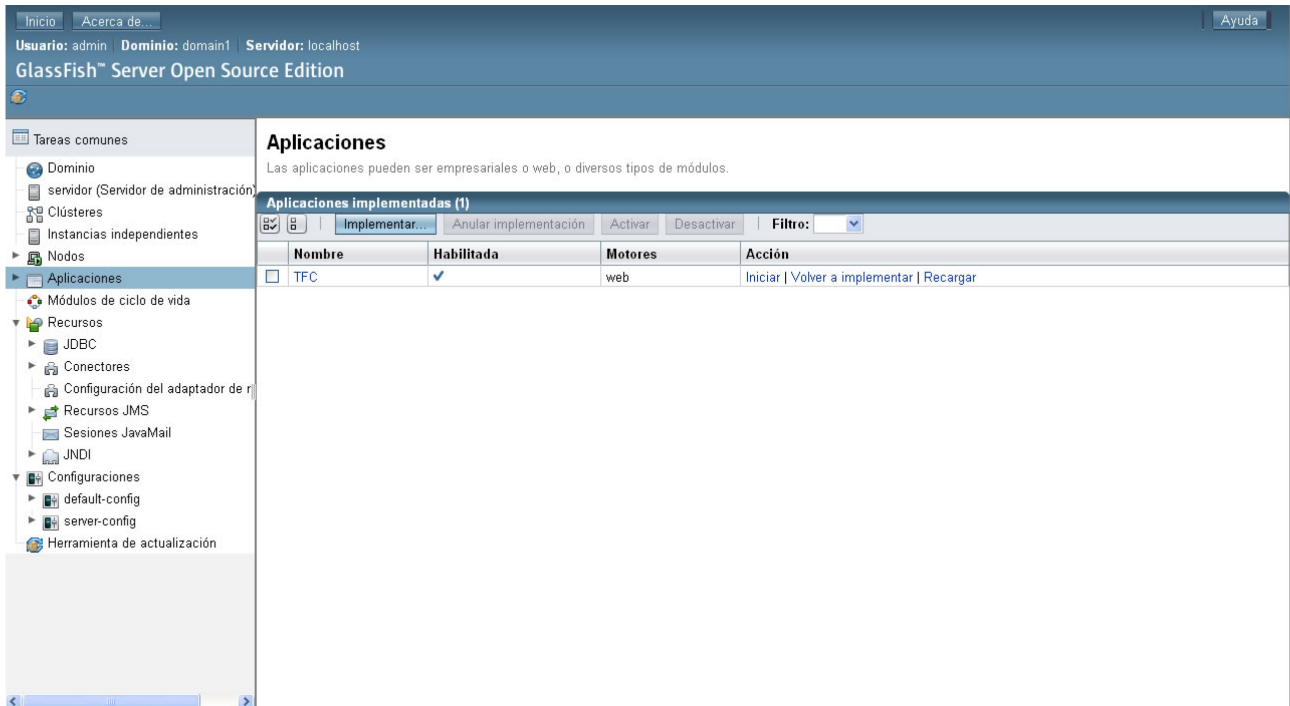
Ahora para ejecutar el proyecto solo debemos de ejecutar la opción **run on server**



5.3 Instalación alternativa

Utilizando el archivo war de nuestro proyecto que contiene todas las bibliotecas o bien el war en el que no vienen incluidas una vez las hayamos añadido en eclipse y tomando por defecto que hemos instalado el servidor Glassfish indicado en **C:\glassfish3** ejecutaremos el comando **C:\glassfish3\bin\asadmin.bat start-domain domain1**

Después introducimos http://localhost:4848/ en nuestro el navegador y vamos al menú de administración de Glassfish desde ese menú implementaremos nuestro archivo war. Es importante tener en cuenta el nombre del proyecto para saber que dirección debemos poner en el navegador.



5.4 Actualizador de datos (Muy importante)

El programa que mantiene actualizado el listado de valores puede ejecutarse desde el archivo **actualizador.jar** que contiene el archivo proyecto. Hay que tener en cuenta que este programa realiza peticiones a una web externa con lo que es necesario que disponga de conexión a Internet. Ademas tiene una frecuencia de actualización de 2 minutos con lo que tardara este tiempo en realizar la primera actualización. A partir de ese momento las sucesivas actualizaciones llevaran esa frecuencia. **Es importante ejecutar este programa primero y mantenerlo siempre corriendo, para que los datos estén actualizados.**

Por motivos de comodidad para la corrección he puesto que el periodo apertura y cierre del IBEX 35 comprenda todo el día. En un correcto funcionamiento se indicarían las horas de apertura y cierre correctas, de manera que no realice peticiones innecesarias.

5.5 Visualización del proyecto

Una vez tengamos instalado el proyecto podremos verlo introduciendo la dirección

http://localhost:8080/TFC/ (o el nombre que le hayamos indicado al proyecto)

5.6 Test del proyecto – Pruebas unitarias

Para verificar el funcionamiento del proyecto realice los siguientes pasos:

- Cree el usuario **roberto** con la contraseña de acceso **123123**.
- Añadí un tres tarjetas de crédito, elimine una, Después compre crédito con una de las otras dos.
- Compre acciones de varias entidades, después vendí acciones de una.
- Intente comprar acciones por valor de un precio superior a mi crédito para verificar que no me lo permitía.
- Intente vender un numero de acciones superior al que tenia, para verificar que no me lo permitía.
- Para verificar que mis beneficios variaban visualice mi cartera de acciones entre las 9 y las 3 del mediodía y teniendo el actualizador ejecutándose.

6 Bibliografía

- **STRUTS 2 El framework de desarrollo de aplicaciones Java EE de Lafosse, Jerome**
- **Manual Hibernate:** http://www.courses.coreservlets.com/Course-Materials/pdf/hibernate/
- **Manual Hibernate:** http://docs.jboss.org/hibernate/core/3.6/reference/es-ES/html/
- **Manual:** http://www.srikanthtechnologies.com/blog/java/struts2hib.aspx
- **Manual:** http://programacionconejemplos.blogspot.com/2009/06/2-segundo-ejemplo-de-struts2-manejo-de.html

7 Conclusiones

En mi opinión la experiencia del TFC ha sido muy positiva. Llegando a adquirir conocimientos bastante altos acerca de Struts2 e Hibernate. Ademas del funcionamiento del patrón MVC en entornos J2EE.

He valorado sobre todo la parte de documentación y planificación la cual considero vital a la hora de llevar a buen puerto un proyecto ya que te centra en unos objetivos y plazos concretos.
A medida que se realiza la implementación siempre surgen nuevas ideas que mejorarían mucho el programa, pero si se quieren cumplir los plazos es muy importante seguir el guión inicial y dejar esas ideas para futuras actualizaciones.

También quiero remarcar que el aprendizaje de utilización de los diversos Frameworks es mucho mas sencillo de lo que en un principio esperaba, ya que en apenas mes y medio he aprendido lo suficiente para poder emprender casi cualquier proyecto.