

UNIVERSIDAD DE ALCALÁ



Escuela Politécnica Superior

Grado en Ingeniería Informática

Trabajo Fin de Grado

**ESTUDIO DEL FRAMEWORK DE DESARROLLO WEB:
STRUTS2**

Jorge Lillo Cobacho

Julio / 2013

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

GRADO EN INGENIERÍA INFORMÁTICA

Trabajo Fin de Carrera

ESTUDIO DEL FRAMEWORK STRUTS2

Autor: Jorge Lillo Cobacho

Director: Salvador Otón Tortosa

Tribunal:

Presidente: _____

Vocal 1º: _____

Vocal 2º: _____

Calificación: _____

Alcalá de Henares a de de 2013

Agradecimientos

A mis padres por todo lo que han hecho por mí y lo que me han enseñado.

A mi hermana, por ser un modelo a seguir.

A mis amigos de toda la vida por estar siempre ahí cuando los necesito.

A mi abuelo Alfonso, por enseñarme a luchar por lo que quiero y de la forma que quiero.

A mis compañeros de facultad por estos cuatro años juntos.

A todos ellos GRACIAS.

ÍNDICE RESUMIDO

1. INTRODUCCIÓN	6
2. OBJETIVOS DEL PROYECTO	8
3. DESARROLLO DEL ESTUDIO	9
4. RESUMEN Y CONCLUSIÓN.....	69
5. BIBLIOGRAFÍA	73
6. APÉNDICES	74

ÍNDICE DETALLADO

1. INTRODUCCIÓN	6
2. OBJETIVOS DEL PROYECTO	8
3. DESARROLLO DEL ESTUDIO	9
3.1. FRAMEWORK WEB	9
3.2. PATRÓN MODELO-VISTA-CONTROLADOR (MVC)	11
3.3. ¿QUÉ ES STRUTS2?	13
3.4. CARACTERÍSTICAS STRUTS2	14
3.5. CONFIGURACIÓN Y USO BÁSICO DE STRUTS2	15
(a) Registro del Filtro Controlador en el web.xml	16
(b) Creación de un Componente Action	16
(c) Creación del archivo validation.xml	18
(d) Registro del Action en el archivo struts.xml	18
(e) Creación de las Vistas: JSPs	19
3.6. CICLO DE VIDA	21
3.7. COMPONENTES STRUTS2	22
(a) FilterDispatcher	22
(b) Actions	23
(c) Results	26
(d) Interceptors	35
(e) OGNL	39
(f) Scopes	41
(g) Etiquetas	45
(h) Formularios	67
4. CONCLUSIONES	69
4.1. PRESUPUESTO	69
(a) Costes Personal	69
(b) Costes Material	71
(c) Presupuesto Final	71
4.2. CONCLUSIÓN Y LÍNEAS FUTURAS	71
5. BIBLIOGRAFÍA	73
6. APÉNDICES	74
6.1. ANEXO A: MANUAL DE USUARIO SUPERMARKET	74
6.2. ANEXO 2: PREPARACIÓN PARA EL USO DE STRUTS2 CON NETBEANS 7.2	95

ÍNDICE DE FIGURAS

ILUSTRACIÓN 1: EXPLICACIÓN MODELO-VISTA-CONTROLADOR.....	11
ILUSTRACIÓN 2: JERARQUÍA DE UN PROYECTO CON STRUTS2	15
ILUSTRACIÓN 3: EJEMPLO BÁSICO STRUTS2.....	20
ILUSTRACIÓN 4: CICLO DE VIDA STRUTS2.....	21
ILUSTRACIÓN 5: EJEMPLO WEB.XML FILTERDISPATCHER	23
ILUSTRACIÓN 6: EJEMPLO WEB.XML STRUTS2PREPAREANDEXECUTEFILTER	24
ILUSTRACIÓN 7: EJEMPLO MAPEO ACTION.....	24
ILUSTRACIÓN 8: EJEMPLO ACTION, CLASE JAVA.....	25
ILUSTRACIÓN 9: EJEMPLO ACTION, CLASE JAVA CON ANOTACIONES.....	26
ILUSTRACIÓN 10: RESULT TYPE DISPATCHER	28
ILUSTRACIÓN 11: RESULT TYPE DISPATCHER SIMPLIFICADO	28
ILUSTRACIÓN 12: RESULT TYPE REDIRECT	29
ILUSTRACIÓN 13: RESULT TYPE CHAIN.....	30
ILUSTRACIÓN 14: RESULT TYPE HTTPHEADER	32
ILUSTRACIÓN 15: INTERCEPTOR STRUTS.XML	37
ILUSTRACIÓN 16: EJEMPLO INTERCEPTOR LOGGER	38
ILUSTRACIÓN 17: OUTPUT INTERCEPTOR LOGGER.....	38
ILUSTRACIÓN 18: EJEMPLO INTERCEPTOR TIMER	38
ILUSTRACIÓN 19: OUTPUT INTERCEPTOR TIMER.....	38
ILUSTRACIÓN 20: INTERCEPTOR GLOBAL EN STRUTS.XML.....	39
ILUSTRACIÓN 21: OBJETOS ACTIONCONTEXT STRUTS.....	40
ILUSTRACIÓN 22: CLASE IMPLEMENTANDO SESSIONAWARE Y REQUESTAWARE	42
ILUSTRACIÓN 23: USO DE SCOPE SESSION	42
ILUSTRACIÓN 24: USO DE SCOPE REQUEST	42
ILUSTRACIÓN 25: SCOPE ACTIONCONTEXT.....	43
ILUSTRACIÓN 26: ETIQUETA COMBOBOX RESULTADO.....	53
ILUSTRACIÓN 27: ETIQUETA DOUBLESELECT RESULTADO	58
ILUSTRACIÓN 28: EJEMPLO FORMULARIO CON STRUTS2	67
ILUSTRACIÓN 29: FORMULARIO VISTO DESDE LA VISTA.....	68
ILUSTRACIÓN 30: ACTION QUE RECIBE LOS DATOS DEL FORMULARIO.....	68

Resumen

Este proyecto consiste en el estudio del framework de desarrollo web Struts2, partiendo de conceptos básicos como qué es un framework o en qué consiste el patrón modelo-vista-controlador, entrando en los detalles del funcionamiento del mismo.

Se abarcarán los distintos componentes que forman Struts2 y se detallarán ejemplos con el fin de comprender cómo trabaja el framework, su utilidad a la hora de realizar desarrollo web y por qué la mayoría de las empresas actuales lo utilizan en sus proyectos.

Además, para asimilar los diferentes conceptos expuestos, se adjunta la aplicación "SuperMarket" que demostrará la potencia y de lo que es capaz Struts2.

Summary

This project lies in the study of the web framework Struts2, starting from basic notions like what a framework is, or what the Model-View-Controller pattern consists of. Besides, it explains in detail how Struts2's components work.

It encompasses different components that define Struts2. On the other hand, some examples will be detailed to understand its working, as it can be quite useful when the goal is to perform web development and explain why most companies use it in their projects nowadays.

In addition and in order to understand these terms, the application "SuperMarket" is attached, so that the power and capacity of Struts2 can be demonstrated.

Palabras clave:

Desarrollo web, Framework (Marco de trabajo), Struts2, Patrón Modelo-Vista-Controlador, Java EE (Java Empresarial)

Keywords:

Web development, Framework, Struts2, Model-view-controller pattern, Java EE (Enterprise Edition)

1. INTRODUCCIÓN

¿Qué es una aplicación web? Una aplicación web es un conjunto de componentes web que se coordinan y actúan entre ellos para realizar tareas concretas. Además, ofrece servicios completos al usuario.

Ideas de aplicación web:

- Unificar acceso a recursos (único directorio jerárquico)
- Url común
- Control unificado: descriptor web.xml común que controla el comportamiento de la web

Servidor de aplicaciones web: es el lugar donde residen las aplicaciones web. Facilita la gestión de acceso a los recursos del sistema, gestor de conexión con bases de datos, contenedores de recursos (JSP, servlets...). Todo ello, controlado por descriptores xml.

Formas de realizar desarrollo web dinámico:

- Los servlets son aplicaciones java que crean contenido HTML a base de sentencias “out.print”. Sin embargo, es tedioso y complejo crear y mantener páginas con contenido HTML.
- JSP es la otra alternativa java a la generación de contenidos de forma dinámica en el lado del servidor. El código Java queda embebido dentro del código HTML de forma similar a PHP o ASP, separa el código java del HTML. Es más conveniente que los servlets para generar contenido HTML.

JSP es, en el fondo, una tecnología equivalente a los servlets, dado que las páginas JSP se traducen en servlets que ejecuta el servidor en cada petición. JSP permite la escritura de código Java dentro de ellos, consiguiendo todas las ventajas, tanto de los servlets como del código HTML.

- La construcción de un jsp hace más sencillo el desarrollo y mantenimiento que un HTML.
 - o Llamar al servlet cuando hagamos scripting
 - o Usar beans y custom tag para páginas más complejas
- Pero esto no es suficiente, ya que el procesamiento de JSP complejos es incómodo.
- Separando el código real en servlets, tags y clases, el JSP tiene un aspecto de página única.

Para facilitar todas estas tareas existen los framework de desarrollo web. Los objetivos principales que persigue un framework son: acelerar el proceso de desarrollo, reutilizar código ya existente y promover buenas prácticas de desarrollo como el uso de patrones.

Un framework web, por tanto, podemos definirlo como un conjunto de componentes (por ejemplo, clases en java y descriptores y archivos de configuración en XML) que componen un diseño reutilizable que facilita y agiliza el desarrollo de sistemas web.

Existen varios tipos de frameworks web: orientados a la interfaz de usuario, como Java Server Faces; orientados a aplicaciones de publicación de documentos, como Coocon; orientados a la parte de control de eventos, como Struts y algunos que incluyen varios elementos como Tapestry.

La mayoría de frameworks web se encargan de ofrecer una capa de controladores de acuerdo con el patrón MVC o con el modelo 2 de servlets y JSP, ofreciendo mecanismos para facilitar la integración con otras herramientas para la implementación de las capas de negocio y presentación.

Struts es un framework de desarrollo web basado en MVC (modelo vista controlador), que permite desarrollar servlet y JSP mediante MVC.

Struts permite reducir el tiempo de desarrollo. Su carácter de software libre y su compatibilidad con todas las plataformas en las que Java Enterprise esté disponible, lo convierten en una herramienta altamente utilizable. Con la versión 2 del framework, se introdujeron algunas mejoras sobre la primera para simplificar las tareas más comunes en el desarrollo de aplicaciones web, así como mejorar su integración con AJAX, etc.

Struts ofrece su propio componente controlador y proporciona integración con otras herramientas para implementar el modelo, mediante tecnologías de acceso a datos como JDBC o Hibernate, y la vista, mediante JSP, Velocity o XSLT.

Struts ofrece un sistema de tuberías que permite la comunicación entre el modelo que contiene los datos, y las vistas que ofrecen estos datos a los usuarios y reciben sus órdenes.

Este es el punto de partida del siguiente trabajo en donde se abarcarán las diferentes funciones y características de Struts2 a lo largo de la memoria.

2. OBJETIVOS DEL PROYECTO

El objetivo del proyecto es conocer en profundidad una de las técnicas más usadas en la actualidad en el mundo del desarrollo de aplicaciones web: el uso del framework de desarrollo web Struts2.

Está destinado a gente con todo tipo de conocimientos, no solo a gente adentrada en el área del desarrollo web, puesto que alguien que quiera empezar a conocer este mundo podrá comprender para qué se usa un framework, o qué es el patrón modelo-vista-controlador, el cual se puede ver actualmente en cualquier proyecto: desarrollo web, desarrollo de aplicaciones móviles...

No obstante, el objetivo principal del proyecto consiste en el uso de Struts2: se expondrá qué es y para qué sirve, qué ventajas puede ofrecer el uso de este framework frente a programar aplicaciones web sin él.

En un primer acercamiento y para poder comprender a qué nos enfrentamos, se detallarán los pasos que se tienen que seguir para poder usar struts2 en una aplicación web básica. A su vez, se ampliarán los conceptos básicos explicados con anterioridad y para enterarse de qué va pasando internamente en el programa ejemplo, se explicará el ciclo de vida que sigue el framework Struts2, qué acciones se llevan a cabo en cada paso y cómo llegamos al resultado final.

Tras entender qué es, cómo funciona y qué hace en cada paso Struts2, nos adentraremos en los distintos componentes del framework. Además, se expondrá la función de cada componente y se mostrará qué opciones nos da Struts2 para resolver distintos casos que se nos plantean a la hora de programar una aplicación web, por ejemplo cómo poder, desde una misma clase, redireccionar a una vista distinta dependiendo del resultado (Results) o cuando queremos evitar que se lleve a cabo una acción (Interceptor)...

Finalmente y una vez vistos todos los componentes de Struts2, se comprenderá que usar Struts2 facilita la vida al programador web, ayuda a poder avanzar con mayor rapidez y, a su vez, al usar el patrón modelo vista controlador se dividen las aplicaciones, de tal forma que, al dividirnos todo en vistas, controladores o modelos, la dificultad de mantenimiento de los programas se reduce considerablemente.

Para poder ver todo lo explicado en este documento, se adjunta el programa “SuperMarket”, programa que simula la web de un supermercado online para poder realizar compras. SuperMarket usa los distintos componentes de Struts2 y la mayoría de los ejemplos de código vistos en el apartado de componentes de esta memoria están extraídos de él.

Todo ello, servirá para ver un ejemplo no solo teórico, sino también práctico de lo que es capaz de hacer el framework Struts2, así como todas sus ventajas y características (desarrolladas en esta memoria) que lo hacen, a día de hoy, uno de los frameworks más usados y valorados en el mundo del desarrollo web.

3. DESARROLLO DEL ESTUDIO

3.1. Framework Web

En el desarrollo de software, un framework es una estructura de soporte definida en la cual un proyecto de software puede ser organizado y desarrollado. Típicamente, un framework puede incluir soporte de programas, bibliotecas y un lenguaje de scripting para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

Por framework nos estamos refiriendo a una estructura de software compuesta de componentes personalizables e intercambiables para el desarrollo de una aplicación. En otras palabras, un framework se puede considerar como “una aplicación genérica incompleta y configurable a la que podemos añadirle las últimas piezas para construir una aplicación concreta”.

Se trata de extensiones de un lenguaje que, a través de una o más jerarquías de clases implementan una funcionalidad y pueden ser extendidas de manera opcional.

Características básicas de los frameworks web:

- Abstracción de URLs y sesiones: no es necesario manipular directamente las URLs ni las sesiones, el framework ya se encarga de hacerlo.
- Acceso a datos: incluyen las herramientas e interfaces necesarias para integrarse con herramientas de acceso a datos, en BBDD, XML, etc...
- Controladores: la mayoría de frameworks implementan una serie de controladores para gestionar eventos, como una introducción de datos mediante un formulario o el acceso a una página. Estos controladores suelen ser fácilmente adaptables a las necesidades de un proyecto concreto.
- Autenticación y control de acceso: incluyen mecanismos para la identificación de usuarios mediante login y password, y permiten restringir el acceso a determinadas páginas a determinados usuarios.
- Internacionalización
- Separación entre diseño y contenido.

El objetivo de los frameworks es hacer que nos centremos en el verdadero problema, y no preocuparnos por implementar funcionalidades que son de uso común en muchas aplicaciones, como podría ser el proceso de login de usuarios o establecer la conexión con la base de datos. Por ello, cuando usamos frameworks, nuestra mente ha de centrarse en el verdadero centro del problema y hacer fluir todos los detalles “menores”, ya que seguramente el framework nos dará una solución para ellos.

Ventajas de los frameworks:

Estructuración de directorios: así, si conocemos las reglas de estructuración, cuando necesitemos tocar algo en el código, sabremos rápidamente dónde localizar dicho código porque tendremos montada una buena jerarquía de directorios, mucho mejor de la que podríamos crearnos manualmente si no utilizáramos frameworks. No obstante, algunos frameworks dan facilidades para poder crear jerarquías propias.

El código de las librerías base que aportan los frameworks está muy probado. Sería posible desarrollar de forma manual una librería con las mismas funcionalidades, pero sería complejo poder alcanzar un nivel de testeo tan grande como el de los framework. (No es necesario reinventar la rueda).

Grandes comunidades (unas más grandes que otras) de usuarios. En estas comunidades de usuarios, algunos de ellos desarrollan módulos o extensiones para el framework y lo distribuyen gratuitamente.

Permite que el trabajo en equipo sea más sencillo. Si todo el equipo conoce el funcionamiento del framework, todos sabrán, por ejemplo, la estructura de directorios de la aplicación y sabrán localizar con facilidad el fichero de código fuente con el que requieran trabajar. Además, si, en un futuro, es necesario incorporar más personal a la plantilla, con que conozcan el framework será suficiente.

Desventajas de los frameworks:

La principal desventaja del uso de frameworks es la curva de aprendizaje. La afirmación de que el uso de framework hace que las aplicaciones se desarrollen en menos tiempo es cierta, pero siempre y cuando se conozca y se tenga experiencia en el uso del framework.

En la primera aplicación que se desarrolle con un framework nuevo, seguramente será necesario invertir más tiempo que si se desarrollase sin usar ningún framework, pero, a medio/largo plazo y una vez comprendido el framework, se podrá ahorrar más tiempo.

Una vez conocido el framework, permitirá un mantenimiento de la aplicación mucho más ágil, además de que la aplicación, desde sus cimientos, será mucho más robusta que si se desarrollara sin ningún framework.

3.2. Patrón Modelo-Vista-Controlador (MVC)

Como anteriormente se ha comentado, Struts se basa en el patrón de diseño MVC, el cual separa tres secciones diferenciadas llamadas Modelo, Vista y Controlador.

Esto busca separar el modelo de datos, las interfaces de usuario y la lógica de negocios en tres componentes diferentes, dividiendo las responsabilidades en tres capas bien diferenciadas.

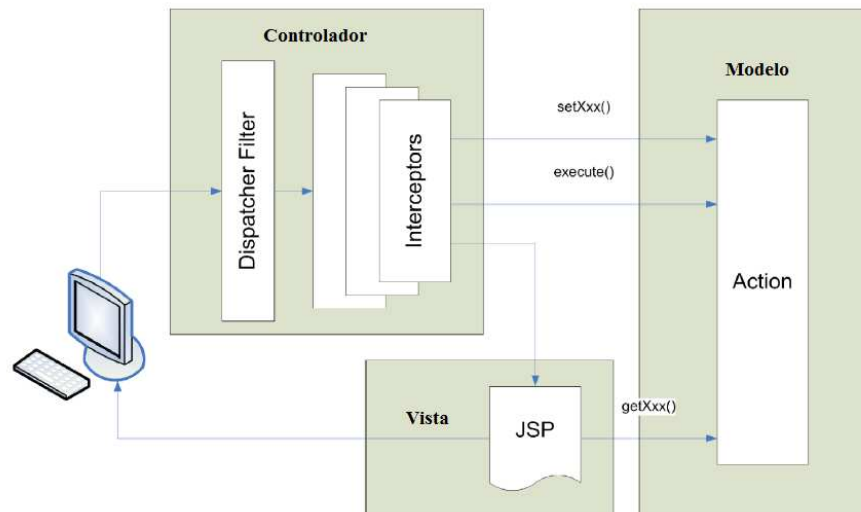


Ilustración 1: Explicación Modelo-Vista-Controlador

- El modelo hace referencia a los datos que maneja la aplicación y a las reglas de negocio que operan sobre ellos, y que se traducen en Struts 2 en las acciones (Actions).

El Modelo lo forman una serie de componentes independientes del Controlador y la Vista, que permiten su reutilización y el desacoplamiento entre las capas.

- La vista, encargada de generar la interfaz con la que la aplicación interacciona con el usuario, origina las respuestas que deben ser enviadas al cliente.

Cuando esta respuesta tiene que incluir información proporcionada por el Controlador, el código XHTML de la página no será fijo, sino que deberá ser generado de forma dinámica, por lo que su implementación corre a cargo de una página JSP (Java Server Page).

Las páginas JSP resultan apropiadas para la generación de las vistas por parte de los servlets, al tratarse de documentos de texto, resulta sencilla la incorporación de bloques estáticos XHTML.

- El controlador comunica la vista y el modelo, respondiendo a eventos generados por el usuario en la vista e invocando cambios en el modelo. Asimismo, devuelve a la vista la información del modelo necesaria para que pueda generar la respuesta adecuada para el usuario. El controlador se implementa en Struts 2 mediante el filtro FilterDispatcher.

Las ventajas de este componente implican un desarrollo más sencillo y limpio. Además, facilita el futuro mantenimiento de la aplicación realizándola más escalable.

Flujo de control de MVC:

1. El usuario realiza una acción en la interfaz
2. El controlador trata el evento de entrada (previamente se ha registrado)
3. El controlador notifica al modelo la acción del usuario, lo que puede implicar un cambio del estado del modelo (si no es una mera consulta)
4. Se genera una nueva vista. La vista toma los datos del modelo (El modelo no tiene conocimiento directo de la vista)
5. La interfaz de usuario espera otra interacción del usuario, que comenzará otro nuevo ciclo.

Frameworks que usan el patrón MVC:

- ASP.NET MVC Framework (Microsoft)
- Google Web Toolkit (GWT, para crear aplicaciones Ajax con Java)
- Apache Struts (framework para aplicaciones web J2EE)
- Ruby on Rails (framework para aplicaciones web con Ruby)

Ventajas:

- Separación total entre las capas de presentación, lógica de negocio y acceso a datos.
- Esta separación es fundamental para el desarrollo de aplicaciones consistentes, reutilizables y más fácilmente mantenibles. Su resultado es un ahorro de tiempo.
- En Java se puede implementar el patrón MVC con la clase Observer, pero de una manera sencilla, sin embargo con Struts se aplica en toda una aplicación web convencional.

3.3. ¿Qué es Struts2?

Apache Struts2 es un nuevo framework para desarrollar aplicaciones web. No se trata solo de una nueva versión de Struts, es un nuevo y completo Framework basado en el Framework WebWork de OpenSymphony. Cabe destacar que, aunque Struts2 originalmente fue conocido como WebWork2, al final se le asignó el nombre de Struts2, probablemente por razones de marketing.

Struts2 está construido sobre la tecnología de servlets, lo que le da una infraestructura básica a la hora de construir aplicaciones web sobre la plataforma Java. Esta tecnología es, en realidad, una especificación de Sun en la que se detalla el comportamiento del contenedor de servlets, el cual se encarga de analizar las peticiones HTTP, gestionar sesiones y compilar las páginas JSP, así como recoger las peticiones por URL y decidir qué servlet será ejecutado: `service()`. Es, en este contenedor de servlets, donde se desplegará la aplicación web VisualGate. Además, en este proyecto se usará Apache Tomcat como contenedor de servlets.

Struts2 sigue las mejores prácticas y patrones de diseño actuales. Este también fue el objetivo de su padre: Struts, el cual introdujo el patrón MVC dentro de los framework de aplicaciones web. Aprovechando toda esta experiencia en el mercado, se introdujeron varias características nuevas que hicieron el framework más limpio y flexible. Estas nuevas características incluyen los famosos interceptores de WebWork2 que permiten la interceptación de la petición antes de que se ejecute la lógica de la acción.

Por otro lado, se aprovecha la aparición de Java5 para incluir configuraciones mediante anotaciones y se reduce al máximo la configuración mediante ficheros XML. Por último, también hay que destacar la introducción de un poderoso lenguaje de expresiones llamado OGNL (Object-Graph Navigation Language) que, junto a la Value Stack, permitirá a Struts2 acceder a los datos de una manera más rápida y eficaz que otros Frameworks.

Struts2 se caracteriza por ser un framework extensible y elegante para el desarrollo de aplicaciones web empresariales de cualquier tamaño, ya que está diseñado y creado para agrupar todas las fases de desarrollo de una aplicación.

3.4. Características Struts2

Características Struts2:

Algunas de sus características principales son las siguientes:

- Todas las clases del framework están basadas en interfaces. El núcleo principal es independiente del HTTP.
- Basado en el patrón MVC bajo la plataforma J2EE.
- Alta configuración y extensibilidad.
- Permite el uso de plugins de componentes e integración con otros frameworks.
- Cualquier clase puede llegar a ser usada como una “Action class” (POJO)
- Las acciones de Struts2 son fácilmente integrables con el Framework Spring. (OF)
- Integración de AJAX, esto hace la aplicación más dinámica.

Otras características:

- Dependencia con la API de servlets: las acciones de Struts2 son POJO. Struts2 permite acceder a objetos de contexto para poder acceder al Request y al Response si es preciso. Esto se traduce en una reducción del acoplamiento con el framework o facilidad de testeo de las actions, entre otras.

- Clases Action: no es necesario que se extienda de una clase base, puesto que se tiene la opción de implementar una serie de interfaces como Action o extender a la clase ActionSupport, la cual implementa una serie de interfaces que debería proporcionar todas las herramientas necesarias para la correcta ejecución de la Action

- Validación Action: Struts2 presenta una validación declarativa proporcionada por “Xwork Validation Framework” y programática.

- Modelo ThreadingAction: las acciones de Struts2 son instanciadas por cada petición. Aunque parezca una penalización de rendimiento, no lo es tanto; ya que tiene a su favor que las acciones son POJO.

- Testeabilidad: las acciones de Struts2, como ya se ha comentado, tienen muy poco acoplamiento con el framework, llegando a ser nulo en la mayoría de casos. Esto lo hace más sencillo a la hora de las pruebas.

- Encapsulación de los parámetros de entrada: Struts2 usa las propiedades de la acción como las propiedades de entrada, eliminando así la necesidad de un segundo objeto para la entrada de datos. Las propiedades de la Action pueden ser accedidas desde la página web (Vista) por medio de las librerías de Tags que acceden directamente a los métodos de la Action. Además de esto, también se puede acceder inmediatamente a los objetos del dominio por medio de las Actions, puesto que soportan la inyección de dependencias.

- Lenguaje de Expresiones: Struts2 puede usar JSTL (usado por Struts1), pero el framework soporta un lenguaje de expresiones más flexible llamado OGNL (Object Graph Navigation Language).

- Enlazando valores dentro de la vista: Struts2 usa la tecnología Value Stack para que las librerías de Tags accedan a sus valores.

- Conversión de tipos: Struts2 usa OGNL para la conversión. Por otra parte, Struts2 incluye una serie de Converters para tipos básicos y comunes. Éstos se pueden configurar por medio de Annotations. Además, se pueden realizar Converters personalizados.

- Control de la ejecución de la Action: soporta diferentes ciclos de vida por Action. Éstos están basados en las pilas de interceptores. También se permite personalizar esta pila de interceptores y, por lo tanto, crear ciclos de vida personalizados por Action.

Ventajas:

La utilización de esta metodología conlleva una serie de ventajas que nos ayudan a reducir el tiempo requerido para el desarrollo y facilitar el mantenimiento de la aplicación web:

- Transporte automático de los datos introducidos en el cliente (JSP) hasta el controlador (Action) mediante formularios (ActionForm).
- Transporte automático de los datos enviados por el controlador (Action) a la parte de presentación (JSP) mediante formularios (ActionForm).
- Implementa la parte común a todas las aplicaciones en la parte del Controlador (ActionServlet). La parte particular de cada aplicación es fácilmente configurable (struts-config.xml).
- La separación de los componentes en capas (MVC) simplifica notablemente el desarrollo y su mantenimiento.

3.5. Configuración básica de Struts2

Para poder usar Struts2, es necesario instalar las librerías que utiliza el framework. Puede instalar las incluidas en el CD o mirar el anexo 2 para poder descargarlas de forma manual.

Se van a comentar los archivos necesarios para la configuración de un proyecto basado en Struts2. Al crear un nuevo proyecto web desde Netbeans (New project>Java Web> Web application) se creará la jerarquía básica y estos archivos por defecto.

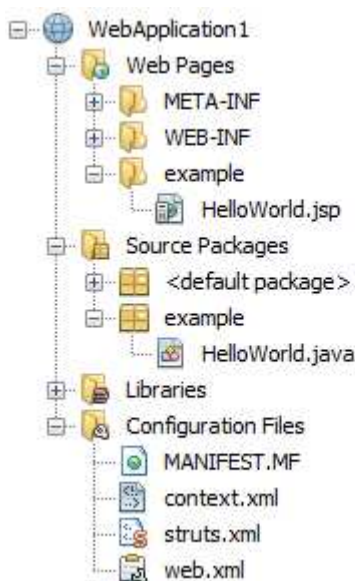


Ilustración 2: Jerarquía de un proyecto con Struts2

(a) Registro del Filtro Controlador en el web.xml

```
<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>
<session-config>
  <session-timeout>
    30
  </session-timeout>
</session-config>
<welcome-file-list>
  <welcome-file>example/HelloWorld.jsp</welcome-file>
</welcome-file-list>
```

En este archivo se registra el Filtro Controlador del framework Struts 2. Éste capturará todas las peticiones generadas por el cliente, puesto que el url-pattern tiene como valor "/*".

Nota: para ejecutar esta aplicación recuerde que es necesario disponer de un servidor de aplicaciones o contenedor web, por ejemplo: Apache Tomcat.

(b) Creación de un Componente Action

Se creará Action, llamado “UsuarioAction”, dentro de nuestro paquete ejemplo.java.action. Para este caso, sólo tendremos los atributos “nombre y apellido” con sus respectivos getters y setters. Además, es importante que esta clase herede de la clase ActionSupport para poder realizar las validaciones y otras funcionalidades adicionales. Sin embargo, no es indispensable que herede de esta clase. Por otra parte, el código que se ejecutará al invocar a este action estará contenido dentro la función execute().

```
package ejemplo.java.action;

import com.opensymphony.xwork2.ActionSupport;

public class UsuarioAction extends ActionSupport{

    private String nombre;
    private String apellido;

    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public String getApellido() {
        return apellido;
    }
    public void setApellido(String apellido) {
        this.apellido = apellido;
    }

    @Override
    public String execute() throws Exception {

        return SUCCESS;
    }
}
```

(c) Creación del archivo validation.xml

Este archivo nos servirá para validar campos de nuestro form. Para ello, debemos nombrar al archivo de la siguiente manera: MiAction-validation.xml. En nuestro ejemplo, será llamado “UsuarioAction-validation.xml”. A continuación, podemos observar las validaciones correspondientes a cada uno de nuestros atributos: “nombre y apellido”.

```
<!DOCTYPE validators PUBLIC
    "-//OpenSymphony Group//XWork Validator 1.0.2//EN"
    "http://www.opensymphony.com/xwork/xwork-validator-1.0.2.dtd">

<validators>
    <field name="nombre">
        <field-validator type="requiredstring">
            <message>Debe ingresar su nombre</message>
        </field-validator>
    </field>
    <field name="apellido">
        <field-validator type="requiredstring">
            <message>Debe ingresar su apellido</message>
        </field-validator>
    </field>
</validators>
```

(d) Registro del Action en el archivo struts.xml

Ahora registraremos nuestro UsuarioAction en el struts.xml. Para ello, es necesario que le asignemos un nombre, el cual nos servirá para invocarlo posteriormente desde nuestros forms. Asimismo, debemos detallar la clase a la que pertenece nuestro action. Por otro lado, también podemos especificar el método que será ejecutado al invocar nuestro action.

Además, dentro de cada registro de action, tenemos la etiqueta “Result”, donde colocaremos las vistas a las que será direccionado el usuario ante un determinado evento.


```

<package name="default" namespace="/" extends="struts-default">
    <default-action-ref name="index" />
    <action      name="usuarioAction_input"      class="ejemplo.java.action.UsuarioAction"
method="input">
        <result name="input">/usuario.jsp</result>
    </action>
    <action name="usuarioAction" class="ejemplo.java.action.UsuarioAction">
        <result>/usuario.jsp</result>
        <result name="input">/usuario.jsp</result>
        <result name="success">/bienvenido.jsp</result>
    </action>
</package>

```

Por ejemplo, si los datos del formulario no han sido ingresados correctamente, debemos colocar el valor de "input" en el result para que sea devuelto a la misma página. De lo contrario, si todos los campos han sido validados y la operación se realizó satisfactoriamente, se colocará el valor de "success" para continuar, donde en este caso se le redireccionará al usuario a otra vista.

(e) Creación de las Vistas: JSPs

Ahora, en el WebContent creamos nuestras vistas: usuario.jsp y bienvenida.jsp. El contenido de "usuario.jsp" será un form que invocará a nuestro action, registrado anteriormente en nuestro archivo struts.xml. Éste contendrá los campos como se muestran a continuación:

```

<% @ taglib prefix="s" uri="/struts-tags"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">

```

```

<title>Usuario</title>

</head>

<body>

<s:form action="usuarioAction">

    <s:textfield name="nombre" label="Nombre"/>

    <s:textfield name="apellido" label="Apellido"/>

    <s:submit value="Continuar"/>

    <s:reset value="Borrar"/>

</s:form>

</body>

</html>

```

Resultados:

Usuario

Nombre:

Apellido:

Usuario

Debe ingresar su nombre

Nombre:

Debe ingresar su apellido

Apellido:

Ilustración 3: Ejemplo básico Struts2

3.6. Ciclo de vida

Para comprender la forma de trabajar de Struts2 se va a explicar el funcionamiento interno del framework:

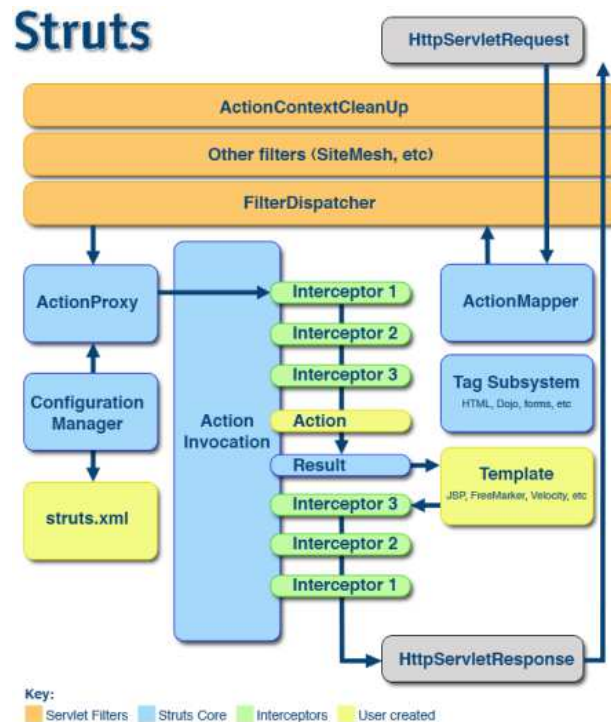


Ilustración 4: Ciclo de vida Struts2

Los pasos que habría que seguir serían los siguientes:

1. El usuario envía el requerimiento al servidor. (Llega un “request” a la aplicación)
2. El **FilterDispatcher**, haciendo uso del **ActionMapper**, interpreta el request recibido y determina la acción (action) que se va a ejecutar y que conjunto de interceptores pueden ser invocados.
3. Los interceptores (interceptors), previamente configurados, son aplicados al request. Se realizan las acciones de los interceptores de forma previa a la ejecución del método del action. Ej. interceptor que valide la autenticación del usuario, antes de realizarse el login se comprobará que es posible iniciar sesión de lo contrario el interceptor impedirá que se llegue a realizar la acción.
4. Se ejecuta el action determinado (lógica de negocio)
5. Se examina el resultado obtenido del Action y se determina el Result correspondiente
6. Se aplican los interceptores, pero en orden inverso. Por ejemplo, si en el punto 3 se aplicaron los interceptores int_1, int_2 e int_3, aquí se aplicarán int_3, int_2 e int_1.

7. Mediante el Result determinado, se genera la vista y, según la configuración definida sobre él, se invoca el proceso de generación de la vista.
8. Se retorna el control al contenedor, el cual envía la respuesta (la vista generada) al usuario.

Este esquema resume la forma de trabajar de Struts2, así mismo se destaca la importancia de los siguientes archivos de configuración:

- struts.xml: es el principal archivo de configuración del framework. En él se definen los ActionMapping de la aplicación, su división en paquetes (packages), el registro de interceptores, la asignación de los interceptores a los paquetes...
- struts-default.xml: este archivo define una configuración básica de un paquete, del cual es conveniente que el resto de paquetes hereden para obtener los beneficios del framework. Podemos redefinir esta configuración ubicando un archivo propio con el mismo nombre en el classpath.
- struts.properties: este es un archivo de propiedades que contiene un conjunto de pares key (valor que nos permite definir un conjunto de parámetros del framework). Este archivo es cargado por defecto por el framework (obteniéndolo de struts-core.jar), pero podemos redefinir los parámetros ubicándolo en el classpath de la aplicación.

3.7. Componentes Struts2

En el siguiente apartado, se van a exponer los componentes más importantes usados por Struts. Para ello, se proporcionarán diferentes ejemplos, así como toda la información posible para poder usar el framework de forma correcta.

(a) FilterDispatcher

Como ya se ha comentado con anterioridad, el FilterDispatcher es el “controlador” de nuestro modelo-vista-controlador encargado de comunicar las vistas con los modelos.

Es el punto de entrada del framework y, a partir de él, se lanza la ejecución del procesamiento para cada petición (request) que involucre al framework. Sus principales responsabilidades son:

- Ejecutar los actions (handlers para los request)
- Comenzar la ejecución de la cadena de interceptores asociados a la petición.
- Limpiar el ActionContext (para evitar fugas de memoria). El ActionContext es el contexto sobre el cual se ejecuta un action.

El trabajo del FilterDispatcher es verificar primero la petición url y determinar qué acción se va a invocar. El FilterDispatcher puede hacer esto utilizando el método de manipulación de cadenas. Sin embargo, esto no es un buen método debido a que, cada vez que se realiza un cambio, se tiene que volver a compilar.

Por ello, Struts2 utiliza el archivo de configuración struts.xml que hace coincidir los url con las clases de acción. Se tiene que crear un struts.xml y ponerlo en la carpeta WEB-INF/classes. En este archivo xml, se deben colocar las definiciones de acciones concretas, el nombre de la clase de acción y la url para invocar la acción.

Durante la invocación de un action, el FilterDispatcher realiza los siguientes pasos:

- 1) Consulta al Administrador de configuración para averiguar qué acción se tiene que llevar a cabo en función de la petición de url.
- 2) El primer interceptor rellena las propiedades de la acción.
- 3) Ejecuta la acción.
- 4) Ejecuta el resultado.

Nota:

En versiones anteriores de Struts2, se usaba el filtro FilterDispatcher; sin embargo, éste ha sido marcado como “deprecated” en las últimas versiones y se recomienda usar el nuevo filtro de Struts: Struts2PrepareAndExecuteFilter.

Este filtro realiza las mismas acciones que FilterDispatcher y se usaba hasta la versión 2.1.3. A partir de ahí y hasta la actual 2.3.15, se recomienda usar el nuevo filtro mejorado.

1. Ejemplo FilterDispatcher:

```
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3
  <filter>
    <filter-name>struts2</filter-name>
    <filter-class>org.apache.struts2.dispatcher.FilterDispatcher</filter-class>
  </filter>
  <filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
  <session-config>
    <session-timeout>
      30
    </session-timeout>
  </session-config>
  <welcome-file-list>
    <welcome-file>example/HelloWorld.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

Ilustración 5: Ejemplo web.xml FilterDispatcher

2. Struts2PrepareAndExecuteFilter

```
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
  <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>
  <session-config>
    <session-timeout>
      10
    </session-timeout>
  </session-config>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

Ilustración 6: Ejemplo web.xml Struts2PrepareAndExecuteFilter

(b) Actions

Los Actions son el núcleo del framework Struts2. Cada dirección url tiene mapeada una acción específica, la cual aporta lógica de proceso para la petición del usuario. Las Actions serán las encargadas de ejecutar la lógica necesaria para manejar una petición determinada. A diferencia de la versión anterior de Struts (versión 1), los Actions no están obligados a implementar o heredar de una interfaz o clase ya definida. Pueden ser POJO's. ("Plain Old Java Object": instancia de una clase que no extiende ni implementa nada en especial; por ejemplo, clase persona con una serie de atributos y operaciones sin más).

Igualmente, Struts2 posee una interfaz Action. Esta interfaz permite definir el método por defecto que se ejecutará sobre el Action para que no tengamos que definirlo por otro medio (existen varias formas de indicarle a Struts2 el método que se tiene que invocar sobre un action). También existe una implementación de esta interfaz, denominada ActionSupport, que proporciona una funcionalidad de gran utilidad, necesaria por un conjunto de interceptores, para poder manipular el Action a invocar.

Ejemplo de codificación de un action:

1. Mapear un action en una clase
2. Mapear un resultado con una vista
3. Escribir la lógica del controlador en la clase action.

```
<action name="MostrarProductos" class="action.admin.MostrarProductosAction">
  <result name="ERROR">admin.jsp</result>
  <result name="SUCCESS" type="chain">VerProductosAdmin</result>
</action>
```

Ilustración 7: Ejemplo mapeo action

Este mapeo especifica que, cuando el action “MostrarProductosAction” sea llamado y se ejecute, si el método execute del action devuelve error, se direccionará a la página admin.jsp. Si devuelve success procederá a realizar el action “VerProductosAdmin” (esto se debe a que es de tipo chain que, como veremos más adelante, encadena la ejecución con otro action, pero si dejamos el tipo por defecto, se podría mandar a otra página normal, al igual que se ha hecho con error).

Los action responden a una acción del usuario, ejecutan la lógica de negocio (o llaman a otras clases) y, posteriormente, devuelven un resultado que indica a Struts lo que se debe representar.

Generalmente, los acción extienden de la clase ActionSupport, que es proporcionada por el framework Struts 2. La clase ActionSupport proporciona implementaciones por defecto para las acciones más comunes (por ejemplo, ejecutar, entrada...) y también implementa varias interfaces útiles de Struts2.

En el método execute de cada action se deberá realizar la lógica que representa esa acción.

```
public class MostrarProductosAction extends ActionSupport implements RequestAware, SessionAware {

    //Scopes
    private Map session;
    private Map<String, Object> request;
    //Acceso a datos
    private String mensaje;

    public MostrarProductosAction() {
    }

    public String execute() throws Exception {
        ProductoDAO productoDAO = new ProductoDAO();
        ArrayList<Producto> listadoTotal;
        listadoTotal = (ArrayList<Producto>) productoDAO.findAllProducts();

        ArrayList<Producto> listado = new ArrayList<Producto>();

        for (int i = 0; i < listadoTotal.size(); i++) {
            Producto aux = listadoTotal.get(i);
            if (aux.isVisible()) {
                listado.add(aux);
            }
        }

        request.put("listaProductos", listado);
        return "SUCCESS";
    }
}
```

Ilustración 8: Ejemplo action, clase java

En este caso, la acción MostrarProductosAction se encargará de crear una lista con todos los productos de la aplicación (realizando una consulta sobre la base de datos). Una vez relleno el array y guardado en el scope request, devuelve SUCCESS marcando que su función ha sido completada de forma correcta.

Existe otra forma de crear acciones usando Struts2: realizar el mapeo utilizando anotaciones. En vez de hacerlas en el archivo struts.xml, se hacen en el propio action.

Un ejemplo de esta forma sería el action LogOff:

```
public class LogOffAction {

    public LogOffAction() {
    }

    @Action(value = "LogOff", results = {
        @Result(name = "SUCCESS", location = "/index.jsp")})
    public String execute() {

        //Limpiamos la sesion activa
        ActionContext.getContext().getSession().clear();
        return "SUCCESS";
    }
}
```

Ilustración 9: Ejemplo action, clase java con anotaciones

Antes del método execute, se marca el nombre del action con el que será llamado (value) y el result asociado a la acción.

Al igual que en el ejemplo anterior, una vez acaba su función; en este caso, borrar la sesión activa, al devolver success se direccionará a la página index.jsp.

(c) Results

Una vez que el Action ha sido procesada, es el turno de enviar la información resultante (Results) de vuelta al usuario. En Struts2, esta tarea está dividida en dos partes:

- El método de la Action que procesa la petición retorna un String como resultado. (Result)
- Según el resultado y la configuración, se procesará un tipo de Result (Result Type).

Cada vez que un Action termina su ejecución, se muestra un resultado al usuario. Estos resultados pueden ser de muchos tipos y tener muchos significados. El tipo más común de resultado es mostrar al usuario una nueva página web con cierta información, pero ¿y si quisiéramos hacer otra cosa? ¿Qué ocurre si queremos regresar un archivo como texto plano? ¿Y cuando queremos regresar un archivo binario? ¿Y si necesitamos redirigir la petición a otro Action o enviar al usuario a otra página?

Todas estas preguntas pueden ser respondidas mediante las combinaciones de results y results type. El framework struts2 nos da las opciones necesarias para satisfacer las necesidades con nos puedan surgir mientras se programa.

Cuando el método "execute" de un action termina de ejecutarse, este siempre regresa un objeto de tipo string. El valor de ese string se usa para seleccionar un elemento de tipo "<result>", si trabajamos con archivos de configuración en xml o "@Result", si trabajamos con anotaciones.

Dentro del elemento se indica qué tipo de Result se quiere regresar al usuario (enviar un recurso, redirigirlo a otro action, un archivo de texto plano, llamar a otro action de manera encadenada...), el recurso que se quiere regresar, algunos parámetros adicionales que requiera el result y, lo más importante, el nombre que tendrá este Result, en donde este nombre es precisamente lo que se indica con la cadena que es regresada de la ejecución del método "execute".

La interface "Action", que es implementada por la clase "ActionSupport", proporciona un conjunto de constantes que contienen los nombres de los results más comunes:

- "success": indica que todo ha salido correctamente (validaciones y el proceso de lógica en general) durante la ejecución del método
- "error": error nos indica que algo ha fallado durante la realización de la acción. Algún cálculo que no ha llegado a realizarse o alguna excepción que haya podido surgir
- "input": indica que algún campo proporcionado en un formulario es incorrecto. Puede ser que el valor no sea del tipo esperado o no cumpla con el formato o rango requerido
- "login": indica que el recurso al que el usuario está intentado acceder solo está disponible para usuarios registrados del sitio y, por lo tanto, debe loguearse primero. Para poder usar este result de forma correcta debemos crear un result global en la configuración del sitio
- "none": este es un result de un tipo especial, ya que le indica a Struts 2 que no debe enviar al usuario a ningún lugar (cancelación).

Tipos de results en Struts2:

- "dispatcher": este es el result más usado y el default. Envía como resultado una nueva vista, usualmente una jsp.
- "redirect": le indica al navegador que debe redirigirse a una nueva página que puede estar en nuestra misma aplicación o en algún sitio externo y, por lo tanto, este creará una nueva petición para ese recurso.
- "redirectAction": redirige la petición a otro Action de nuestra aplicación. En este caso, se crea una nueva petición hacia el nuevo Action.
- "chain": al terminarse la ejecución del Action se invoca otro Action. En este caso, se usa la misma petición para el segundo Action, el cual se ejecuta de forma completa con todo su stack de interceptores y sus results (es posible encadenar todos los action que se desee).
- "stream": permite enviar un archivo binario de vuelta al usuario.
- "plaintext": envía el contenido del recurso que indiquemos como un texto plano. Típicamente, se usa cuando necesitamos mostrar una jsp o html sin procesar

- "httpheader": permite establecer el valor de la cabecera http de código de estatus que se regresará al cliente. Así, por ejemplo, podemos usarlo para enviar un error al cliente (estatus 500), un recurso no encontrado (404), un recurso al que no tiene acceso (401)...
- "xslt": se usa cuando el resultado generará un xml. Este será procesado por una hoja de transformación de estilos para generar la vista que se mostrará al cliente.
- "freemarker": para integración con FreeMarker
- "velocity": para integración con Velocity
- "tiles": para integración con Tiles

Antes de empezar a describir los distintos tipos, es importante tener presente que cada uno de los tipos de results tienen siempre un parámetro por default. Si solo vamos a utilizar un parámetro en nuestro result y ese parámetro es el parámetro por default, entonces podemos omitir el elemento "<param>" y colocar directamente el valor dentro del elemento "<result>".

Dispatcher:

```
<action name="VerClientes" class="action.admin.VerClientesAction">
  <result name="SUCCESS" type="dispatcher">verClientes.jsp</result>
  <result name="ERROR" type="dispatcher">admin.jsp</result>
</action>
```

Ilustración 10: Result type dispatcher

Se podría poner:

```
<result name="SUCCESS" type="dispatcher">
  <param name="location">/verClientes.jsp</param>
</result>
```

Pero, como se ha comentado al usar el parámetro location, por defecto podemos omitirlo.

A su vez, dispatcher es el tipo básico y por defecto de results y también podemos omitirlo como se muestra en el siguiente ejemplo:

```
<action name="VerPedidos" class="action.cliente.VerPedidosAction">
  <result name="SUCCESS">mispedidos.jsp</result>
  <result name="ERROR">index.jsp</result>
</action>
```

Ilustración 11: Result type dispatcher simplificado

Redirect

Paramámetros:

- Location (por defecto): indica la ubicación a la cual irá el resultado (puede ser una página dentro de nuestro sitio o fuera de él).
- Parse: indica si el parámetro "location" (el anterior) puede ser obtenido a través de una expresión en OGNL, logrando que la ubicación se pueda establecer de forma dinámica (por defecto se encuentra activado).
- Anchor: se puede especificar un ancla para el resultado (un ancla es un lugar en una página html al que se le ha dado un nombre especial, único en el documento, y que nos permite enviar al navegador del usuario a ese punto específico). Esta parámetro es opcional.

Este result le indica al navegador que debe dirigirse hacia otro recurso de la aplicación o a algún lugar externo al sitio. Esto termina con la ejecución de la petición actual del cliente. Es muy útil cuando queremos enviar al usuario a otro sitio como, por ejemplo, Google.

Ejemplo dentro de la aplicación:

```
<action name="DeshacerPedido" class="action.cliente.DeshacerPedidoAction">
    <result name="SUCCESS" type="redirect">cesta.jsp</result>
</action>
```

Ilustración 12: Result type redirect

Ejemplo redirect externo:

```
<action name="redirectExterno" class="com.java.actions.RedirectExternoAction">
    <result type="redirect">http://www.google.es</result>
</action>
```

RedirectAction:

Se trata de una forma especializada del redirect, en donde el navegador del usuario es redirigido y ejecuta el Action que le especifiquemos. Nos permite especificar qué parámetros queremos enviar a este Action desde el archivo de configuración, los cuales pueden ser parámetros estáticos o dinámicos.

Parámetros:

- actionName (default): el nombre del Action al que seremos redirigidos.
- namespace: el namespace al que pertenece el Action (por defecto se usa el namespace actual).
- suppressEmptyParameters: evita que los parámetros que estén vacíos sean enviados al siguiente action. Su valor por defecto es "false" (parámetros vacíos serán enviados por defecto).

Ejemplo:

```
<result type="redirectAction">
<param name="actionName">nombreAction</param>
<param name="nombre">Jorge</param><!--parámetros necesarios para la realizar la action -->
</result>
```

Chain:

Este result permite que se invoque otro Action completo, incluyendo todos sus interceptores y su result.

La diferencia con RedirectAction es que éste le indicaba al navegador que debía redirigirse hacia otro action. En este caso, todos los Actions de la cadena se ejecutan en el servidor y éste nos regresa el resultado del último Action.

Parámetros:

- actionName (default): el nombre del Action que será encadenado
- namespace: indica en cuál namespace se encuentra el Action
- method: si queremos invocar un método en el Action que no sea el método "execute" lo indicamos con este parámetro.
- skipActions: una lista, separada por comas, de los nombres de los Actions que pueden ser encadenados a este.

```
<action name="BorrarCliente" class="action.admin.BorrarClienteAction">
  <result name="ERROR">admin.jsp</result>
  <result name="SUCCESS" type="chain">
    <param name="actionName">VerClientes</param>
  </result>
</action>
```

Ilustración 13: Result type chain

Stream:

Este result nos permite regresar un flujo de bytes al navegador del usuario para que este los interprete de alguna manera. Por ejemplo, podemos enviar un flujo de bytes que representen un archivo pdf y, si el navegador del cliente tiene el plugin adecuado, lo podrá visualizar.

Parámetros:

- `contentType`: el mime-type (el tipo del contenido) que es enviado al usuario. Por defecto, su valor es `"text/plain"`.
- `contentLength`: el número de bytes contenidos en el flujo que enviaremos al usuario. Se usa para que el navegador pueda mostrar al usuario, de forma correcta, una barra de progreso de la descarga.
- `contentDisposition`: este parámetro sirve para establecer el valor de la cabecera `"content-disposition"`, que el navegador usa para establecer el nombre del archivo que recibirá. El valor por defecto es `"inline"`.
- `inputName`: el nombre del atributo de nuestro Action que contiene el flujo de bytes. El atributo debe ser de tipo `"InputStream"`. Por defecto, se busca un atributo de nombre `"inputStream"`.
- `bufferSize`: el tamaño del buffer que se usa para copiar los valores de la entrada al flujo de salida del navegador. Por defecto, es de `"1024"`.
- `allowCaching`: indica si el navegador debe o no mantener en caché los datos que le hemos enviado. Esto quiere decir que solamente descargará el archivo una vez y, posteriormente, lo regresará del caché. Esto es útil si el contenido que se le envía siempre es el mismo. Su valor por defecto es `"true"`.
- `charset`: el juego de caracteres que se usan en el contenido (este parámetro es útil si regresamos texto plano en algún formato como xml).

Ejemplo:

```
<result type="stream">
  <param name="contentType">image/png</param>
  <param name="contentLength">${bytesArchivo}</param>
  <param name="inputName">streamImagen</param>
  <param name="contentDisposition">attachment;filename="${nombreImagen}"</param>
</result>
```

PlainText:

Este tipo de result envía el contenido de un archivo como texto plano al navegador. Con este result, podemos, por ejemplo, enviar el contenido de un jsp sin que esta sea procesada o cualquier otro archivo que pueda ser abierto con un editor de texto plano.

Parámetros:

- location (default): la ubicación del archivo que será mostrada como texto plano. Esta ubicación será relativa a la raíz del proyecto web.
- charset: el conjunto de caracteres con el que será mostrado el archivo.

```
<result type="plainText">
  <param name="location">/pagina.html</param>
  <param name="charset">UTF-8</param>
</result>
```

HttpHeader:

Este result permite establecer el valor de las cabeceras de la respuesta que se le envía al cliente. Con este result, podemos, por ejemplo, enviar estatus de error al cliente, indicando errores o falta de permisos para ejecutar ciertas acciones.

Parámetros:

- Status: el estatus de la respuesta http.
- Parse: indica si los parámetros "headers" deben ser parseados como expresiones OGNL (activado por defecto)
- Headers: los valores de las cabeceras que queramos establecer. Estos se establecen colocando el nombre de la cabecera que queramos fijar precedida por un punto.
- Error: el código de error http que será enviado como respuesta.
- ErrorMessage: el mensaje de error que será mostrado en la respuesta.

El uso de este tipo de result es enviar errores al usuario, dado que cualquier petición exitosa le llevará a su página correspondiente.

```
<action name="http404">
  <result type="httpheader">
    <param name="error">404</param>
    <param name="errorMessage">Lo sentimos el recurso enviar CV no se encuentra
  </result>
</action>
```

Ilustración 14: Result type httpheader

Xslt

XSLTResult utiliza xslt para transformar un objeto action en xml.

Parámetros:

- Location (default): la ubicación del archivo tras la ejecución
- Parse: activada por defecto. Si está desactivada el parámetro location no será parseado por expresiones OGNL.

Configuración relacionada: archivo de configuración struts.properties

struts.xslt.nocache - El valor predeterminado es falso. Si se establece en verdadero, desactiva el almacenamiento en caché de estilos. Bueno para el desarrollo, malo para producción.

```
<result name="success" type="xslt">
  <param name="location">foo.xslt</param>
  <param name="matchingPattern">^/result/[^/*]${</param>
  <param name="excludingPattern">.*(hugeCollection).*</param>
</result>
```

Freemarker

Representa una vista usando el motor de plantillas Freemarker.

FreeMarker es un motor de plantillas basado en Java que es una gran alternativa a la jsp. Es ideal para situaciones en las que los resultados de su acción, posiblemente, puedan cargarse desde fuera de un contenedor de servlets

Parámetros:

- Location (default): localización de la plantilla a procesar.
- Parse: activada por defecto. Si está desactivada, el parámetro location no será parseado por expresiones OGNL.
- contentType: por defecto, "text/html"
- writeIfCompleted: desactivado por defecto. Transmite sólo si no hay errores en el procesamiento de la plantilla.

Ejemplo:

```
<result name="success" type="freemarker">foo.ftl</result>
```

Archivo FTL

```
<html>
  <head>
    <title>Hello</title>
  </head>
  <body>
    Hello, ${name}
  </body>
</html>
```

Velocity

Usando el contenedor de servlets JspFactory, este resultado simula un entorno de ejecución jsp y, a continuación, muestra una plantilla de Velocity que será transmitida directamente a la salida del servlet.

Parámetros:

- Location (default): localización de la plantilla a procesar.
- Parse: activada por defecto. Si está desactivada, el parámetro location no será parseado por expresiones OGNL.

```
<result name="success" type="velocity">
  <param name="location">foo.vm</param>
</result>
```


Tiles

Es un framework de plantillas diseñado para permitir fácilmente la creación de páginas de la aplicación web con una apariencia consistente. Se puede utilizar tanto para la decoración de página como para la componentización.

```
<action name="sample" class="org.apache.struts2.tiles.example.SampleAction" >
  <result name="success" type="tiles">tilesWorks</result>
</action>

<%@ taglib uri="http://tiles.apache.org/tags-tiles" prefix="tiles" %>
<%@ taglib prefix="s" uri="/struts-tags" %>
<%-- Show usage; Used in Header --%>
<tiles:importAttribute name="title" scope="request"/>
<html>
  <head><title>Struts2 Showcase - <tiles:getAsString name="title"/></title></head>
<body>
  <tiles:insertAttribute name="header"/>
  <tiles:insertAttribute name="body"/>
  <p>Notice that this is a layout made in JSP</p>
</body>
</html>
```

(d)Interceptors

Los interceptores proveen un camino simple para añadir lógica de proceso alrededor del método que está siendo llamado en la action. Nos permiten añadir cierta funcionalidad que queremos aplicar a un conjunto de Actions.

- Proveen lógica de pre-procesamiento antes de que la acción sea llamada
- Interactúan con la Action, proporcionando información como la inyección de dependencias controlado por Spring o poner a punto los parámetros de la petición en la Action.
- Proveen lógica de pro-procesamiento después de que la acción haya sido llamada.

Los interceptores realizan tareas antes y después de la ejecución de un action y también pueden evitar que un Action se ejecute (por ejemplo, si estamos haciendo alguna validación que no se ha cumplido).

Sirven para ejecutar algún proceso particular que se quiere aplicar a un conjunto de Actions. De hecho, muchas de las características con que cuenta Struts2 son proporcionadas por los interceptores.

Si alguna funcionalidad que necesitamos no se encuentra en los interceptores de Struts 2, podemos crear nuestro propio interceptor y agregarlo a la cadena o pila que se ejecuta por default.

De la misma forma, podemos modificar la pila de interceptores de Struts 2, por ejemplo para quitar un interceptor o modificar su orden de ejecución según nos convenga. Además, a cada action se le puede aplicar más de un interceptor.

Struts2 permite crear pilas o stacks de interceptores y aplicarlas a los actions. Cada interceptor es aplicado en el orden en el que aparece en la pila. También podemos formar pilas de interceptores con base en otras pilas. Una de estas pilas de interceptores es aplicada por default a todos los Actions de la aplicación.

Con Struts2 vienen configuradas varias pilas por defecto. Las más importantes son: la basicStack y la defaultStack.

BasicStack, la pila que ofrece los interceptores mínimos necesarios para que una aplicación pueda proporcionar las funcionalidades básicas, tiene los siguientes interceptores:

- exception
- servletConfig
- prepare
- checkbox
- params
- conversionError

La segunda pila importante es la que se aplica por default a todos los actions de nuestra aplicación ("defaultStack"). Esta pila aplicará todos los interceptores, en el orden en que se encuentran:

- exception
- alias
- servletConfig
- prepare
- i18n
- chain
- debugging
- profiling
- scopedModelDriven

- modelDriven
- fileUpload
- checkbox
- staticParams
- conversionError
- validation
- workflow

Es importante tener en cuenta cuáles y en qué orden se ejecutan estos interceptores para comprender algunos de los ejemplos que se mostrarán.

Usando un interceptor podemos impedir que se realice una acción, como en el ejemplo siguiente en el que si se sube un archivo de distinto formato a png o jpeg, no se ejecuta la acción.

```
<action name="AnadirProducto" class="action.admin.AnadirProductoAction">
  <interceptor-ref name="fileUpload">
    <param name="allowedTypes">image/png,image/jpeg</param>
  </interceptor-ref>
  <interceptor-ref name="defaultStack"/>
  <result name="ERROR">verProductosAdmin.jsp</result>
  <result name="SUCCESS" type="chain">VerProductosAdmin</result>
  <result name="input" type="chain">VerProductosAdmin</result>
</action>
```

Ilustración 15: Interceptor struts.xml

Struts2 permite agregar a la ejecución de un action interceptores que no estén declarados en ninguna pila (y de igual forma configurarlos).

Ejemplos:

Logger: registra el inicio y el final de la ejecución de un action. Agregarlo a su ejecución consiste en:

- Conocer el nombre del interceptor (logger, timer...)
- Indicar en qué punto se debe comenzar a registrar la información.

```

<action name="AnadirProducto" class="action.admin.AnadirProductoAction">
  <interceptor-ref name="fileUpload">
    <param name="allowedTypes">image/png,image/jpeg</param>
  </interceptor-ref>
  <interceptor-ref name="defaultStack"/>
  <interceptor-ref name="logger" />
  <result name="ERROR">verProductosAdmin.jsp</result>
  <result name="SUCCESS" type="chain">VerProductosAdmin</result>
  <result name="input" type="chain">VerProductosAdmin</result>
</action>

```

Ilustración 16: Ejemplo interceptor logger

```

07-jul-2013 18:58:10 com.opensymphony.xwork2.util.logging.jdk.JdkLogger info
INFO: Starting execution stack for action //AnadirProducto
07-jul-2013 18:58:10 com.opensymphony.xwork2.util.logging.jdk.JdkLogger info
INFO: Finishing execution stack for action //AnadirProducto
07-jul-2013 18:58:10 com.opensymphony.xwork2.util.logging.jdk.JdkLogger info

```

Ilustración 17: Output interceptor logger

Timer: si lo colocamos antes del Action (de los results) sólo tomará el tiempo de ejecución del Action. Si lo colocamos antes de los interceptores, tomará el tiempo de ejecución del Action y del resto de los interceptores.

```

<action name="ActualizarProducto" class="action.admin.ActualizarProductoAction">
  <interceptor-ref name="fileUpload">
    <param name="allowedTypes">image/png,image/jpeg</param>
  </interceptor-ref>
  <interceptor-ref name="defaultStack"/>
  <interceptor-ref name="timer" />
  <result name="ERROR">verProductosAdmin.jsp</result>
  <result>verDetallesProductoAdmin.jsp</result>
  <result name="input" type="chain">VerProductosAdmin</result>
  <result name="SUCCESS" type="chain">VerProductosAdmin</result>

```

Ilustración 18: Ejemplo interceptor timer

```

INFO: Executed action [//ActualizarProducto!execute] took 160 ms.

```

Ilustración 19: Output interceptor timer

Interceptores en stack:

Para evitar tener que repetir declaraciones de interceptores, que usaremos de manera frecuente en nuestros actions, Struts2 permite agregar los interceptores que queramos en una pila de interceptores para los actions de un paquete de Struts y de todos los paquetes que hereden de este.

Se realiza en el propio archivo struts.xml, como se muestra a continuación:

```
<interceptors>
  <interceptor name="session" class="interceptor.SessionInterceptor" />
  <interceptor-stack name="sessionExpirayStack">
    <interceptor-ref name="defaultStack"/>
    <interceptor-ref name="session"/>
  </interceptor-stack>
</interceptors>
```

Ilustración 20: Interceptor global en struts.xml

(e) OGNL

OGNL es un lenguaje de expresiones de código abierto para Java, el cual permite obtener y establecer propiedades (a través de métodos ya definidos `getProperty` y `setProperty` similares a los presentes en todos los JavaBeans), así como la ejecución de métodos de clases Java.

Es el acrónimo de Object Graph Navigation Language, un lenguaje de expresiones muy poderoso que nos permite leer valores de objetos Java.

Este lenguaje nos permite no sólo leer valores, sino también ejecutar métodos (que regresen algún valor) para mostrar los valores o resultados de los mismos en las páginas jsp creadas, usando las etiquetas de Struts. Además proporciona una conversión automática de tipos que permite convertir datos desde texto http a objetos Java.

Struts no funciona exactamente usando una versión estándar de OGNL, usa una propia a la que agrega ciertas funcionalidades.

OGNL usa un contexto estándar de nombres para evaluar las expresiones. El objeto de más alto nivel es un Map, al cual llamamos mapa de contexto o contexto simplemente

Maneja siempre un objeto raíz dentro del contexto. Este objeto raíz es el objeto default al que se hacen las llamadas, a menos que se indique lo contrario. Cuando usamos una expresión, las propiedades del objeto raíz pueden ser referenciadas sin ninguna marca especial. Las referencias a otros objetos son marcadas con un signo de número (#).

El OGNL de Struts2 tiene un ValueStack, el cual permite simular la existencia de varias raíces. Todos los objetos que pongamos en el ValueStack se comportarán como la raíz del mapa de contexto.

En el caso de Struts2, el framework establece el contexto como un objeto de tipo "ActionContext", que es el contexto en el cual se ejecuta un action (cada contexto es básicamente un contenedor de objetos que un Action necesita para su ejecución, como los objetos "session", "parameters"...), y el ValueStack como el objeto raíz.

Siempre que Struts2 ejecuta uno de nuestros actions, los coloca en la cima de la pila y busca en el stack un objeto que tenga un getter para esa propiedad, en este caso nuestro Action.

Cuando hacemos la petición para el atributo "nombre", usando la etiqueta "s:property", Struts busca en el ValueStack, de arriba a abajo, un objeto que tenga un método "getNombre()".

Además del ValueStack, Struts2 coloca otros objetos en el ActionContext, incluyendo Maps que representan los contextos de "application", "session" y "request", los cuales veremos en detalle en el apartado scopes.

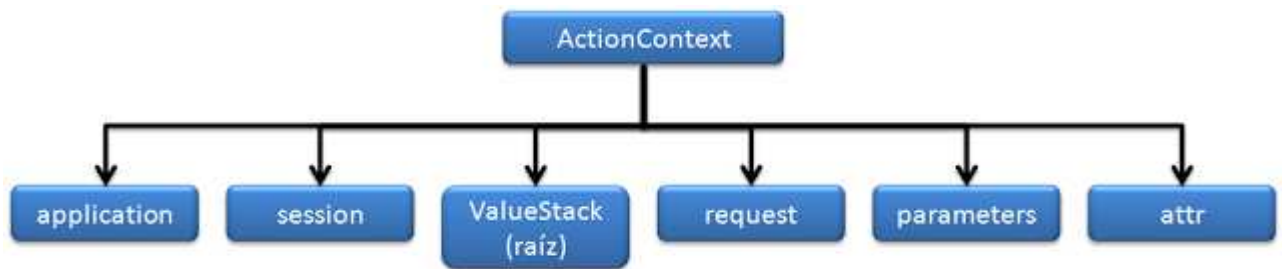


Ilustración 21: Objetos ActionContext Struts

- "application" representa el ApplicationContext, el contexto completo de la aplicación.
- "session" representa el HttpSession, la sesión del usuario.
- "request" representa el ServletRequest, la petición que se está sirviendo.
- "parameters" representa los parámetros que son enviados en la petición, ya sea por GET o por POST.
- "attr" representa los atributos de los objetos implícitos. Cuando preguntamos por un atributo, usando attr, se busca el atributo en los siguientes scopes: page, request, session, application. Si lo encuentra, regresa el valor del atributo y no continúa con la búsqueda.

Con OGNL no solo es posible acceder a los objetos dentro del ActionContext, sino prácticamente a cualquier objeto java que sea visible.

(f) Scopes

Cuando estamos desarrollando una aplicación web debemos almacenar la información que va ser procesada de distinta manera. Dependiendo de cuál sea el propósito de esta información, queremos que tenga un tiempo de vida más corto o más largo. Alguna información deberá permanecer disponible durante todo el momento que viva nuestra aplicación, mientras que otra solo nos interesará que viva durante una petición.

Estos tiempos de vida son llamados scopes y, en las aplicaciones web, tenemos un cierto número de ellos. A la información que colocamos en los distintos scopes, la llamamos atributos.

Las aplicaciones web con Java tienen básicamente tres scopes o tiempos de vida:

- **Application:** es el scope más largo, ya que abarca el tiempo de vida completo de la aplicación; esto es, los datos vivirán mientras la aplicación esté activa.
- **Session:** este scope nos permite tener datos que vivirán a lo largo de múltiples peticiones http para un mismo usuario, mientras el usuario esté dentro de la aplicación. Cada usuario verá únicamente sus datos y no habrá forma de que vea los de los demás.
- **Request:** este es el scope más pequeño. Los datos asociados con la petición únicamente estarán disponibles mientras se realiza dicha petición.

La información o atributos que se puede colocar dentro de estos scopes son pares nombre valor, en donde el nombre debe ser una cadena y el valor puede ser cualquier objeto que nosotros queramos.

Struts 2 nos proporciona tres formas para colocar y leer los atributos que se encuentren en estos scopes:

- Implementación de interfaces Aware
- Uso del objeto "ActionContext"
- Uso del objeto "ServletActionContext"

*** Para poder acceder a objetos `HttpServletRequest` y `ServletContext`, podemos usar las interfaces `aware` y el objeto `ServletActionContext`.

Interfaces aware:

Struts2 proporciona un conjunto de interfaces "Aware", las cuales permiten que nuestros Actions reciban cierta información al momento de inicializarlos. La mayoría de estas interfaces proporcionan un Map con los pares nombre valor de los atributos de cada uno de los scopes. Struts2 contiene las siguientes interfaces "Aware" para obtener y leer atributos de los scopes:

- `ApplicationAware`
- `SessionAware`
- `RequestAware`

```

public interface RequestAware{
    public void setRequest(Map<String, Object> map);
}

public interface SessionAware{
    public void setSession(Map<String, Object> map);
}

public interface ApplicationAware{
    public void setApplication(Map<String, Object> map);
}

```

```

/**
 * Action para cancelar el pedido y eliminar la cesta de la sesion activa
 *
 * @author Jorge
 */
public class DeshacerPedidoAction extends ActionSupport implements RequestAware, SessionAware {

```

Ilustración 22: Clase implementando SessionAware y RequestAware

Es posible obtener el valor de los atributos colocados en estos scopes usando el operador punto (".") o colocando el nombre del parámetro entre corchetes ("[" y "]").

```

<s:if test="#session.user.privilegios == 1">
    <jsp:forward page="mainadmin.jsp" />
</s:if>

```

Ilustración 23: Uso de scope session

```

<h2><span class="text-success" style="text-align:center">
    <s:property value="%{#request.idPedido}"/></span></h2>

```

Ilustración 24: Uso de scope request

```

<s:property value="#application.datoAplicacion" /><br />

```


Cómo comprobar el tiempo de vida de los scopes:

- Request: al generar una nueva petición se borrará el dato guardado.
- Session: se puede esperar a que se termine la sesión de forma automática (por defecto 30 minutos) o usar otro navegador para acceder a la misma dirección.
- Application: lo guardado en application sólo se borrará hasta que se detenga el servidor.

ActionContext:

La segunda forma que tenemos de manejar los atributos de los scopes es a través de un objeto especial de Struts2 llamado "ActionContext". Este objeto es el contexto en el cual el Action se ejecuta. Cada contexto es básicamente un contenedor para objetos que un Action necesita para su ejecución, como la sesión sus parámetros.

Dentro de los objetos que se encuentran dentro de "ActionContext", están justamente los mapas que contienen los atributos de sesión y aplicación.

Para obtener una instancia del "ActionContext", se usa el método estático "getContext()" que nos regresa a la instancia de "ActionContext" adecuada para el Action que estamos ejecutando:

```
public String execute() {  
    session = ActionContext.getContext().getSession(); // Get session
```

Ilustración 25: Scope ActionContext

Y una vez que se tiene una referencia al "ApplicationContext", lo único que debemos hacer es usar el método "getApplication()", para obtener un mapa con los atributos del scope application y, "getSession()", para obtener un mapa con los atributos del scope session:

```
Map<String, Object> application = contexto.getApplication();  
Map<String, Object> sesion = contexto.getSession();  
  
application.put("datoAplicacion", datosAplicacion);  
sesion.put("datoSesion", datoSesion);
```

ServletActionContext

ServletActionContext no es muy recomendado de usar, ya que rompe con el esquema de trabajo de Struts 2, el cual oculta detalles de la especificación de servlets.

Con esta última forma, tendremos acceso directo a los objetos "HttpServletRequest" y "ServletContext" de la especificación de servlets, haciendo uso de otro objeto especial de Struts2: "ServletActionContext".

Este objeto especial "ServletActionContext" contiene una serie de métodos estáticos que nos dan acceso a otros objetos de utilidad (entre ellos "ActionContext"). Los métodos que, en este momento, nos interesan son: "getRequest", con el que podemos obtener el objeto "HttpServletRequest" (desde el cual tenemos además acceso a la sesión) asociado con la petición que el Action está sirviendo, y "getServletContext", que nos regresa el objeto "ServletContext" que representa el contexto de la aplicación web.

"ServletActionContext" nos proporciona métodos estáticos que nos regresan estas referencias de forma directa, por lo que lo único que tenemos que hacer es invocarlos, para obtener la sesión (el objeto "HttpSession"). Se obtiene usando el método "getSession()" del objeto "request":

```
HttpServletRequest request = ServletActionContext.getRequest();  
ServletContext context = ServletActionContext.getServletContext();  
HttpSession session = request.getSession();
```

Podemos usar el acceso a HttpServletRequest para poder invalidar la sesión de un usuario.

```
request.getSession().invalidate();
```

Se puede comprobar que, algunas veces, es útil tener acceso a los objetos de la especificación de servlets cuando el framework que estamos usando no nos pueda, por la razón que sea, dar todas las facilidades que necesitemos para hacer algunas cosas específicas.

(g) Etiquetas

Struts 2 proporciona una librería de etiquetas que facilitan, entre otras, las tareas de validación e internacionalización. Estas etiquetas, cuyo tld podemos encontrar en META-INF/struts-tags.tld, en el jar de struts2-core, pueden utilizarse tanto con jsp, como con Velocity o FreeMarker.

Antes de empezar a detallar algunas de las etiquetas más importantes de Struts2, se van a enumerar algunos atributos comunes a todas las etiquetas de interfaz de usuario.

- `cssClass`: el atributo `class` de html. Indica la clase css que se va a utilizar para el elemento html generado.
- `cssStyle`: el atributo `style` de html. Permite definir el estilo del elemento inline, en lugar de utilizar un archivo css externo o una etiqueta `style`.
- `disabled`: determina si el control está deshabilitado.
- `label`: etiqueta que acompañará al widget. Genera una clásica etiqueta `label` de html.
- `required`: booleano indicando si el campo es obligatorio. Si es así, muestra un asterisco al lado de la etiqueta.
- `tabindex`: el atributo `tabindex` de html. Es utilizado para establecer el orden que se debe seguir al recorrer los controles cuando el usuario pulsa la tecla de “Tabulación”.
- `template`: plantilla que se utiliza.
- `theme`: Tema que se emplea.

Cabe destacar la importancia de las siguientes etiquetas:

ACTION

Permite ejecutar una acción desde una vista indicando el nombre de la acción y, opcionalmente, el espacio de nombre. Se pueden pasar parámetros utilizando la etiqueta “`param`”.

`org.apache.struts2.views.jsp.ActionTag`

- `executeResult`: determina si el resultado de la acción (normalmente otra vista) se debe ejecutar / renderizar también.
 - `ignoreContextParams`: indica si se deben incluir los parámetros de la petición actual al invocar la acción.
 - `name` (requerido): el nombre de la acción que se va a ejecutar.
 - `namespace`: el espacio de nombres de la acción que se va a ejecutar
- `index.jsp`

```
<%@ taglib uri="/struts-tags" prefix="s"%>
<html>
<body>
Antes de s:action<br/>
<s:action name="Accion" executeResult="true"/><br/>
Después de s:action
</body>
</html>
```

Accion.java

```
import com.opensymphony.xwork2.ActionSupport;

public class Accion extends ActionSupport {
    private String web;

    public String getWeb() {
        return web;
    }

    public String execute() {
        web = "www.lacuevalibre.blogspot.com";
        return SUCCESS;
    }
}
```

resultado.jsp

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

```
Visita <s:property value="web"/>
```

ACTIONERROR

Muestra los errores que se produjeron en las acciones, si es que existen. Podemos añadir errores utilizando el método `addActionError(String error)` de la interfaz `ValidationAware`, que `ActionSupport` implementa. Los métodos y propiedades de `ValidationAware` también estarán disponibles en la vista, por lo que es posible, en su lugar, comprobar si existen errores con el método `hasActionErrors()` e iterar directamente sobre la colección `actionErrors`.

`org.apache.struts2.views.jsp.ui.ActionErrorTag`

Accion.java

```
import com.opensymphony.xwork2.ActionSupport;
```

```
public class Accion extends ActionSupport {  
    public String execute() {  
        addActionError("¡error!");  
        return ERROR;  
    }  
}
```

resultado.jsp

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

<html>
<body>
<s:actionerror />
</body>
</html>
```

ACTIONMESSAGE

Similar a actionerror, pero, en lugar de errores en la acción, sirve para mostrar mensajes de la acción, los cuales añadimos utilizando el método addActionMessage(String mensaje) de la interfaz ValidationAware. Como el anterior, también podríamos utilizar el método hasActionMessages() para comprobar la existencia de mensajes e iterar sobre la colección de strings actionMessages.

org.apache.struts2.views.jsp.ui.ActionMessageTag

Accion.java

```
import com.opensymphony.xwork2.ActionSupport;

public class Accion extends ActionSupport {
    public String execute() {
        addActionMessage("Mensaje");
        return SUCCESS;
    }
}
```

resultado.jsp

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

<html>

<body>

<s:actionmessage/>

</body>

</html>
```

BEAN

Instancia un Java Bean. Se puede pasar valores a las propiedades del bean utilizando etiquetas param.

org.apache.struts2.views.jsp.BeanTag

- name (requerido): la clase que se debe instanciar.
- var: nombre con el que se añadirá la instancia a ValueStack.

Usuario.java

```
public class Usuario {
    private String nombre;

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
}
```

resultado.jsp

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

<html>
<body>
<s:bean name="Usuario" var="miUsuario">
    <s:param name="nombre">Jorge</s:param>
</s:bean>

Bienvenido <s:property value="#miUsuario.nombre"/>
</body>
</html>
```

CHECKBOX

Crea un checkbox html. Para una lista de varios checkboxes relacionados, podemos utilizar la etiqueta checkboxlist.

org.apache.struts2.views.jsp.ui.CheckboxTag

- value: booleano que indica si el checkbox está marcado o no.

index.jsp

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

<html>
<body>
<s:form action="Accion">
    <s:checkbox label="Aceptar condiciones" name="condiciones" value="true" />
    <s:submit value="Enviar" />
</s:form>
</body>
</html>
```


Accion.java

```
import com.opensymphony.xwork2.ActionSupport;

public class Accion extends ActionSupport {

    private boolean condiciones;

    public boolean isCondiciones() {
        return condiciones;
    }

    public void setCondiciones(boolean condiciones) {
        this.condiciones = condiciones;
    }

    public String execute() {
        return SUCCESS;
    }
}
```

resultado.jsp

```
<%@ taglib uri="/struts-tags" prefix="s"%>
<html>
<body>
<s:if test="condiciones">
Marcado como sí
</s:if>
<s:else>
Marcado como no
</s:else>
</body>
</html>
```

COMBOBOX

Crea una combinación de select y caja de texto. El valor de la caja de texto se autorellena, según el elemento seleccionado en el select.

org.apache.struts2.views.jsp.ui.ComboBoxTag

- emptyOption: indica si queremos añadir una opción vacía.
- list (requerido): iterable con los valores con los que generar la lista.
- listKey: propiedad de los objetos del iterable del que el checkbox correspondiente tomará su valor.
- listValue: exactamente igual al anterior, pero, en lugar del valor, el contenido.
- name: nombre que se va a usar para el elemento.
- readonly: si queremos que el usuario pueda escribir sus propios valores en la caja de texto.

Accion.java

```
import com.opensymphony.xwork2.ActionSupport;
import java.util.*;

public class Accion extends ActionSupport {
    private List<String> lenguajes;

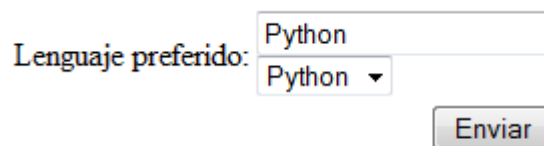
    public List<String> getLenguajes() {
        return lenguajes;
    }

    public String execute() {
        lenguajes = new ArrayList<String>();
        lenguajes.add("Phyton");
        lenguajes.add("Pascal");
        lenguajes.add("Ruby");
        lenguajes.add("C#");
        lenguajes.add("C++");
        lenguajes.add("Java");
        return SUCCESS;
    }
}
```

resultado.jsp

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

<html>
<body>
<body>
<s:form action="Otro">
    <s:combobox list="lenguajes" name="nombre-lenguajes"
        label="Lenguaje preferido" readonly="true" />
    <s:submit value="Enviar" />
</s:form>
</body>
</html>
```



The screenshot shows a web form with the label "Lenguaje preferido:". To the right of the label is a combobox (dropdown menu) with "Python" selected and a small downward arrow. Below the combobox is a button labeled "Enviar".

Ilustración 26: Etiqueta combobox resultado

DATE

Permite mostrar una fecha almacenada en una cierta variable, indicando opcionalmente el formato que se va a emplear.

org.apache.struts2.views.jsp.DateTag

- **format:** formato que se va a utilizar para mostrar la fecha. Si queremos usar siempre el mismo formato, podemos crear un archivo properties con una entrada `struts.date.format`. Por defecto, se utiliza el formato `DateFormat.MEDIUM`.
- **name (requerido):** nombre de la variable que contiene la fecha que se va a mostrar.
- **nice:** utiliza un formato que facilita la lectura. Por defecto, se utiliza inglés para mostrar los mensajes. Si queremos traducirlo a algún otro idioma, tendremos que recurrir a las funciones de internacionalización de Struts2.

Accion.java

```
import com.opensymphony.xwork2.ActionSupport;

import java.util.Calendar;
import java.util.Date;

public class Accion extends ActionSupport {
    private Date ahora;
    private Date anyoNuevo;

    public Date getAhora() {
        return ahora;
    }

    public Date getAnyoNuevo() {
        return anyoNuevo;
    }

    public String execute() {
        ahora = new Date();

        Calendar cal = Calendar.getInstance();
        int anyo = cal.get(Calendar.YEAR);
        cal.set(anyo + 1, Calendar.JANUARY, 1);
        anyoNuevo = cal.getTime();

        return SUCCESS;
    }
}
```

resultado.jsp

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

<html>

<body>

<body>

Hoy es <s:date name="ahora" format="d 'de' MMMM 'de' yyyy" /><br/>

Año nuevo <s:date name="anyoNuevo" nice="true" />

</body>

</html>
```

DEBUG

Imprime información de depuración (entre otros, el contenido de ValueStack).

org.apache.struts2.views.jsp.ui.DebugTag

DOUBLESELECT

Crea dos elementos select html, en donde los valores de uno de ellos cambian dependiendo de lo seleccionado en el primero.

org.apache.struts2.views.jsp.ui.DoubleSelectTag

- doubleList (requerido): lista con los valores que tendrá el segundo select.
- doubleMultiple: determina si en el segundo select se pueden seleccionar varios valores o solo uno.
- doubleName (requerido): nombre del elemento.
- list (requerido): lista con los valores que tendrá el primer select.

Artista.java

```
import java.util.List;

public class Artista {

    private String nombre;
    private List<String> canciones;

    public Artista(String nombre, List<String> canciones) {
        this.nombre = nombre;
        this.canciones = canciones;
    }

    public String getNombre() {
        return nombre;
    }

    public List<String> getCanciones() {
        return canciones;
    }
}
```

Accion.java

```
import com.opensymphony.xwork2.ActionSupport;
import java.util.List;
import java.util.ArrayList;

")
public class Accion extends ActionSupport {
    private List<Artista> artistas;
```

```

public List<Artista> getArtistas() {
    return artistas;
}

public String execute() {
    artistas = new ArrayList<Artista>();

    List<String> canciones_p = new ArrayList<String>();
    canciones_p.add("El equilibrio es imposible");
    canciones_p.add("Años 80");
    canciones_p.add("Promesas que no valen nada");
    Artista piratas = new Artista("Los piratas", canciones_p);
    artistas.add(piratas);

    List<String> canciones_i = new ArrayList<String>();
    canciones_i.add("The Tropper");
    canciones_i.add("Fear of the dark");
    canciones_i.add("Run to the hills");
    Artista ironMaiden = new Artista("Iron Maiden", canciones_i);
    artistas.add(ironMaiden);

    return SUCCESS;
}
}

```

resultado.jsp

```
<%@ taglib uri="/struts-tags" prefix="s"%>
<html>
<body>
<body>
<s:form action="Otro">
    <s:doubleselect label="Selecciona la canción" list="artistas"
        listValue="nombre" doubleList="canciones" doubleName="cancion" />
    <s:submit value="Enviar" />
</s:form>
</body>
</html>
```

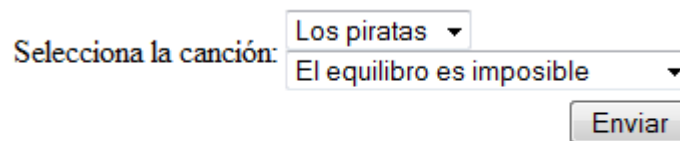


Ilustración 27: Etiqueta DoubleSelect Resultado

FIELDERROR

Muestra los errores que se han detectado al validar los campos del formulario. Por defecto, se muestran todos los errores, pero podemos seleccionar que se muestren sólo los relativos a ciertos campos pasándole etiquetas param. El funcionamiento es parecido al de `actionerror` y `actionmessage`: podemos añadir errores utilizando el método `addFieldError` (`String fieldName`, `String errorMessage`) de la interfaz `ValidationAware`, que `ActionSupport` implementa. No obstante, hay que tener en cuenta que, si redirigimos al formulario de entrada, el propio formulario imprimirá estos errores por defecto sin necesidad de añadir esta etiqueta. Los métodos y propiedades de `ValidationAware` también estarán disponibles en la vista, por lo que es posible en su lugar comprobar si existen errores con el método `hasFieldErrors()` e iterar directamente sobre la colección `fieldErrors`.

`org.apache.struts2.views.jsp.ui.FieldErrorTag`

index.jsp

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

<html>

<body>

<s:fielderror />

<s:form action="Accion">
    <s:textfield label="Nombre" name="nombre" />
    <s:submit value="Enviar" />
</s:form>

</body>
</html>
```

Accion.java

```
import com.opensymphony.xwork2.ActionSupport;

public class Accion extends ActionSupport {
    private String nombre;

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
}
```

```
public String execute() {  
    if(nombre.length() > 10) {  
        addFieldError("nombre", "El nombre no puede tener más de 10 caracteres.");  
        return ERROR;  
    }  
  
    return SUCCESS;  
}  
}
```

resultado.jsp

```
<%@ taglib uri="/struts-tags" prefix="s"%>  
  
<html>  
<body>  
<body>  
<s:fielderror />  
</body>  
</html>
```

FORM

Crea un elemento form de html.

org.apache.struts2.views.jsp.ui.FormTag

- action: acción a la que se enviará la petición con los datos del formulario. También se pueden enlazar otras páginas o servlets. Si no utilizamos el atributo para especificar el destino, se utiliza la misma página del formulario.
- namespace: espacio de nombres al que pertenece la acción a la que se enviará la petición. Por defecto, se utiliza el espacio de nombres actual.
- validate: si queremos validar los campos del formulario antes de enviarlos.

IF-ELSEIF-ELSE

La típica sentencia condicional.

org.apache.struts2.views.jsp.IfTag

org.apache.struts2.views.jsp.ElseIfTag

org.apache.struts2.views.jsp.ElseTag

test (requerido para if y elseif): la expresión a comprobar

index.jsp

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

<html>
<body>

<s:if test="ventas < 1000">Comisión del 10%</s:if>
<s:elseif test="ventas < 2000">Comisión del 20%</s:elseif>
<s:else>Comisión del 30%</s:else>

</body>
</html>
```

ITERATOR

Para iterar sobre colecciones. En cada iteración el objeto recuperado se coloca en ValueStack para poder acceder a sus propiedades fácilmente.

org.apache.struts2.views.jsp.IteratorTag

- status: crea una instancia de IteratorStatus con el nombre indicado. Este objeto expone algunas propiedades muy útiles como index (índice del elemento actual), first (booleano que indica si es el primer elemento), even (booleano que indica si es impar), last (booleano que indica si es el último elemento) u odd (booleano que indica si es par).
- value: colección sobre lo que iterar.

Accion.java

```
import com.opensymphony.xwork2.ActionSupport;
import java.util.List;
import java.util.ArrayList;

public class Accion extends ActionSupport {
    private List<String> lenguajes;

    public List<String> getLenguajes() {
        return lenguajes;
    }

    public String execute() {
        lenguajes = new ArrayList<String>();
        lenguajes.add("Python");
        lenguajes.add("Java");
        lenguajes.add("Ruby");
        lenguajes.add("C#");
        lenguajes.add("C++");
        lenguajes.add("Lisp");
        return SUCCESS;
    }
}
```

resultado.jsp

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

<html>

<body>
Tus lenguajes preferidos son:
<ul>
    <s:iterator value="lenguajes">
        <li><s:property /></li>
    </s:iterator>
</ul>
</body>
</html>
```

PARAM

Utilizada para añadir parámetros a otras etiquetas.

org.apache.struts2.views.jsp.ParamTag

- name: nombre del parámetro.
- value: valor del parámetro.

PROPERTY

Muestra una propiedad de ValueStack u otro objeto de ActionContext.

org.apache.struts2.views.jsp.PropertyTag

- default: valor que se muestra en caso de que el valor pedido sea nulo.
- escape: determina si queremos que se escape el html. Por defecto, es true.
- value: valor que se muestra.

Accion.java

```
import com.opensymphony.xwork2.ActionSupport;

public class Accion extends ActionSupport {
    private String web;

    public String getWeb() {
        return web;
    }

    public void setWeb(String web) {
        this.web = web;
    }

    public String execute() {
        web = "mundogeek.net";
        return SUCCESS;
    }
}
```

resultado.jsp

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

<html>
<body>

Visita <s:property value="web" default="google.es" />

</body>
</html>
```

RESET

Crea un botón que borra los datos introducidos en el formulario.

org.apache.struts2.views.jsp.ui.ResetTag

- value: texto que se utiliza para el botón

resultado.jsp

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

<html>
<body>

<s:form action="Otro">
  <s:textfield label="Nombre" />
  <s:password label="Contraseña" />
  <s:reset value="Borrar" />
  <s:submit value="Enviar" />
</s:form>

</body>
</html>
```

SET

Asigna un valor a una variable, opcionalmente, indicando el ámbito al que añadirla. El valor se puede indicar utilizando el atributo value o encerrando el valor en la propia etiqueta.

org.apache.struts2.views.jsp.SetTag

- name: nombre que se utiliza para la variable.
- scope: ámbito en el que se añade la variable. Puede ser application, session, request, page o action. Por defecto, se utiliza action.
- value: valor a asignar.

resultado.jsp

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

<html>
<body>
<s:set name="nombre">Raúl</s:set>
Hola <s:property value="nombre" />
</body>
</html>
```

SUBMIT

Crea un botón para enviar el formulario.

org.apache.struts2.views.jsp.ui.SubmitTag

- value: texto que se utiliza para el botón

resultado.jsp

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

<html>
<body>
<s:form action="Otro">
    <s:textfield label="Nombre" />
    <s:password label="Contraseña" />
    <s:reset value="Borrar" />
    <s:submit value="Enviar" />
</s:form>
</body>
</html>
```


(h) Formularios

En el desarrollo de aplicaciones web, una de las partes más importantes que existen (sino es que la más importante) es el manejo de datos que son recibidos del usuario a través de los formularios de nuestra aplicación.

Recepción de parámetros simples:

Forma muy rápida de obtener parámetros de un formulario. Lo único que hay que hacer es:

- Colocar un formulario en nuestra jsp usando la etiqueta `<s:form>`
- Colocar los campos del formulario usando las etiquetas correspondientes de Struts
- Crear un Action y colocar setters para cada uno de los elementos que recibiremos a través del formulario
- Procesar los datos del formulario en el método "execute".

Ejemplo1:

```
<s:form action="datosUsuario">
  <s:textfield name="nombre" label="Nombre" />
  <s:textfield name="username" label="Username" />
  <s:password name="password" label="Password" />
  <s:textfield name="edad" label="Edad" />
  <s:textfield name="fechaNacimiento" label="Fecha de Nacimiento" />
  <s:submit value="Enviar" />
</s:form>
```

Ejemplo2:

```
<s:form action="AnadirProductoCesta" method="post">
  <div>
    <div class="input-prepend">
      <span class="add-on">
        <i class="icon-shopping-cart"></i>
      </span>
      <input style="width: 20px" type="text" value="1" id="" name="cantidad">
      <input name="id" type="hidden" value="<s:property value="id"/>">
    </div>
    <button type="submit" class="btn btn-primary pull-right">
      <i class="icon-shopping-cart icon-white"></i> Añadir</button>
    </div>
</s:form>
```

Ilustración 28: Ejemplo formulario con Struts2

Este ejemplo consiste en el paso del id del producto y de la cantidad deseada a la cesta de la compra. Corresponde a la siguiente pantalla:



Ilustración 29: Formulario visto desde la vista

Al pulsar sobre el botón “Añadir” le pasaremos a nuestro action el id del producto. En este caso, del “YogurtLado” y la cantidad .

La clase AñadirProductoCestaAction está preparada para recibir los campos detallados en el formulario (id y cantidad) con los que se realizará su función, que es incluir ese producto en la cantidad adecuada a la cesta, como se aprecia en las siguientes imágenes:

```
public class AnadirProductoCestaAction extends ActionSupport

    //Scopes
    private Map session;
    private Map<String, Object> request;
    // Acceso a datos
    PedidoDAO pedidoDAO;
    ProductoDAO productoDAO;
    private String id, cantidad;
    private String alerta;

    public void setId(String id) {
        this.id = id;
    }

    public void setCantidad(String cantidad) {
        this.cantidad = cantidad;
    }
}
```

Ilustración 30: Action que recibe los datos del formulario

4. CONCLUSIONES

4.1. Presupuesto

En este apartado se tratará de exponer de manera aproximada el presupuesto que se ofrecería a un cliente interesado en comprar o adquirir el software desarrollado y su documentación.

Para ello, se desglosarán los costes de producción en dos apartados: costes de personal y costes de material. Una vez detallados todos estos costes, se realizará la suma de todos ellos para obtener el presupuesto final.

(a) Costes Personal

Dentro del siguiente enlace: <http://www.boe.es/boe/dias/2012/03/28/pdfs/BOE-A-2012-4284.pdf>

Se obtiene la siguiente tabla de niveles salariales 2012:

Nivel	Mes x 14	Anual
Nivel 1	1.673,63	23.430,82
Nivel 2	1.243,21	17.404,94
Nivel 3	1.198,81	16.783,34
Nivel 4	1.099,08	15.387,12
Nivel 5	960,55	13.447,70
Nivel 6	827,55	11.585,70
Nivel 7	799,80	11.197,20
Nivel 8	744,42	10.421,88
Nivel 9	692,70	9.697,80
Nivel 10	492,49	6.894,86

El nivel 1 corresponde a licenciados y titulados (2º y 3.er ciclo universitario), el nivel 2 a diplomados y titulados (1.er ciclo universitario).

Se adaptará nuestro rango actual al de nivel dos, por tanto: 17.404,94 más el plus del convenio (2092,95), hacen un salario anual de 19497.89€.

Para el establecimiento de las horas anuales se usará el siguiente cálculo:

Tomando como referencia 8 horas semanales: 8 horas x 5 días x 4 semanas x 12 meses

= 1920 horas totales.

En Madrid hay 14 días festivos = 14 días x 8 horas = 112 horas no trabajadas

Horas sin vacaciones = 1920 – 112 = 1808, aproximadamente 1800 horas

Para la cotización a la Seguridad Social:

En este caso, se trata del 29,90% en concepto de Contingencias Comunes (23,6%), Desempleo (5,5%), Fondo de Garantía Salarial (0,2%) y Formación Profesional (0,6%)

Salario Anual	Cotización seguridad social	Coste Total Empresa	Coste por hora
19497.89	29.9	25327.76	14.07

Cometido	Horas dedicadas	Coste/hora	Total
Análisis de la aplicación	150	14.07	2110,5
Diseño de la base de datos	50	14.07	703,5
Programación funcionalidades	120	14.07	1688,4
Diseño de interfaces	60	14.07	844,2
Documentación	50	14.07	703,5
Total	430	14.07	6050,1

(b) Costes Material

Material	Coste
Portátil Toshiba Satellite L850-1RX Core I5 3230M 2.5GHZ/4GB	650 €
Total	650€

(c) Presupuesto Final

Presupuesto = (costes empresa de personal + costes de materiales) * (1 + porcentaje de beneficio en tanto por uno).

Beneficio a obtener: 55% = 0,55

Presupuesto: $(6050,1 + 650€) * (1 + 0,55) =$
 $6700,1 * 1.55 = 10385,155 €$

4.2. Conclusión y líneas futuras

Hoy en día vivimos en el mundo de la web, acceder a Internet desde nuestros ordenadores, tablets o smartphones es algo cotidiano. Por ello, el desarrollo de aplicaciones web que puedan ser compatibles con todo tipo de dispositivos está en pleno auge.

El desarrollo web tiene muchas ventajas que son explotadas a su favor como no requerir de software especial para los clientes, información centralizada, copias de seguridad...

Para realizar aplicaciones web podemos decidarnos por distintos lenguajes de programación, que nos ofrecen diferentes aproximaciones para los problemas que nos pueda plantear el desarrollo de aplicaciones. No hay una solución perfecta ni mejor ni peor; se trata de saber elegir la solución que mejor se adapte a nuestros problemas y esto no siempre es equivalente a la opción más famosa o extendida.

Dentro de estas ideas, para facilitar el desarrollo de aplicaciones y poder realizar las tareas de forma más rápida y sencilla, surgieron los framework.

Los framework hacen que nos centremos en el verdadero problema que se tiene que solucionar, abstrayéndonos de los menos significantes.

Además, proporcionan una buena jerarquía de directorios, ya que sus librerías se encuentran muy probadas. Usarlos nos permite acceder a sus comunidades, las cuales pueden ayudarnos con plugins que nos facilitan ciertas tareas.

Actualmente, hay muchísimos frameworks para el desarrollo web: String, Ruby on Rails, Django, Symfony, Struts2...

Como se ha expuesto a lo largo de este proyecto, cada uno usa un lenguaje de programación y tiene sus características propias. Struts2, gracias a la tecnología de servlets, proporciona una infraestructura básica a la hora de construir aplicaciones web sobre la plataforma Java.

Asimismo, al usar el patrón de diseño modelo-vista-controlador divide la distintas tareas en módulos bien diferenciados, lo que contribuye a que, si se conoce el framework -aun sin conocer nada del proyecto, se sabrá por qué camino buscar.

Todas las características de Struts2, comentadas a lo largo de la memoria, hacen que sea un framework más limpio y flexible. Las acciones, la posibilidad de diferenciar los tipos de resultados, el uso de su característico OGNL, sus etiquetas, interceptores que permiten la interceptación de la petición antes de que se ejecute la lógica de la acción... Todo ello, nos da mucha flexibilidad para poder realizar la lógica del negocio.

Para poder aprovechar cualquier framework, se debe superar la curva de aprendizaje para aprender a usarlo de forma correcta. Al principio, no se notarán las ventajas, pero a medio-largo plazo la rapidez y ventajas, que nos ofrece su uso, hará que no nos arrepintamos del tiempo empleado en aprender a usarlo.

En la actualidad, Struts2 se encuentra en la versión 2.3.15, liberada el 22 de junio del 2013. Esta última versión añade algunas mejoras como la capacidad de poder emplear cookies dentro de los action, una nueva interfaz ValidationAware o algunos bugs menores, entre otras.

Se puede observar que Struts2 se sigue actualizando, dado su gran uso en el mundo empresarial. En un futuro, la comunidad de Struts seguirá solucionando errores y añadiendo mejoras en sus puntos más débiles.

La capacidad de integración de Struts2 con otros framework hacen de él un gran candidato para elegirlo como centro de las aplicaciones web. Además, es posible mejorar su rendimiento con la combinación con otros frameworks como Spring o Hibernate.

Struts2 se puede compaginar fácilmente con Spring para permitir la inyección de objetos en la aplicación desde el fichero de configuración de Spring, lo que aumenta la portabilidad y, sobre todo, la integración de Struts2 con otras APIs.

Resulta relativamente sencillo establecer una arquitectura que emplee Spring como elemento central que permita la comunicación de Struts2 para proporcionar una capa Web y de Hibernate como capa de persistencia. Este tipo de arquitectura se adapta fácilmente a la inclusión de Web Services mediante el empleo de AXIS2. Todas estas APIs, en sus versiones actuales, aportan un modelo basado en POJO's que resulta fácil de integrar en un sistema.

Todas estas características, junto con las mencionadas y descritas a lo largo de este trabajo, sitúan a Struts2 como un framework elegante y extensible, que proporciona muchas facilidades para resolver los problemas típicos del desarrollo web. Por ello, merecerá la pena dedicarle algo de tiempo en nuestra formación, ya que engloba muchos conceptos realmente interesantes y muy útiles para el mundo web.

5. BIBLIOGRAFÍA E INFOGRAFÍA

Bibliografía:

- [1] Brown, Don; Davis, Chad Michael; Stanlick, Scott (2008), *Struts 2 in Action*, Manning.
- [2] Groussard, Thierry (2010), *Java Enterprise Edition. Desarrollo de aplicaciones web con JEE 6*, Ediciones ENI.
- [3] Kogent Learning Inc. Solutions (2008), *Struts 2 Black Book*, Dreamtech Press.

Infografía:

<http://asaes.wordpress.com/2009/06/28/struts-2-framework-mvc/>

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

<http://joeljil.wordpress.com/2010/05/31/struts2/>

<http://struts.apache.org/development/2.x/>

<http://struts.apache.org/release/2.3.x/docs/coding-struts-2-actions.html>

<http://struts.apache.org/release/2.3.x/docs/form-validation.html>

<http://struts.apache.org/release/2.3.x/docs/getting-started.html>

<http://struts.apache.org/release/2.3.x/docs/how-to-create-a-struts-2-web-application.html>

<http://struts.apache.org/release/2.3.x/docs/using-struts-2-tags.html>

<http://todosobreprogramacion.blogspot.com.es/2013/02/conociendo-struts-2-armando-mi-primer.html>

<http://www.kamlov.site90.net/?p=613>

<http://www.programania.net/desarrollo-agil/spring-mvc-vs-struts/>

http://www.slideshare.net/mraible/comparing-jsf-spring-mvc-stripes-struts-2-tapestry-and-wicket-presentation?from_search=2

http://www.slideshare.net/techmi/curso-java-avanzado-1-introduccion-al-desarrollo-web?from_search=2

<http://www.webtutoriales.com/articulos/struts-2>

6. APÉNDICES

6.1. Anexo A: Manual de usuario SuperMarket

El siguiente manual tiene como función explicar el funcionamiento de la aplicación “SuperMarket”, adjunta en el CD del trabajo.

¿Qué es SuperMarket?

SuperMarket es una aplicación básica que simula la web de cualquier supermercado actual. En ella, se muestra mediante una serie de filtros o por el buscador de la barra de navegación los distintos productos que ofrece. Además, estos productos se pueden agregar a la cesta de la compra y procesar el pedido.

Para poder cerrar el pedido, debemos “loguearnos” en la aplicación. En caso de que no se tenga usuario, se podrá registrar sin problema.

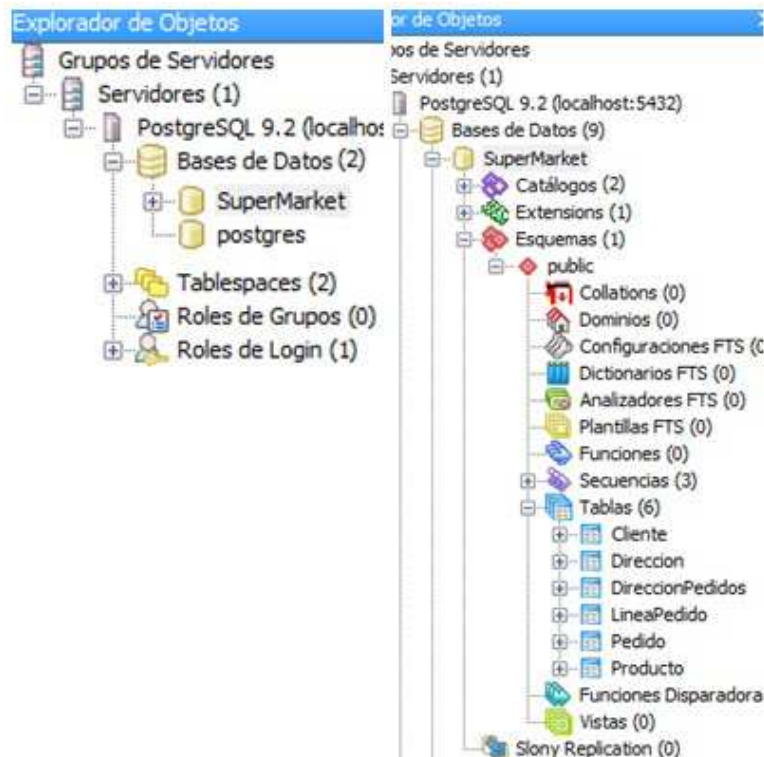
Al margen de los clientes, existen los administradores. Al iniciar sesión les trasladará a su módulo correspondiente dónde podrán gestionar los productos, los pedidos de los usuarios y a los propios usuarios.

Requisitos:

- PostgreSQL 9.1
- Netbeans 7.2 (probado con JDK 1.6)
- Librerías NetBeans: Struts 2.3.4 Core Libraries y PostgreSQL JDBC Driver
 - Las librerías de Struts están en el CD. Para instalarlas, acceda desde Netbeans a Tools/Plugins/Downloaded/Add Plugins y añada los 3 archivos necesarios.
 - Para más información, consulte el [Anexo 2](#) de esta memoria.
- Servidor Apache Tomcat 7.0.27

Pasos previos:

- Se supone que se usará el usuario por defecto “postgres” y la contraseña “password”. Si no es su caso, pase al siguiente apartado.
- Crear en postgresql la base de datos “SuperMarket”. Para ello, abra el archivo Script BBDD.txt de la carpeta scripts del CD y ejecútelo.
- Una vez creada la base de datos, ejecute el segundo script “Script Tablas.txt” para crear las tablas necesarias para la aplicación.



*** Caso especial si no usamos los datos por defecto:

- Cambiar todas las sentencias de los scripts: OWNER postgres -> OWNER "tuUsuario" o OWNER TO postgres -> OWNER TO "tuUsuario" y ejecutarlos.
- En la clase PostgreSQLConnectionManager del paquete src.java.tools deberá cambiar los strings usr y psw por sus datos.

```

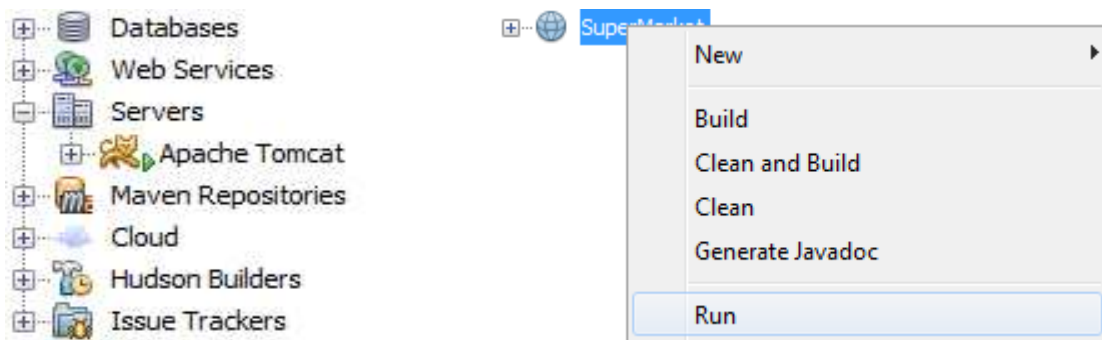
/**
 *
 * @author Jorge
 */
public class PostgreSQLConnectionManager {

    private PostgreSQLConnectionManager() {
    }
    ;

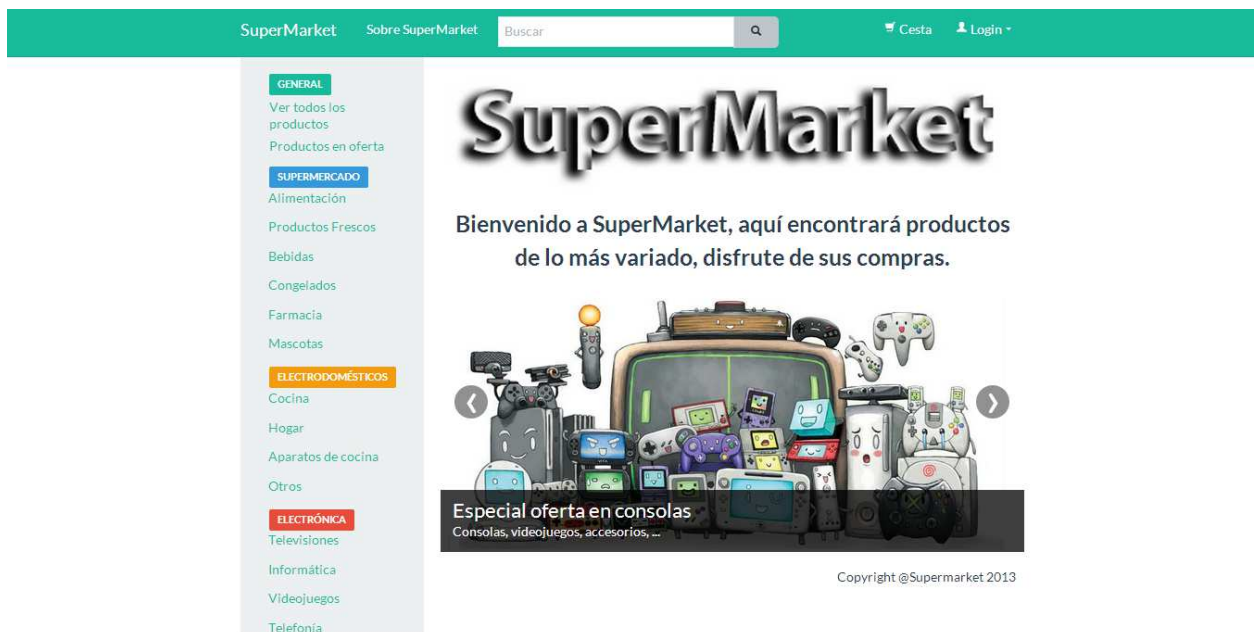
    private static boolean driverLoad = false;
    private static final String pgDriver = "org.postgresql.Driver";
    private static final String pgUrl = "jdbc:postgresql://localhost:5432/SuperMarket";
    private static final String usr = "postgres";
    private static final String psw = "password";
    private static final PostgreSQLConnectionManager INSTANCE = new PostgreSQLConnectionManager();

```

Una vez preparada la base de datos, inicie el servidor Apache Tomcat de NetBeans y arranque la aplicación.



Tras ello aparecerá la pantalla inicial de “SuperMarket”



Desde la pantalla inicial, podemos mirar los productos que ofrece “SuperMarket” sin necesidad de iniciar sesión. Más tarde, cuando se vaya a realizar el pedido, se avisará de la necesidad de login.

Se puede iniciar sesión para no tener que hacerlo más adelante, registrarse en caso de no tener usuario y ver la información sobre “SuperMarket” y su modelo de negocio.

Si se inicia sesión, se observará que se habilita el acceso a los datos de la cuenta y los pedidos.



En los carruseles de la página de inicio, si se pulsa sobre el botón de SuperMarket en Facebook nos llevará desde una nueva pestaña del navegador a la página de Facebook de la empresa (<https://www.facebook.com/supermarketStruts2>)



En la pantalla de registro se encuentran los datos necesarios para realizar los pedidos (dirección de contacto y datos personales). Tras el registro, se iniciará sesión de forma automática.

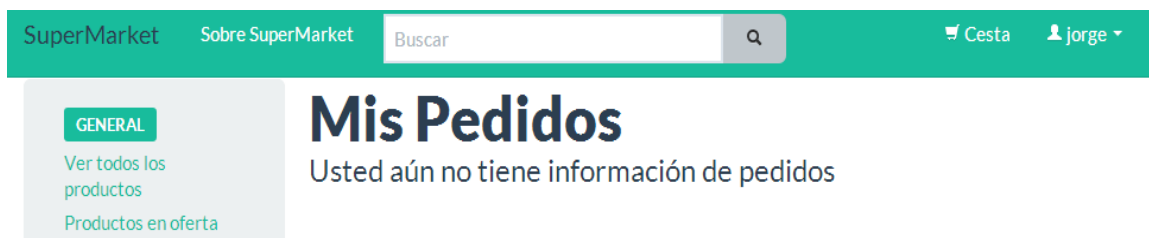
Pulsando sobre “Mi cuenta” se accede a la página de gestión de datos de usuario, donde se podrán cambiar los datos de registro.

The screenshot shows the 'Actualizar Contraseña' (Update Password) page in the SuperMarket user profile. The page has a green header with 'SuperMarket', 'Sobre SuperMarket', a search bar, and a shopping cart icon. The left sidebar contains navigation links: 'GENERAL' (Ver todos los productos, Productos en oferta), 'SUPERMERCADO' (Alimentación, Productos Frescos, Bebidas, Congelados, Farmacia, Mascotas), 'ELECTRODOMÉSTICOS' (Cocina, Hogar, Aparatos de cocina, Otros), and 'ELECTRÓNICA' (Televisiones). The main content area has two tabs: 'Información Básica' and 'Actualizar Contraseña'. The 'Actualizar Contraseña' tab is active, showing a form with fields for 'Nombre de Usuario' (jorge), 'Nombre' (Jorge), 'Apellido' (Lillo), 'Email' (jorge@email.com), 'Número de contacto' (912277111), and 'Tarjeta de Crédito' (1234567). There are also fields for 'Dirección de entrega' (Tipo de vía, Calle, Nombre, Número, Piso, Puerta, Localidad, Provincia, País, Código Postal). A dark blue 'Actualizar Información' button is at the bottom.

Para actualizar la contraseña, se deberá pulsar sobre la pestaña “Actualizar contraseña”. Tras indicar la contraseña antigua se podrá actualizar una nueva.

The screenshot shows the 'Actualizar Contraseña' (Update Password) page in the SuperMarket user profile. The page has a green header with 'SuperMarket', 'Sobre SuperMarket', a search bar, and a shopping cart icon. The left sidebar contains navigation links: 'GENERAL' (Ver todos los productos, Productos en oferta), 'SUPERMERCADO' (Alimentación, Productos Frescos, Bebidas, Congelados, Farmacia, Mascotas), 'ELECTRODOMÉSTICOS' (Cocina, Hogar, Aparatos de cocina, Otros), and 'ELECTRÓNICA' (Televisiones). The main content area has two tabs: 'Información Básica' and 'Actualizar Contraseña'. The 'Actualizar Contraseña' tab is active, showing a form with fields for 'Contraseña Actual' (password masked with dots), 'Nueva Contraseña' (password masked with dots), and 'Repetir Nueva Contraseña' (password masked with dots). A dark blue 'Actualizar' button is at the bottom.

Si se intenta acceder a la cesta de compra o a “Mis pedidos” sin haber realizado ninguno, se obtendrá el siguiente resultado:



Pulsando la opción “Sobre SuperMarket”, se muestra la pantalla de información de la empresa ficticia “SuperMarket”

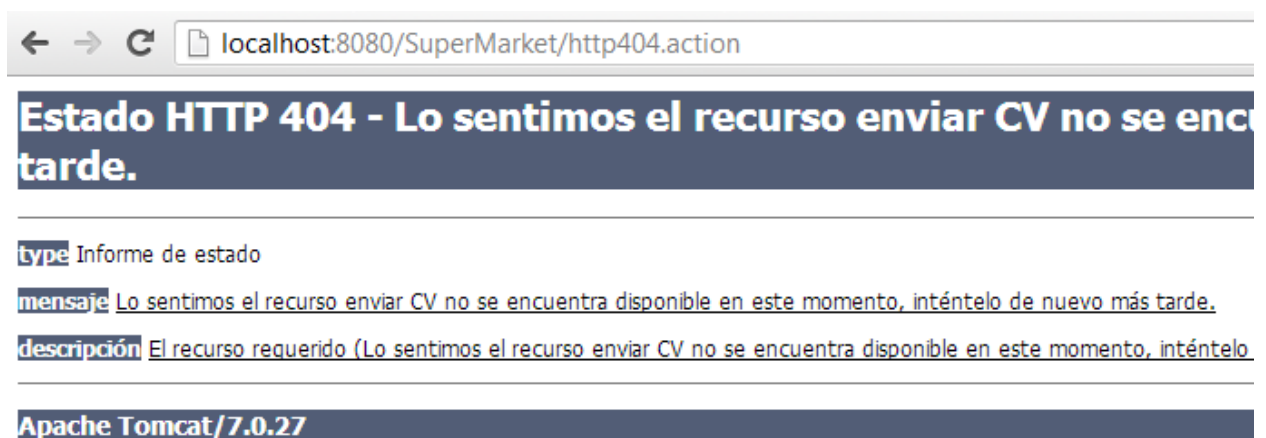


Nuestra comunidad

En SuperMarket somos especialistas en productos frescos, ofreciendo al cliente ese ambiente de mercado tradicional, donde todo el personal se involucra en el servicio al cliente, y donde además, conviven en equilibrio las primeras marcas con nuestras marcas propias. Atentos a las nuevas tecnologías ofrecemos una amplia gama de electrodomésticos y consolas, abarcando el mayor terreno posible para que nuestros clientes queden satisfechos.

Si desea trabajar en nuestras instalaciones por favor mande su CV pulsando en el siguiente enlace [Enviar](#)

Seleccionando en el enlace Enviar, se muestra la siguiente pantalla:

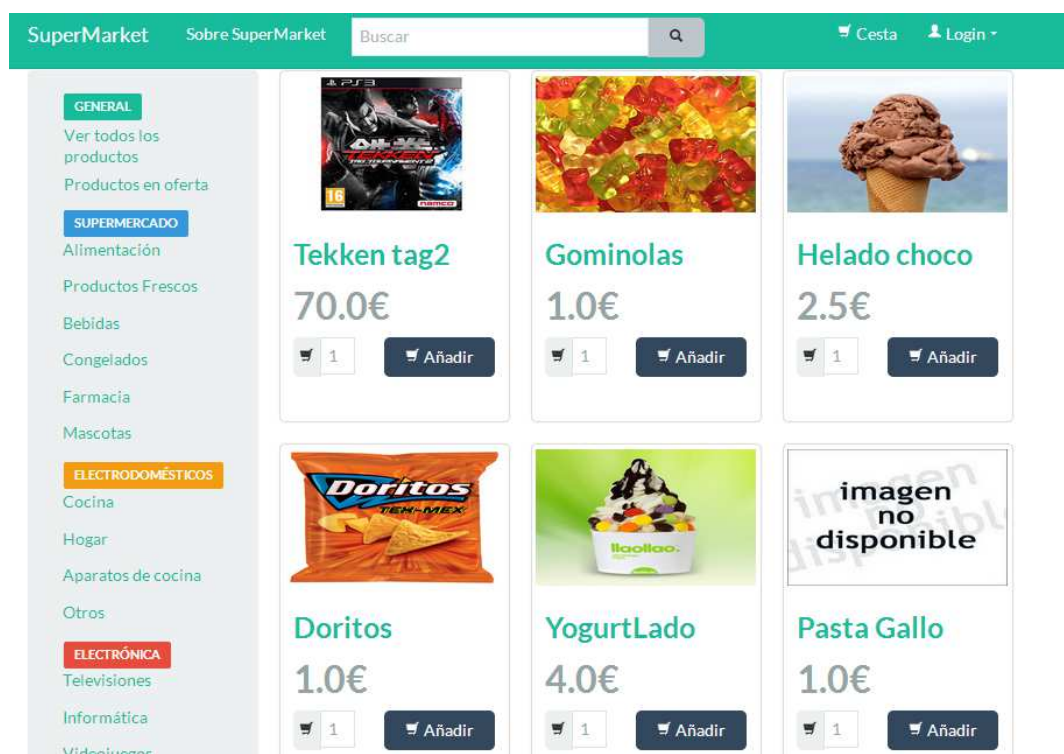


En esta pantalla (realizada para probar el tipo de result httpheader) se muestra información específica a un error previamente declarado, un error más específico que pueda dar información a los clientes que puedan comprender. (Al usuario esta opción no le aporta nada, ya que siempre se muestra el error).

Pulsando sobre el texto “SuperMarket”, de la esquina superior izquierda, volvemos a la pantalla de inicio (esta función se repite a lo largo de las pantallas).

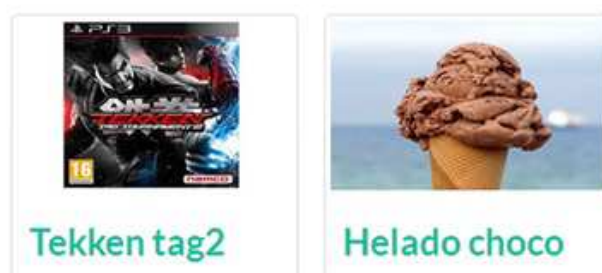
Los filtros de la parte izquierda de la aplicación muestran los diferentes productos de la aplicación, según una serie de criterios. Dependiendo de la opción elegida, se mostrarán unos productos u otros.

Por ejemplo: si buscamos todos:



O los productos en oferta (y destacados):

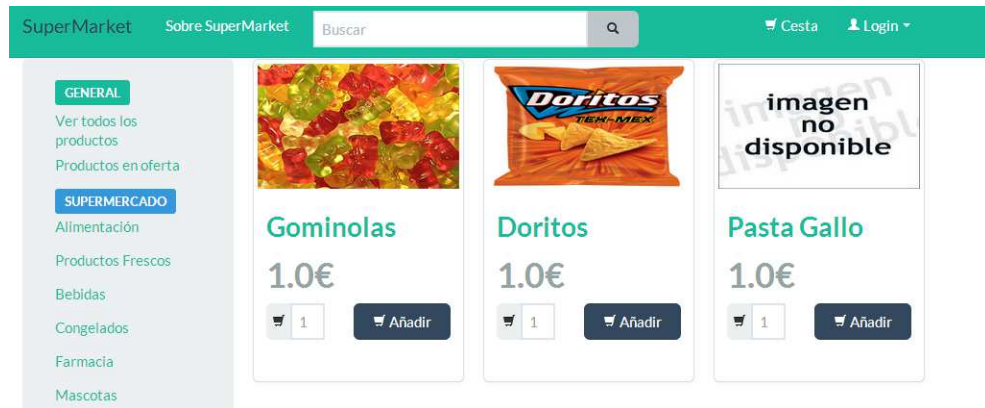
Productos Destacados



Productos en Oferta



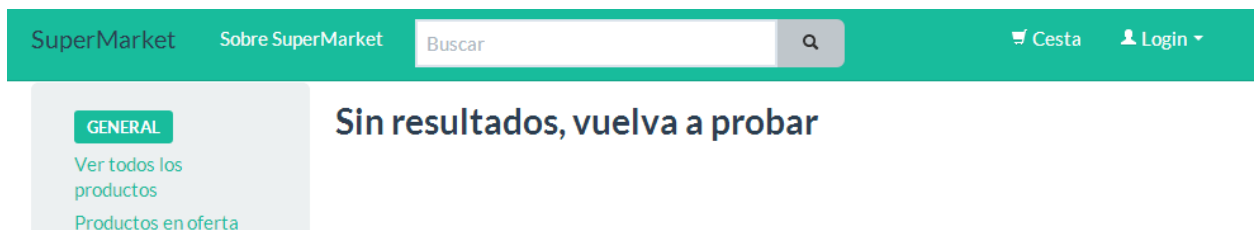
Ver por sección (Ej. Alimentación)



Ver por categoría específica (Ej. Helados)



En lugar de usar los filtros, se puede buscar utilizando el buscador de la barra de navegación. Éste nos mostrará los productos similares a nuestra búsqueda y en caso de que no encuentre ninguno, se verá la siguiente pantalla:



Se ha podido observar en las pantallas anteriores que se muestran los productos con su nombre e imagen (si está disponible), junto con una caja de texto de la cantidad del producto elegido y el botón “Añadir”. Este botón introducirá en la cesta de la compra el elemento seleccionado y la cantidad.

Cesta de la compra

Los productos añadidos a su carrito son:



YogurtLado

Precio Unidad: 4.0€

1

Actualizar

X

Total: 4.0€

Datos de envío

Dirección de entrega ⓘ

Tipo	nombre	numero
piso	puerta	localidad
provincia	codigo postal	
pais		

Actualizar Datos

Tras ver toda la información de su cesta de productos usted puede realizar el pedido.

Para realizar su pedido con éxito pulse sobre 'Realizar Pedido'. Si en caso contrario quiere cancelar su pedido pulse sobre 'Deshacer Pedido'

Precio Final del pedido:

4.0 €

Realizar Pedido

Deshacer Pedido

En esta pantalla se muestran los productos seleccionados para comprar. Además, se puede actualizar la cantidad, así como eliminar los productos que se quiera.

Si no se está “logueado”, al pulsar sobre “Realizar pedido”, dará un error y se solicitará que inicie sesión.

SuperMarket

Sobre SuperMarket

Buscar

Cesta 1

Login

GENERAL

Ver todos los productos

Productos en oferta

SUPERMERCADO

Alimentación

Productos Frescos

Bebidas

Congelados

Farmacia

Mascotas

Cesta de la compra

Los productos añadidos a su carrito son:

YogurtLado

Precio Unidad: 4.0€

1

Actualizar

X

Total: 4.0€

Fallo! Conéctese para realizar un pedido

83

Una vez “logueado”, se tomará como dirección de envío la suministrada por el usuario al registrarse y se cargarán los datos en las cajas de texto del apartado “Dirección”. Si no se desea que se envíe a esa dirección, esta se podrá actualizar pero solamente se cambiará en esa sesión; si se cierra y se vuelve a iniciar la sesión, se volverá a cargar la dirección del perfil.

erMarket

Buscar


Q

Cesta 2

jorge

Cesta de la compra

Los productos añadidos a su carrito son:




YogurtLado

Precio Unidad: 4.0€

1

Actualizar

Total: 4.0€



Gominolas

Precio Unidad: 1.0€

3

Actualizar

Total: 3.0€

Datos de envío

Dirección de entrega

calle

rincon de la huerta

2

3

1

Coslada

Madrid

28821

España

Actualizar Datos

Tras ver toda la información de su cesta de productos usted puede realizar el pedido.

Para realizar su pedido con éxito pulse sobre 'Realizar Pedido'. Si en caso contrario quiere cancelar su pedido pulse sobre 'Deshacer Pedido'

Precio Final del pedido:

7.0 €

Realizar Pedido

Deshacer Pedido

Si confirmamos el pedido, veremos la pantalla de resultados con la de dicho pedido.

Su pedido ha sido recibido...

El identificador del pedido es:

1

La dirección donde se enviará el pedido realizado es:

calle rincon de la huerta Nº 2 3º 1
Coslada (28821)
Madrid España

El importe total del pedido es: **7.0€**

Mis Pedidos

Inicio

Una vez realizados los pedidos, si se accede al apartado “Mis pedidos” se mostrará la lista de pedidos.

Mis Pedidos

Fecha de Creación	Precio Total	Estado del pedido	Acciones
04/07/2013 23:30	77.5€	Sin Enviar	⚙ Detalles
04/07/2013 23:29	7.0€	En proceso	⚙ Detalles
04/07/2013 23:31	10.0€	Entregado	⚙ Detalles

En “Detalles” se accederá a los detalles del pedido: estado, precio, fecha, productos, dirección de envío...

Mi Pedido

IDPedido 2	Fecha de Inicio 04/07/2013 23:30	Precio 77.5€	Estado del pedido Sin Enviar
Dirección de entrega avenida falsa Nº 1 1º 3 Coslada (28821) Madrid España			

Detalles de pedido

Nombre	Precio	Cantidad	Total
Tekken tag2	70.0€	1	70.0€
Helado choco	2.5€	3	7.5€

En los “Detalles del pedido” se pueden ver los productos comprados en ese pedido. Tras pulsar en los hipervínculos de los nombres de los productos, se podrá visualizar los detalles del producto seleccionado.



Tekken tag2

70.0€

 1

Comprar

ID

25456

Sección

Videojuegos

Categoría

PS3

País

Japón

Oferta

NO

Destacado

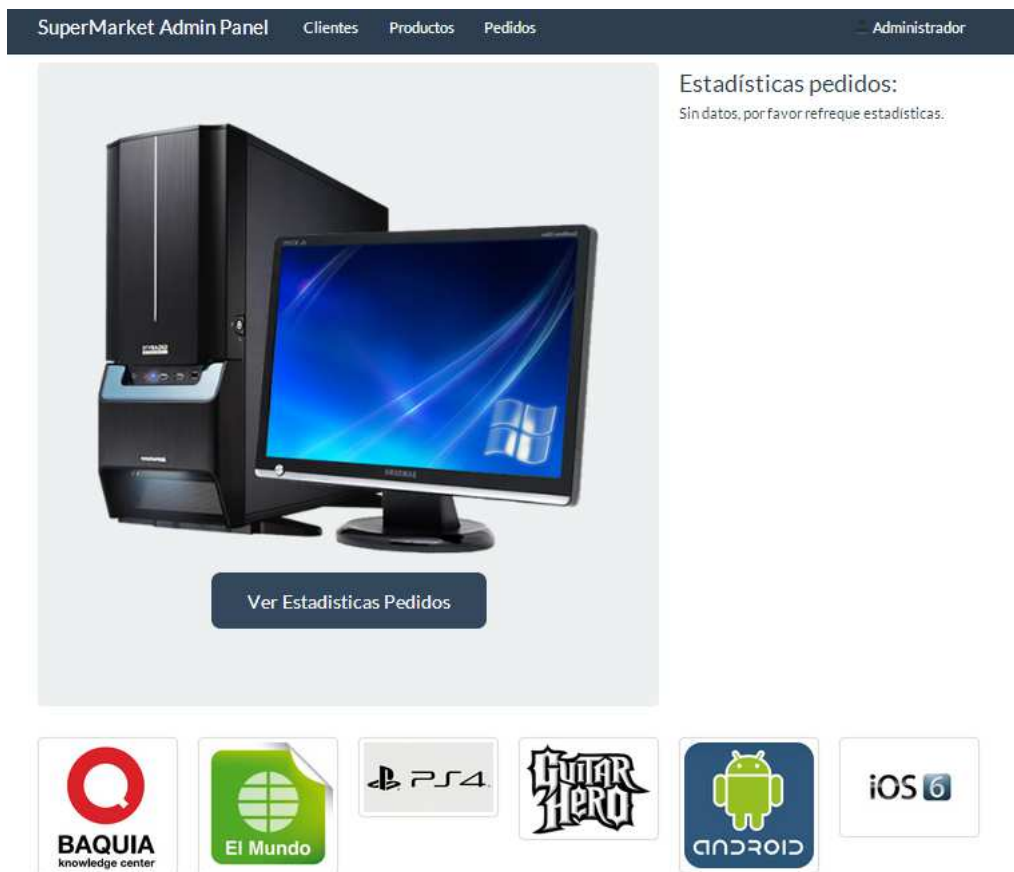
SI

Descripción del Producto

Edición especial we are tekken edition

Con todo ello, concluye el módulo de clientes. Avanzamos, por tanto, al módulo de los administradores dentro del Script Tablas, el cual crea las tablas de la base de datos e introduce además un administrador por defecto.

El usuario es “admin” y la contraseña “admin”. Si iniciamos sesión como este usuario, al tener privilegios de administración, no tendrá acceso a las pantallas comentadas anteriormente, sino que accederá al módulo de administradores.

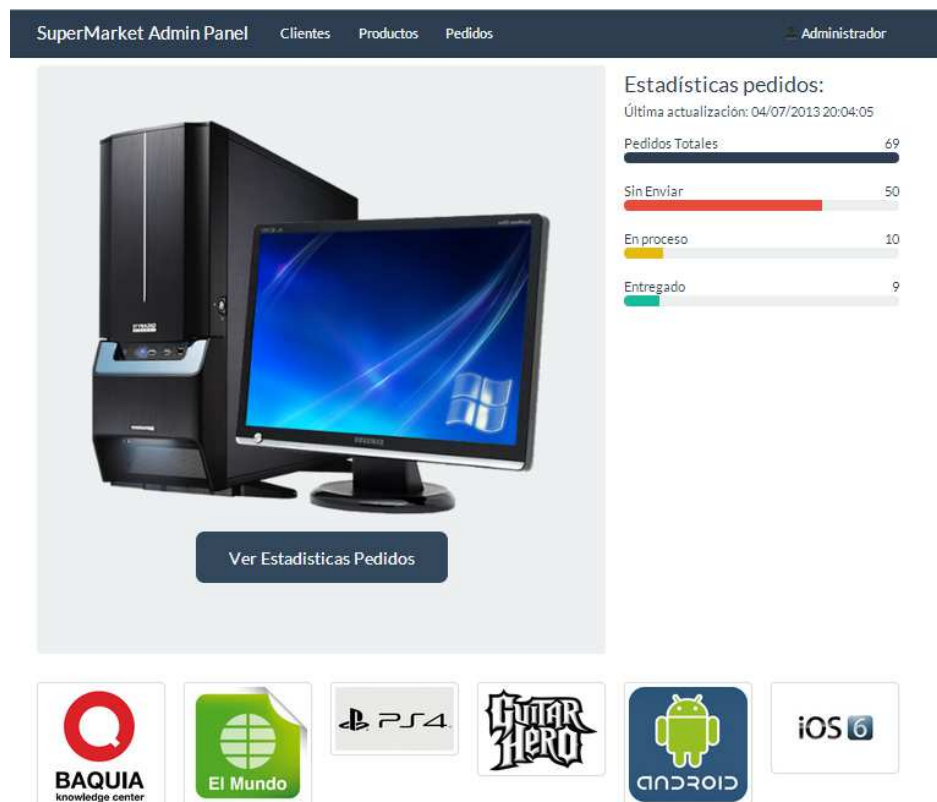


Las funciones disponibles del administrador son:

- Ver estadísticas de los pedidos.
- Gestionar clientes.
- Gestionar productos.
- Gestionar pedidos.

En la parte inferior de la página se observan imágenes de las empresas asociadas a “SuperMarket”

El botón “Ver estadísticas pedidos” recarga las estadísticas globales de la aplicación, en donde se observan los pedidos totales, los que están sin enviar, los que todavía están en proceso de entrega y los entregados.



El módulo de clientes muestra los usuarios de la aplicación. Se puede eliminar usuario, banearlo, desbanearlo o crear nuevos administradores.

SuperMarket Admin Panel Clientes Productos Pedidos Administrador

En esta tabla podrá visualizar todos los Usuarios de la aplicación. Existen dos tipos de usuarios, los clientes registrados en la aplicación y los administradores

En el panel derecho podrá crear nuevos usuarios administradores

Listado de Usuarios

Username	Nombre	Apellido	Email	Privilegios	Acciones
admin	-	-	admin@email.com	Admin	
admin2	-	-	admin2@email.com	Admin	
user1	User1	Apellido1	user1@email.com	Cliente	
jorge	Jorge	Lillo	jorge@email.com	Cliente	
user2	User2	Apellido2	1@email.ca1	Cliente	

Crear Administrador

Username:

Email:

Password:















Repetir Password:

Copyright @Supermarket 2013

El módulo de productos contiene dos zonas diferenciadas:

Por un lado, el listado de productos, donde se muestran todos los productos del sistema y donde se observan sus características básicas. Se puede eliminar o cambiar la visibilidad del producto (si no es visible aunque esté en el sistema a los clientes no les aparecerá) y ver en mayor detalle.

Productos

Nombre	Sección	Categoría	Precio	País	Oferta	Destacado	Acciones
Tekken tag2	Videojuegos	PS3	70.0€	Japón	No	Sí	Detalles  
Gominolas	Alimentación	Chucherías	1.0€	Francia	Sí	No	Detalles  
Helado choco	Congelados	Helados	2.5€	Italia	Sí	Sí	Detalles  
Doritos	Alimentación	Aperitivos	1.0€	España	Sí	No	Detalles  
YogurtLado	Congelados	Helados	4.0€	España	Sí	No	Detalles  
Pasta Gallo	Alimentación	PastaArroz	1.0€	Italia	No	No	Detalles  
Helado V.	Congelados	Helados	1.0€	Holanda	No	No	Detalles  

Los mensajes de eliminar o cambiar visibilidad son iguales en el módulo de clientes.



Y por otro lado, la zona de “Creación de productos nuevos”:

Creación de nuevo Producto

Nombre de Producto

Sección

Televisiones
Informática
Videojuegos
Telefonía

Categoría

Sin Categoría

País

Descripción

Descripción del producto

No se ha seleccionado ningún archivo

Precio

Oferta ☐ Destacado ☐

Visible ☐

Creación de Productos

Recuerde que el precio del producto está en Euros y si tiene decimales debe de expresarlo con punto y no con coma
Ejemplo: 1.2
Formato imágenes: jpg y png, cualquier otra imagen será ignorada.

Tras rellenar los datos, se añadirá el producto al listado de productos. La imagen es opcional. Además, solo se permitirán los formatos jpg y png.

Si se introduce algún carácter no numérico en precio, se mostrará el siguiente error:

Creación de nuevo Producto

Nombre de Producto

Sección

Sin Determinar

Alimentación

Productos Frescos

Bebidas

Categoría

Sin Categoría

País

Descripción

Seleccionar archivo

No se ha seleccionado ningún archivo

Añadir Producto

Creación de Productos

Recuerde que el precio del producto está en Euros y si tiene decimales debe de expresarlo con punto y no con coma

Ejemplo: 1.2

Formato imágenes: jpg y png, cualquier otra imagen será ignorada.

- El precio debe ser un número (Ej. 8.80)

Precio

Oferta ☐

Destacado ☐

Visible ☐

Copyright @Supermarket 2013

Si se pulsa “Detalles de un producto” se accederá a la pantalla de detalles avanzados:

Helado chocolate



Seleccionar archivo

No se ha seleccionado ningún archivo

Detalles del producto

Editar Producto

ID

25455

País de Origen

Italia

Sección

Congelados

Categoría

Helados

Precio

2.0

€

Oferta ☐

Destacado ☒

Visible ☒

Descripción del Producto

Clásico helado de chocolate, esencial en un día caluroso.

Se pueden ver los detalles y también se pueden hacer modificaciones:

Helado chocolate



Seleccionar archivo helado_chocolate.jpg

Detalles del producto

ID

25455

País de Origen

Italia

Sección

Congelados

Categoría

Helados

Precio

2.5

€

Oferta

☒

Destacado

☒

Visible

☒

Editar Producto

Descripción del Producto

Clásico helado de chocolate, esencial en un día caluroso. Ahora en oferta!!

SuperMarket Admin Panel

Cientes

Productos

Pedidos

Administrador

Productos

Correcto! Datos actualizados de forma correcta

Nombre	Sección	Categoría	Precio	País	Oferta	Destacado	Acciones
Helado chocolate	Congelados	Helados	2.5€	Italia	Si	Si	<div>Detalles</div> <div><div></div><div></div></div>

Volviendo a entrar, se observan los cambios de forma correcta:

Helado chocolate



Seleccionar archivo No se ha seleccionado ningún archivo

Detalles del producto

ID

25455

País de Origen

Italia

Sección

Congelados

Categoría

Helados

Precio

2.5

€

Oferta

☒

Destacado

☒

Visible

☒

Editar Producto

Descripción del Producto

Clásico helado de chocolate, esencial en un día caluroso. Ahora en oferta!!

Finalmente, en el módulo de “Pedidos”, se observan los pedidos realizados, así como a qué cliente corresponde y su estado.

El administrador puede ver los detalles del pedido y actualizar el estado del mismo cuando corresponda.

Los estados disponibles son:

- Sin enviar: orden recibida.
- En proceso: pedido generado, transportando.
- Entregado: el pedido ha llegado a su destino.

Pedidos de Clientes

Número de Pedido	Cliente	Fecha de Creación	Estado	Precio Total	Acción
4	jorge	05/07/2013 00:27	Sin Enviar	1.0€	⬇ Detalles Cambiar Estado ▾
5	jorge	06/07/2013 00:33	Sin Enviar	2.5€	⬇ Detalles Cambiar Estado ▾
6	jorge	06/07/2013 00:35	Sin Enviar	70.0€	⬇ Detalles Cambiar Estado ▾
7	jorge	06/07/2013 00:35	Sin Enviar	2.5€	⬇ Detalles Cambiar Estado ▾
8	user1	06/07/2013 01:14	Entregado	2.5€	⬇ Detalles Cambiar Estado ▾
1	jorge	04/07/2013 23:29	Entregado	7.0€	⬇ Detalles Cambiar Estado ▾
2	jorge	04/07/2013 23:30	Entregado	77.5€	⬇ Detalles Cambiar Estado ▾
3	jorge	04/07/2013 23:31	En proceso	10.0€	⬇ Detalles Cambiar Estado ▾
9	user1	06/07/2013 01:14	En proceso	12.0€	⬇ Detalles Cambiar Estado ▾
10	user1	06/07/2013 01:15	Sin Enviar	70.0€	⬇ Detalles Cambiar Estado ▾

Copyright @Supermarket 2013

Seleccionando “Cambiar Estado de un Pedido”, se inicia el diálogo para seleccionar el nuevo estado. Por ejemplo, si seleccionamos el pedido 4:

Pedidos de Clientes

Número de Pedido	Cliente	Fecha de Creación	Estado	Precio Total	Acción
4	jorge	05/07/2013 00:27	Sin Enviar	1.0€	⬇ Detalles Cambiar Estado ▾
5	jorge	06/07/2013 00:33	Sin Enviar	2.5€	⬇ Detalles Cambiar Estado ▾
6	jorge	06/07/2013 00:35	Sin Enviar	70.0€	⬇ Detalles Cambiar Estado ▾

Y si seleccionamos el estado 2 Entregado, se actualiza el nuevo estado a:

4	jorge	05/07/2013 00:27	Entregado	10€	Detalles	Cambiar Estado ▾
---	-------	------------------	-----------	-----	--------------------------	----------------------------------

Para acceder a mayor detalle en “Pedido” se debe pulsar el botón “Detalles”:

Información de Pedido

IDPedido	Precio	Dirección de entrega
1	7.0€	calle rincon de la huerta nº 2 3º 1 Coslada Madrid (28821) España
Fecha de Inicio	Estado del pedido	
04/07/2013 23:29	En proceso	

Lista de Productos adquiridos

Nombre	Precio	Cantidad	Total
YogurtLado	4.0€	1	4.0€
Gominolas	1.0€	3	3.0€

Copyright @Supermarket 2013

Al igual que en el módulo de clientes, observamos todos los atributos del pedido así como los productos asociados a él.

Durante la ejecución de la aplicación, si sobrepasamos el tiempo máximo de sesión estipulado: 10 minutos en el archivo web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0" xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema
  <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>
  <session-config>
    <session-timeout>
      10
    </session-timeout>
  </session-config>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

El interceptor definido nos llevará a una página de error, evitando que nuestras peticiones lleguen a los actions de struts, así no permitimos situaciones críticas como perder la sesión mientras se está procesando el pedido, o cualquier otra acción importante del programa.

La pantalla de error es la siguiente:



Timeout exception, su sesión a caducado

[Volver a inicio](#)

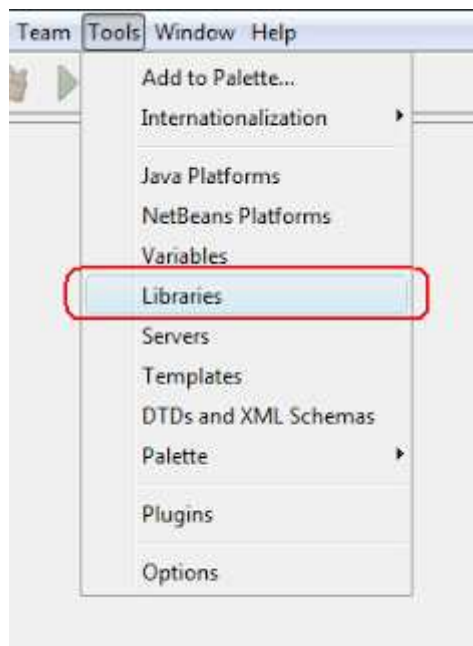
Con todo esto, se ha mostrado un repaso a toda la funcionalidad disponible en “SuperMarket”, con el fin de ayudar al usuario básico, sin conocimientos previos, qué puede ofrecer la aplicación.

6.2. Anexo B: Preparación para el uso de Struts2 con Netbeans 7.2

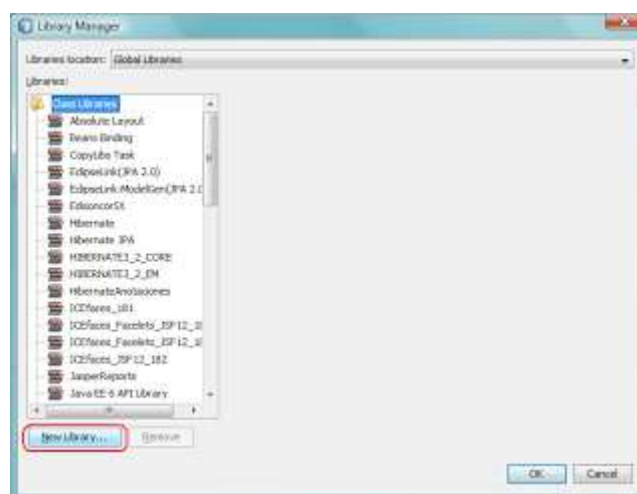
- Opción 1 : Instalación de los plugins adjuntos en el CD de este trabajo,
- Opcion 2: Creación manual de las librerías de Struts2:

Bajar del sitio web de Struts2: <http://struts.apache.org/download.cgi#struts231-SNAPSHOT> las librerías necesarias (struts-2.3.15-lib.zip)

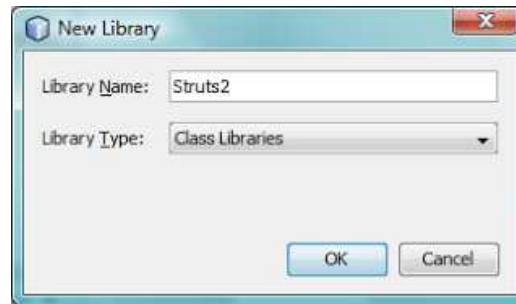
Una vez bajado el archivo, en **NetBeans**, nos dirigimos al menú "**Tools -> Libraries**".



En la ventana que se abre presionamos el botón "New Library...":



En esta nueva ventana, colocamos como nombre de la biblioteca "**Struts2**" y, como tipo, dejamos "**Class Libraries**":



Ahora que tenemos nuestra biblioteca, presionamos el botón "**Add Jar/Folder**" para agregar los nuevos archivos que conformarán la biblioteca:



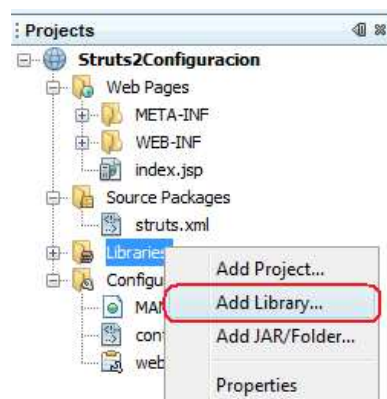
Agregamos los siguientes archivos:

- commons-fileupload-1.2.2.jar
- commons-io-2.0.1.jar
- commons-lang-2.5.jar
- freemarker-2.3.16.jar
- javassist-3.11.0.GA.jar
- ognl-3.0.1.jar
- struts2-core-2.2.3.jar
- xwork-core-2.2.3.jar

Adicionalmente, para poder ejecutar anotaciones en Struts2, es necesario crear una biblioteca adicional que, en este caso, se llamará "**Struts2Anotaciones**". Esta biblioteca debe incluir los siguientes jars:

- **asm-3.1.jar**
- **asm-commons-3.1.jar**
- **commons-fileupload-1.2.2.jar**
- **commons-io-2.0.1.jar**
- **commons-lang-2.5.jar**
- **freemarker-2.3.16.jar**
- **javassist-3.11.0.GA.jar**
- **ognl-3.0.1.jar**
- **struts2-core-2.2.3.jar**
- **xwork-core-2.2.3.jar**
- **struts2-convention-plugin-2.2.3.jar**

A nuestro proyecto web agregamos las bibliotecas creadas. Para esto hacemos clic derecho en el nodo "**Libraries**" del panel de proyectos. En el menú que aparece seleccionamos la opción "**Add Library...**":



En la ventana que aparece seleccionamos las bibliotecas creadas y presionamos "**Add Library**". Con esto, ya tendremos los jars de **Struts 2** en nuestro proyecto.

