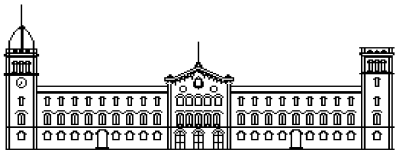


Trabajo final de carrera

INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMES

Facultad de Matemáticas
Universidad de Barcelona

IMPLEMENTACIÓN DE UNA APLICACIÓN WEB UTILIZANDO FRAMEWORKS J2EE

(STRUTS2 – SPRING – HIBERNATE)

Ángel Gómez García

Director: Jesús Cerquides Bueno
Realizado a: Departamento de Matemáticas Aplicada i Análisis. UB
Barcelona, 25 de septiembre de 2008

INDICE

Capitulo 1. INTRODUCCIÓN

Esta introducción quiere acercar al lector los objetivos y contenidos del proyecto, guiándolo por lo que serán las distintas etapas de creación del mismo.

1.1. Descripción general del proyecto

El propósito de este proyecto es la implementación de una aplicación web J2EE. Se estudiará el Framework Struts2 como Framework MVC principal y Spring e Hibernate como Frameworks secundarios para los servicios de negocio y persistencia respectivamente.

En cuanto a la aplicación web, ésta aportará diferentes servicios a los usuarios, entre los cuales destaca, el registro de usuarios y la posibilidad de colaborar con fotos de emplazamientos geográficos, para poder compartir con el resto de la comunidad. La aplicación web, tratará de acercarse al concepto web 2.0 mediante el uso de tecnologías específicas, como pueden ser los RSS feed, AJAX, Google Maps API entre otros.

1.2. Organización de la memoria

El proyecto se ha estructurado en tres grandes partes bien diferenciadas: una destinada a estudiar el funcionamiento del Framework Struts2 y como se integra con diferentes tecnologías, la segunda trata de como se ha diseñado e implementado la aplicación que usará todas estas tecnologías y por último las conclusiones a las cuales se ha llegado, además de información extra aportada mediante Anexos.

La primera parte del proyecto trata de explicar el significado del WEB2.0, y así, entender mejor las características que tendrá nuestra aplicación web. Se hablará sobre el uso de J2EE como tecnología de desarrollo, por último se analizará en detalle el funcionamiento del Framework MVC Struts2, realizando algunas comparativas con su versión anterior y otros Frameworks del mercado. También hablaremos sobre la integración de otros Frameworks a la aplicación como es el caso de Spring e Hibernate.

La segunda parte del proyecto se centra en el diseño e implementación de la aplicación Web VisualGate, donde se mostrarán los requisitos del sistema, planos del proyecto y herramientas utilizadas, además de una descripción detallada de cómo Struts2 nos ha ido ayudando en el desarrollo de la aplicación web. También se realizará un análisis previo sobre la aplicación web, metodología usada y un estudio de costes económicos.

La parte final del proyecto está compuesta por las conclusiones y resultados que se han obtenido en la realización del estudio e implementación de la aplicación web VisualGate.

También se reflejan las dificultades que han ido apareciendo a lo largo del proyecto y las alternativas de mejora para una posible continuación del proyecto en el futuro.

También he añadido las referencias bibliográficas hechas a lo largo del proyecto junto con los Anexos, donde se explica mucho más en detalle algunas características del Framework Struts2 como los interceptores, su funcionamiento y configuración en la aplicación web, también se habla en más detalle sobre Hibernate, Spring y patrones de diseño DAO para el acceso a la capa de persistencia de datos.

1.3.Motivación del proyecto

El mercado actual de las aplicaciones Web no deja de sorprendernos, desde que la primera especificación del Servlet se publicara en 1997, surgieron y siguen apareciendo cada vez más nuevas herramientas que hace que el desarrollo web como la construcción y el diseño sean más fáciles y rápidos de desarrollar. Apache Struts fue lanzado en Mayo del 2000 por Craig McChanahan, técnicamente fue un paso evolutivo en el desarrollo web, pero más importante aún fue que llegó en el momento adecuado, Struts aportaba facilidad en la reutilización y mantenibilidad del código y eso lo convirtió en un estándar para el desarrollo web durante varios años.

Struts2 es la próxima generación de Apache Struts, cuya aparición sale de la necesidad de evolucionar el viejo Struts. La meta marcada de Struts2 es simple, hacer fácil el desarrollo web para el desarrollador, para ello Struts2 provee características para reducir la configuración XML mediante configuraciones por defecto, utilizando anotaciones o usando convenciones en la configuración.

Las Actions ahora son POJO´s, las cuales incrementan la testeabilidad y reducen el acoplamiento al Framework, y los datos introducidos en los campos de texto de los formularios son convertidos automáticamente para que las Actions puedan usarlos sin problemas.

Struts2 disminuye el acoplamiento al Framework haciendo la aplicación más modular y eso ha dado lugar a la aparición de una de las mejores ventajas de Struts2, los interceptores que proveen procesamiento antes y después de las acciones.

La modularidad es uno de los objetivos más buscados en este nuevo Framework, por eso Struts2 aporta, capacidad de añadir nuevos Plugins, hacer que las clases del Framework puedan ser reemplazadas por implementaciones personalizadas, Tags que pueden utilizar una variedad de diferentes temas de renderizado para la vista, incluyendo temas personalizados. También dispone de gran cantidad de “Result Types” para la generación de la respuesta de la petición. Y por último la inyección de dependencias y persistencia que son subministradas por los Frameworks de Spring e Hibernate, los cuales podremos incorporar a Struts2 fácilmente ya que también destaca por su fácil integración con otras tecnologías, mediante Plugins.

La intención del proyecto es poder demostrar como Struts2 puede facilitarnos la creación de una aplicación web, desde su construcción, pasando por su desarrollo y su posterior mantenimiento.

Para ello se ha decidido implementar una aplicación web de gestión de Fotos, al estilo Flickr, donde los usuarios puedan añadir y compartir sus fotos. Este tipo de aplicación usará gran parte de la tecnología de Struts2 y su integración con otras como Spring e Hibernate, también se le sacará partido al Web 2.0 mediante Mashups, Rss para la sindicación y el uso del AJAX.

1.4.Definición y objetivos del proyecto

Uno de los objetivos del proyecto es el de estudiar en detalle el Framework Struts2, ver cómo ha evolucionado desde su anterior versión Struts y poder compararlo con diversos Frameworks del mercado actual.

Tras este estudio se quiere demostrar las ventajas de Struts2 mediante una aplicación web. La creación de la aplicación web, pretende ser lo más fiel posible a la de un diseño real empresarial, pues contará con planos UML, diseño e implementación y una serie de pruebas.

Dentro del proyecto se fomentará el uso de tecnologías actuales y aplicaciones de libre distribución como son el uso del XML, HTML con hojas de estilo CSS, JavaScript , AJAX, bases de datos MySql, además del uso de Tomcat como contenedor de Servlets para desplegar nuestra aplicación web. Finalmente para el desarrollo y despliegue de la aplicación web se usará Eclipse.

Intentaremos acercar algunos conceptos de la tendencia Web 2.0 a la aplicación mediante el uso de AJAX, Mashups, Google Maps y RSS para la sindicación de contenido.

Por otro lado, se usará los Frameworks de Hibernate para la persistencia de datos y Spring para proveer servicios de negocio e inyección de dependencias.

Así que, de manera resumida, podríamos definir los objetivos del proyecto como:

- Implementar y diseñar una aplicación Web usando Struts2 como principal Framework Web.
- Se estudiará y analizará el Framework MVC Struts2 y se comparará con diversos Frameworks del mercado actual.
- Se usará el Framework de Spring para la inyección de dependencias.
- Se usará el Framework de Hibernate para la persistencia de datos.
- Acercarse lo máximo posible al concepto de Web2.0. usando AJAX, Mashups, RSS y la API de Google Maps.
- Fomentar el uso de tecnologías actuales y aplicaciones de libre distribución.

Capitulo 2. ANÁLISIS DE LAS PRINCIPALES TECNOLOGIAS

En este capítulo, trataremos de explicar las principales tecnologías que se han utilizando dentro del proyecto. Empezaremos comentando la tendencia Web 2.0 y que tecnologías están desarrolladas dentro de este concepto, ya que VisualGate tratará de usar diferentes características como pueden ser los RSS feed para la sindicación de contenidos o la utilización de llamadas asíncronas al servidor, para aumentar la interacción con el usuario.

Lo siguiente, será comentar brevemente la tecnología J2EE para el desarrollo del proyecto, remarcando algunas de sus características principales, que nos servirá de introducción al mundo de los Frameworks J2EE. De esta manera, tendremos una idea de lo que son y que representan, además de comentar las diferentes opciones que existen en el mercado, mediante una comparativa que se incluye como Anexo.

Una vez visto lo que es un Framework y que puede hacer por nosotros como desarrolladores web, entraré en detalle en el estudio de Struts2, como principal Framework del proyecto, estudiando las interacciones que existen entre los diferentes componentes del núcleo y que responsabilidades y funciones poseen. También comentaré la integración de otras tecnologías como pueden ser Spring e Hibernate sobre Struts2.

2.1.Web 2.0

Hace un tiempo que se ha pasado de disponer de páginas Web estáticas a páginas dinámicas donde el usuario puede llegar a interactuar con el entorno Web. Es en este contexto, donde entra a escena un nuevo concepto a la hora de entender Internet.

El Web 2.0 se basa en la creación de una web enfocada al usuario y orientada a la interacción como las redes sociales. Es decir, los sitios web 2.0 actúan como puntos de encuentro o páginas webs que dependen directamente de los usuarios. Lo que se pretende es que sea el usuario el que cree contenidos web gracias a la utilización de los servicios de las páginas. Por lo tanto, los sitios Web dejan de tener sentido sin usuarios que exploten los servicios que éste ofrece, ahora los usuarios pasan de tener una actividad pasiva a activa, donde pueden participar aportando contenidos o recursos a la aplicación web.

Desde la perspectiva del programador, ***Web 2.0 es sinónimo de AJAX***. El término de AJAX (Asynchronous JavaScript and XML) fue acuñado en febrero del 2005 por Jesse James Garret y es usado para describir la interacción entre varias tecnologías.

El ***núcleo de AJAX*** es el objeto ***XMLHttpRequest***, el cual es suministrado por el browser. Este objeto fue presentado en el navegador Microsoft Internet Explorer 5, aunque en esa época utilizaba otras técnicas como los ***IFRAMES***.

Junto al objeto XMLHttpRequest, las tecnologías que componen ***una interacción AJAX*** son las siguientes:

- HTML/XHTML (Hypertext Markup Language)
  - Usado para presenter información al usuario.
- DOM (Document Object Model)
  - El documento HTML con una estructura orientada a objetos. Al manipular el DOM con JavaScript, ésta puede ser modificada dinámicamente sin recargar el contenido completo de la página.
- CSS (Cascading Style Sheets)
  - Usado para dar formato y estilo al HTML.
- JavaScript
  - Un lenguaje de programación que puede ser incluido dentro de documentos HTML.
- XML (eXtensible Markup Language)
  - Es el formato de los datos que se retornan en la respuesta del seridor cuando el browser le solicita una llamada asíncrona. La respuesta XML es procesada por JavaScript en el browser añadiendo cambios en el HTML mediante DOM.

Recientemente, otro formato de datos llamado ***JSON (JavaScript Object Notation)*** está ganando bastante popularidad. Similar al XML, JSON retorna información que puede ser procesada por el browser usando JavaScript. La principal ventaja que tiene JSON sobre XML, es la sencillez que ofrece a la hora de asignar el valor de un XML a un objeto JavaScript.

2.1.1. Tecnologías basadas en Web 2.0

El Web 2.0 no pretende exigir la utilización de unas determinadas tecnologías para que todas nuestras aplicaciones Web entren en este esquema, sino más bien ***pretende crear una tendencia***. Sin embargo, existen varias tecnologías que están utilizándose actualmente en busca de seguir evolucionado el Web. Algunas tecnologías y directrices que dan vida a un proyecto Web 2.0 son:

- Transformar software de escritorio hacia la plataforma Web.
- Respeto a los estándares del W3C.
- Separación del contenido del diseño con uso de hojas de estilo CSS.
- Sindicación y agregación de contenidos (RSS / ATOM).
- AJAX
- Utilización de redes sociales al manejar usuarios y comunidades.
- Dar control total a los usuarios en el manejo de su información.
- Proveer APIS o XML para que las aplicaciones puedan ser manipuladas por otros.
- Facilitar el posicionamiento con URL´s sencillos.

2.1.2. Web 2.0 dentro del proyecto

Dentro del proyecto utilizaremos tecnologías que siguen la tendencia Web 2.0 como lo son AJAX, CSS, XML, RSS, Mashups & Google Maps API.

Ilustración 2 : RSS

Ilustración 3 : AJAX

Ilustración 1 : CSS

Ilustración 4 : Google Maps Api

2.2.J2EE – Tecnología de desarrollo del proyecto

La decisión de utilizar ***la tecnología J2EE*** como la herramienta de trabajo para la realización del proyecto ha sido porque se trata de una ***tecnología Open Source*** de la cual existen multitud de herramientas gratuitas. Además incluye una gran cantidad de documentación y API´s superiores a las que puedan existir para otras plataformas como .NET y esto es un factor muy importante ya que facilita y agiliza el diseño e implementación del proyecto. También hay que recordar que uno de los objetivos del proyecto es ***fomentar el uso de tecnologías libres*** a nivel empresarial y poder demostrar que se pueden realizar proyectos sin necesidad de tener que invertir en herramientas que en muchos casos pueden resultar extremadamente costosas. También es interesante que una vez acabada la implementación de la aplicación, ésta pudiera ser alojada en cualquier servidor independientemente de la plataforma elegida.

Así que los motivos por los cuales he elegido J2EE como tecnología de desarrollo han sido:

- Conocimiento previo del lenguaje Java.
- Cubre grandes necesidades tecnológicas como páginas dinámicas JSP y lógica de negocio mediante JAVA.
- Interoperable con otras tecnologías como XML, JavaScript, HTML…
- Soporte: API´s, manuales, ejemplos…
- OpenSource y herramientas de desarrollo como lo es Eclipse.
- Muchas utilidades ya creadas y fáciles de integrar.
- Existencia de una gran variedad de Frameworks MVC para el desarrollo de aplicaciones, entre ellos Struts2, que es el objetivo del proyecto.

2.3.Patrón MVC y Frameworks para J2EE

La gran mayoría de Frameworks web J2EE están basados en el patrón MVC, así que vale la pena echar un vistazo y entender mejor el funcionamiento interno de estos Frameworks.

2.3.1. Patrón MVC – Modelo Vista Controlador

El ***patrón MVC*** está indicado especialmente para el diseño de arquitecturas de aplicaciones que requieran de ***una gran interactividad con los usuarios***, como es el caso de aplicaciones Web. Este patrón organiza la aplicación en tres partes bien diferenciadas. Por un lado tenemos el ***Modelo***, el cual representa los datos de la aplicación y sus reglas de negocio, por otro la ***Vista***, compuesta de vistas que representan los formularios de entrada y salida de datos, y finalmente, el C***ontrolador***, encargado de procesar las peticiones entrantes del usuario y controlar el flujo de ejecución del sistema.

El patrón MVC en la programación web J2EE se le conoce como ***arquitectura de modelo 2***. Esta arquitectura consiste en la utilización de Servlets para procesar las peticiones, que estarían contenidos en el Controlador del patrón, y páginas JSP para mostrar la interfaz del usuario que representaría la Vista, y finalmente los famosos JavaBeans ubicados en el modelo.

Este patrón nos ***proporciona una clara separación entre las distintas responsabilidades*** de la aplicación web.

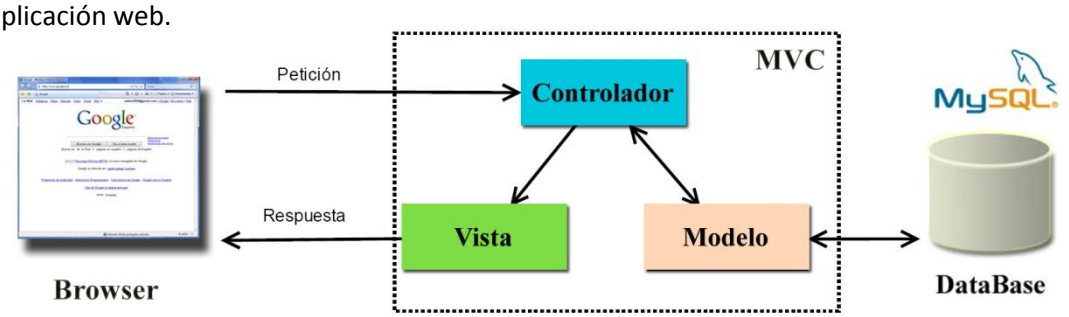


Ilustración 5 **: Patrón Modelo Vista Controlador interactuando con la capa de presentación y datos.**

Controlador	Vista	Modelo
•Todas las peticiones a la capa intermedia que se realicen desde el cliente pasarán por el Controlador, éste determinará las acciones a realizar e invocar al resto de los componentes de la aplicación como pueden ser el modelo o la vista.	•La vista es la encargada de generar las respuestas que deben ser enviadas al cliente. Esta respuesta normalmente incluirá datos generados por el controlador, entonces el contenido de la página no será estático sino que será generado de forma dinámica, y ahí es donde entrarán los JSP.	•Encapsula la lógica de negocio de la aplicación, acceso a los datos y su manipulación.

2.3.1. Frameworks para J2EE

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. Provee una estructura y una metodología de trabajo la cual extiende o utiliza las aplicaciones del dominio. En general, con el término 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.

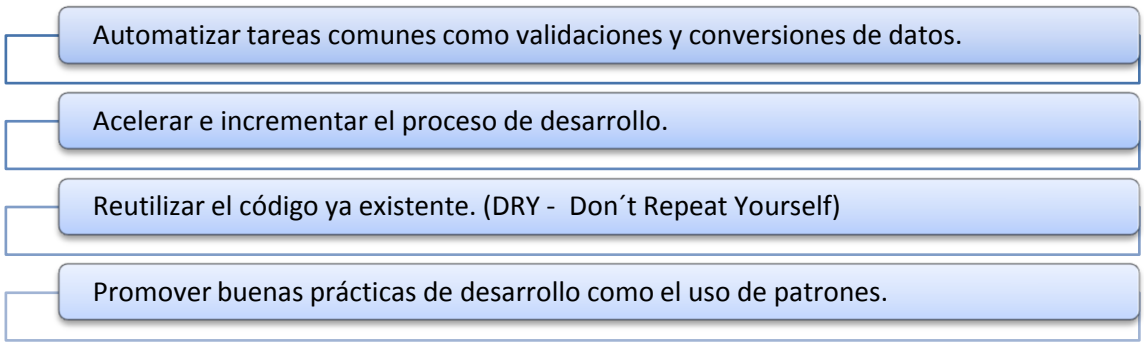


Ilustración 6 : Cualidades que un Framework debería ofrecer.

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

El uso único de un Framework para el desarrollo de una aplicación web no es recomendable, a menos que este Framework nos aporte todas las soluciones que necesite nuestro proyecto. Si éste no es nuestro caso, es recomendable analizar y ver qué otros Frameworks podemos integrar en nuestra aplicación para llegar a solventar estos problemas que no puede solucionar nuestro Framework principal.

El uso de estos Frameworks se irá viendo a lo largo de la memoria, pero podemos avanzar que Struts2 tomará el papel de Framework MVC principal, y el resto que se irán incluyendo como Spring e Hibernate se encargarán de darnos soluciones a problemas que no puede solucionarnos Struts2 o que no lo hace tan bien como estos.

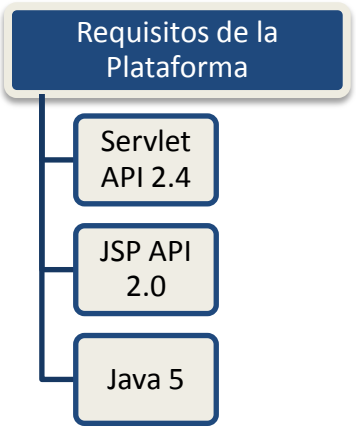
En un principio se había considerado usar JSF (Java Server Faces) para la parte de la Vista de la aplicación, pero visto que Struts2 cumple todas las necesidades del proyecto para esta tarea, como la integración de librerías AJAX por parte de Dojo Toolkit y la famosa Value Stack y su integración con OGNL, JSF fue descartado como Framework para la Vista.

Más sobre Comparativas de Java Web Frameworks:
Ver ANEXO F

2.4.Framework Struts2

Apache Struts2 es un nuevo Framework para desarrollar aplicaciones web. Struts2 no es solo una nueva versión de Struts, es un nuevo y completo Framework basado en el Framework WebWork de OpenSymphony, aunque Struts2 originalmente fue conocido como WebWork2, se quedó con 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, recoger las peticiones por URL y decidir qué Servlet será ejecutado: service(). Es en este contenedor de Servlets, donde desplegaremos nuestra aplicación web VisualGate. En nuestro proyecto usaremos ***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 introducen varias características nuevas que hacen del Framework más limpio y flexible. Estas nuevas características incluyen los famosos interceptores de WebWork2, permitiendo la interceptación de la petición antes de que se ejecute la lógica de la acción, por otro lado aprovecha la aparición de Java5 para incluir configuraciones mediante anotaciones, reduciendo 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 de ser un ***Framework extensible y elegante*** para el desarrollo de aplicaciones web empresariales de cualquier tamaño, está diseñado y creado para agrupar todas las fases de desarrollo de una aplicación.

Actualmente ***Struts2 se encuentra en su versión 2.0.11.2*****,** la cual ha sido usada en este proyecto, aunque también ofrece una versión 2.1.x en fase experimental, de la que cabe destacar la separación de AJAX de Struts2, convirtiéndolo en un Plugin.

2.4.1. Características Principales

Algunas de sus características principales, son las siguientes:

- Todas las clases del Framework están basadas en interfaces y 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.

2.4.2. ¿Por qué usar Struts2 y qué le hace especial?

Hoy en día existen diferentes web Frameworks en el mercado para el desarrollador. Alguno de estos vienen de comunidades Open Source, otros de compañías comerciales y otros son desarrollos internos surgidos por la necesidad de un desarrollo web personalizado.

Con tantas elecciones ahí fuera, ***por qué deberíamos elegir Struts2?*** Éstas son algunas de las características que más le diferencian del resto y que hace Struts2 un Framework muy especial.

- Es un Framework Web basado en "Actions" al igual que su padre, Struts
- Una gran experiencia en el mercado y una comunidad de usuarios brillante.
- Las opciones de configuración son XML y Annotations.
- Las Actions son basadas en POJO, y eso hace fácil su testeo.
- Integración con Spring, SiteMesh y Tiles, entre otros.
- El Gran lenguaje de expresiones OGNL, perfecto para Struts2 y su Value Stack.
- Temas basados en librerías de Tags, entre ellos los Tags de AJAX.
- Múltiples opciones de Vista, entre ellas JSP, FreeMarker, Velocity y XSLT
- Plug-ins para extender y modificar las características del Framework.

2.4.3. Struts vs Struts2

Hoy en día todas las aplicaciones del mercado contienen diferentes versiones, y con cada versión se aumentan las características y ventajas de la versión anterior o almenos ésa es la intención. El Framework Struts no es una excepción, cuando Struts salió al mercado, este se hizo muy popular durante mucho tiempo. Actualmente, Struts se encuentra en su versión 2, y no se parece mucho a su versión anterior de Struts 1.x. Así que, qué mejor comparativa que comparar Struts2 con su predecesor que tantos éxitos obtuvo durante varios años en el mercado de los Frameworks. A continuación se muestra una tabla con las principales diferencias existentes entre los dos Frameworks.

Dependencia con la API de Servlets	
Struts	Las Acciones en Struts tienen dependencia con el Servlet, esto a modo práctico quiere decir que los métodos de la Acción de Struts tienen que recibir varios parámetros, entre ellos el HttpServletRequest y el HttpServletResponse.
Struts2	Esto no ocurre en Struts2, ya que 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, facilidad de testeo de las Actions, entre otras.
Clases Action	
Struts	Uno de los problemas de Struts, es que las Acciones han de extender de una clase base abstracta y no permite interfaces.
Struts2	En cambio en Struts2 no es necesario que se extienda de una clase base, 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	
Struts	Struts soporta validación declarativa proporcionada por “commons-validator” y programática.
Struts2	Struts2 presenta validación declarativa proporcionada por “Xwork Validation Framework” y programática.
Modelo ThreadingAction	
Struts	Las acciones de Struts son singleton y deberían ser Thread-safe, éstas requieren especial cuidado a la hora del desarrollo.
Struts2	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	
Struts	Struts está más acoplado al Framework, por lo tanto dificulta las pruebas.
Struts2	Las acciones de Struts2 como ya hemos 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	
Struts	Struts usa el objeto ActionForm para capturar la entrada de datos de, por ejemplo, un formulario. Todas las ActionForms necesitan ser extendidas por una clase base. Los JavaBeans no pueden ser usados como los ActionForms, por lo tanto los desarrolladores tienen que crear clases redundantes y capturar las entradas, esto viola el principio DRY.
Struts2	Struts2 usa las propiedades de la acción como las propiedades de entrada, eliminando 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 podemos acceder directamente a los objetos del dominio por medio de las Actions, ya que soportan la inyección de dependencias.
Lenguaje de Expresiones	
Struts	Struts está integrado con JSTL y EL, el cual no es muy potente cuando trabaja con colecciones y propiedades indexadas.
Struts2	Struts2 puede usar JSTL, pero el Framework soporta un lenguaje de expresiones más flexible llamado OGNL (Object Graph Navigation Language).
Enlazando valores dentro de la vista	
Struts	Struts usa el método tradicional del JSP para enlazar los objetos en el contexto de la página.
Struts2	Struts2 usa la tecnología Value Stack para que las librerías de Tags accedan a sus valores.
Conversión de tipos	
Struts	Todas las propiedades de Struts son Strings, usa el Commons-beanutils para la conversión.
Struts2	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 y también se puede realizar Converters personalizados.
Control de la ejecución de la Action	
Struts	Todas las acciones en Struts comparten el mismo ciclo de vida.
Struts2	En cambio Struts2, 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.

Más sobre Comparativas de Java Web Frameworks:
Ver ANEXO F

2.4.4. Arquitectura Struts2

En la ilustración 7, se muestra la arquitectura de Struts2 y la relación que tienen los diferentes componentes. A continuación se detallará sus responsabilidades y sus funcionalidades, mediante un ejemplo de petición-respuesta básico.

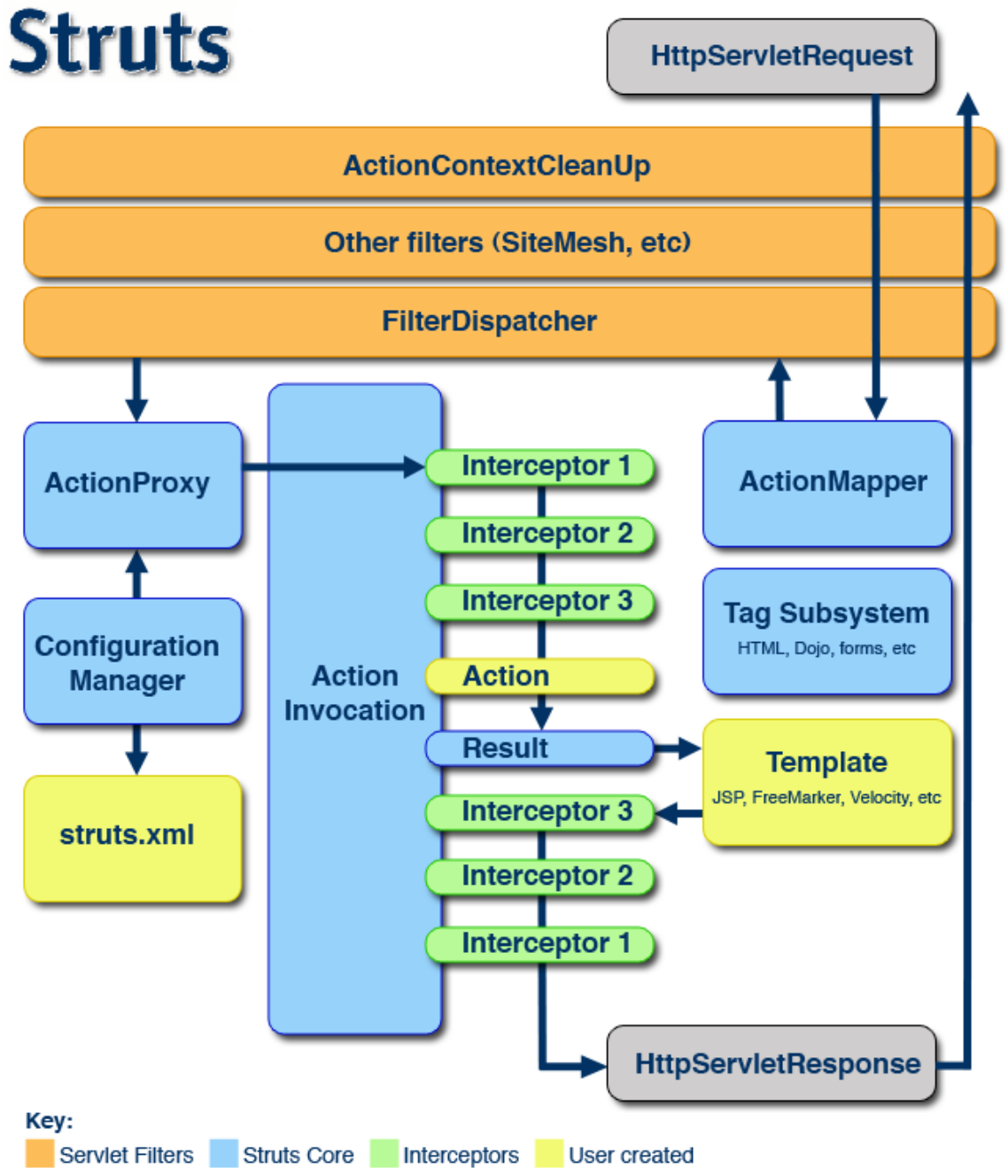


Ilustración 7 : Arquitectura Detallada Struts2

2.4.4.1. Caminando a través de la Petición – Respuesta

- Llega una Petición a la aplicación.
- La Petición es interpretada por el **FilterDispatcher** y determina qué Action y qué conjunto de Interceptors invocará.
- Cada Interceptor ejecuta sus acciones previas a la ejecución del método de Action a invocar.
- Es invocado el método del Action.
- Cada Interceptor ejecuta sus acciones posteriores a la ejecución del método de Action a invocar.
- Se examina el resultado obtenido del Action y se determina el Result correspondiente.
- Mediante el Result determinado se genera la vista, y según la configuración definida sobre él se invoca el proceso de generación de la vista.
- La vista generada retorna al cliente en forma de Respuesta.

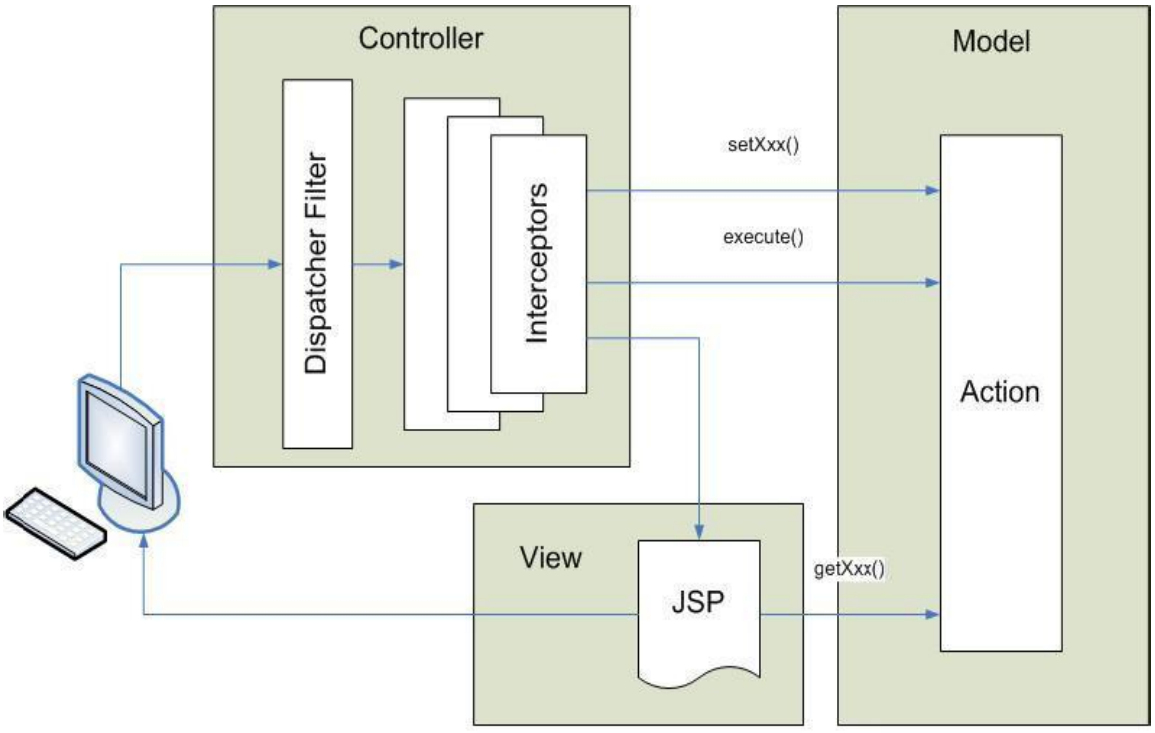


Ilustración 8 : Arquitectura Struts2 MVC

•El inicio de la Petición

El ciclo de la petición-respuesta comienza y finaliza en el browser del usuario. ***La petición arranca cuando se introduce una URL*** directamente en la barra de dirección del browser o cuando es generada mediante un enlace, formulario de la aplicación web.

http://localhost:8080/visualGate/index.action

Todas las peticiones son enviadas al ***Servlet Filter de Struts2***, y éste es el que toma la decisión de procesarla o no, según los archivos de configuración de la aplicación.

•Servlet Filter de Struts2

Las peticiones son recibidas por el contenedor de Servlets, en este caso usamos ***Tomcat*** como contenedor. ***Éste las envía a un Servlet o Filter para que las procese***. En el caso de Struts2 se usa un Filter y la clase que toma la petición se llama **FilterDispatcher.**

FilterDispatcher es el encargado de:

- Servir contenido estático
- Determina la configuración de la acción: El Filter usa las implementaciones de **ConfigurationManager** y el **ActionMapper** para determinar el mapeo de la acción (URL) de la petición entrante. Por defecto, Struts2 buscará la extensión .action
- Crea el **ActionContext**: Struts2 no presenta dependencia con la especificación de Servlets y sus acciones no reciben parámetros como HttpServletRequest y HttpServletResonse como lo hacía Struts, esto sigue la idea de reducir el acoplamiento.
- Crea el **ActionProxy**: Esta clase contiene toda la configuración e información de contexto para procesar la petición y debería contener los resultados de la ejecución después de que haya sido procesada.
- Y por último limpia el **ActionContext** para evitar pérdidas de memoria.

Finalmente cuando la clase ActionProxy es instanciada, creada y configurada, el método execute() es invocado. Ésta es la señal de que la Action está preparada para empezar.

Es probable que nos asalte una pregunta, por qué se usa un Filter y no un Servlet. Una de las razones por las cuales se usa, es que ***los Filters nos permiten participar en el mecanismo de interceptación de la petición***, y Struts2 usa Interceptores.

El objeto **ActionInvocation** gestiona el entorno de ejecución y contiene el estado de la acción. Esta clase es el núcleo de la clase **ActionProxy**. El entorno de ejecución se compone por tres componentes diferentes:

- **Actions**

Una de las primeras tareas que tiene **ActionInvocation** es consultar la configuración que está siendo usada y crear una instancia de la Action.

A diferencia de otros Frameworks que rehúsan las instancias de la acciones, Struts2 crea una nueva por cada petición que se recibe.

- **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.

Cada Action debería tener muchos interceptores configurados. Estos interceptores son invocados en el orden que han sido configurados. Después de haberse ejecutados todos los Interceptors, por convención el método execute() de la Action es llamado. Éste retorna un "String" o un objeto "Result". Después de que se haya ejecutado la lógica de la acción, los Interceptors son llamados, pero esta vez en orden inverso, para añadir post-proceso a la acción.

- **Results**

Después de que el proceso de la Action haya finalizado, es el momento de los Results. El método de la Action que procesa la petición retorna un String como resultado.

Según el resultado y la configuración se procesará un tipo de Result.

La configuración de cada Result determina principalmente: el tipo de vista a mostrar (JSP, Velocity Templates, FreeMarker, entre otros.

Finalmente se retorna la respuesta, si es que se ha generado de vuelta al usuario.

2.4.4.2. Explorando los elementos del Core

En esta sección, exploraremos cada uno de los componentes del núcleo en más detalle.

En la ilustración se pueden ver las relaciones que existen entre los diferentes componentes del núcleo, las flechas negras serían las relaciones que tienen los componentes con el Value Stack y el otro tipo de flechas sería la relación con el proceso de la petición.

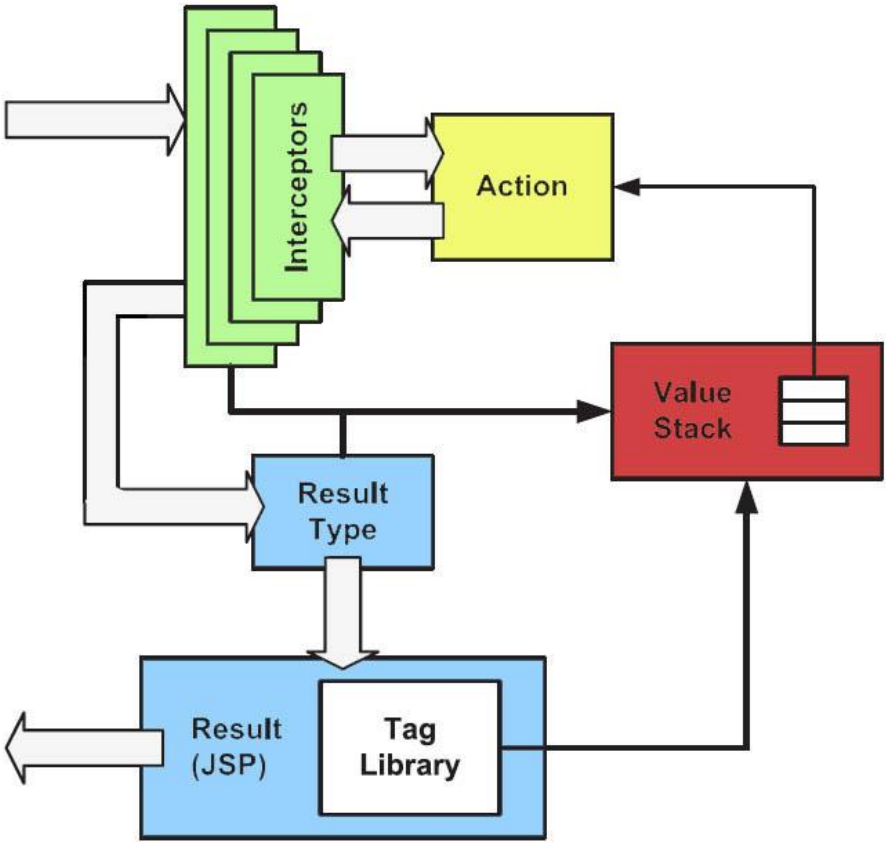


Ilustración 9 : Interacción y relación entre los diferentes elementos del Core

Actions

Las Actions son el núcleo del Framework Struts2. Cada dirección URL tiene mapeada una acción específica la cual provee 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, los Actions no está obligados a implementar o heredar de una interfaz o clase ya definida. Pueden ser POJO´s.

Sin embargo a veces tiene sentido extender alguna clase o implementar según que interfaces.

Struts2 posee varias interfaces de apoyo. Una de estas interfaces es Action:

Interface Action

```
public interface Action {

	public static final String SUCCESS = "success";
	public static final String NONE = "none";
	public static final String ERROR = "error";
	public static final String INPUT = "input";
	public static final String LOGIN = "login";

	public String execute() throws Exception;
}
```

Esta interfaz no hace nada más que aportarnos varios Strings para los valores de retorno del método execute() que deberemos implementar.

También vale la pena mencionar la clase ***ActionSupport***, que podemos extender.

Esta clase es muy interesante y recomendable extender ya que implementa a las siguientes interfaces:

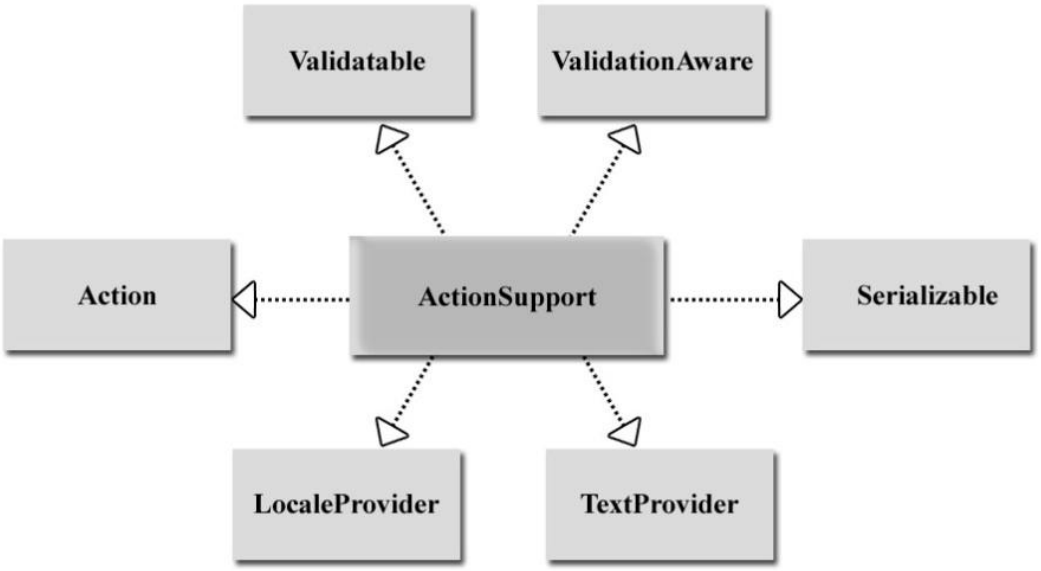


Ilustración 10 : ActionSupport

Las interfaces ***Validateable, ValidationAware*** nos proporcionan soporte en la validación, como las Annotations o archivos .XML

TextProvider, LocaleProvider, proveen soporte para localización e internacionalización.

Interceptors

Ya hemos hablado con anterioridad sobre los Interceptors, lo único que podemos añadir a este componente es lo que podemos llegar a conseguir si los utilizamos.

- Proveen lógica de pre-procesamiento antes de que la acción sea llamada
- Interactúan con la Action, proveyendo información como puede ser la inyección de dependencias controlado por Spring y poner a punto los parámetros de la petición en la Action.
- Proveen lógica de pro-procesamiento después de que la acción haya sido llamada.
- Pueden modificar el resultado que esté siendo retornado
- Recogen excepciones y por lo tanto pueden cambiar el tipo de resultado que se tenga que retornar.

Más sobre los interceptores disponibles en Struts2:
Ver ANEXO B

Value Stack & OGNL

Value Stack y OGNL (Object Graph Navigational Language) son dos cosas que están muy relacionadas. El Value Stack es exactamente lo que significa, una pila de valores, que en este caso, es una pila de objetos, aunque existen unas pequeñas diferencias que la diferencian de una pila normal.

La primera diferencia es que el contenido de la pila está formado por cuatro niveles.

Objetos Temporales
- Durante la ejecución los objectos temporales son creados y ubicados dentro de la Value Stack. Un ejemplo sencillo, sería la actual iteración de una colección de objetos.

Objetos del Modelo
- Si se quiere usar los objetos del modelo de dominio, estos serán ubicados en la Value Stack antes de que se ejecute la Action.

Objetos de la Acción
- Es la acción que está siendo ejecutada.

Objectos Nombrados
- #application, #session, #request, #attr y #parameters.

Otra diferencia es como se usa la pila. En una pila tradicional, para poder llegar a un elemento habría que ir introduciendo o extrayendo hasta llegar al que deseáramos (push & pop).

Con la Value Stack se realiza mediante búsquedas o evaluando una expresión OGNL.

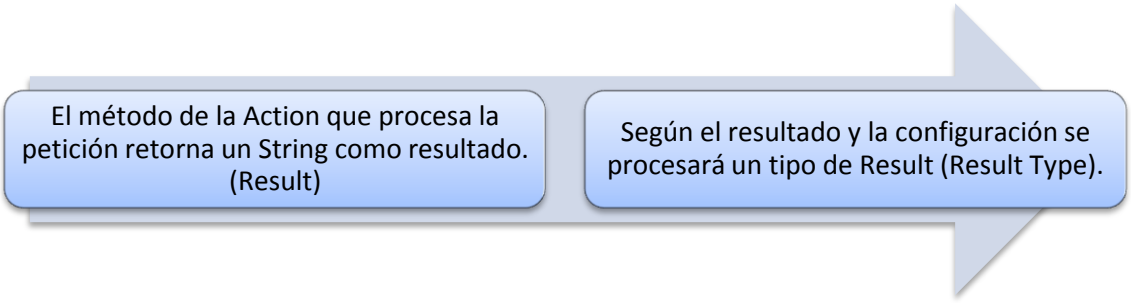
Al igual que en otros lenguajes de expresión, como puede ser el JSTL, OGNL nos proporciona un mecanismo para poder navegar sobre los diferentes objetos de la pila, utilizando una notación de punto, expresiones y llamadas a métodos de los objetos.

Estos son algunos ejemplos de lo que OGNL nos permite hacer:

OGNL	Descripción
address.postcode	Retorna un valor mediante la llamada getAddress().getPostcode()
#session['user']	Obtiene el objeto de usuario de la sesión.
!required	Retorna true si la llamada al método retorna false.
hasActionErrors()	Retorna el valor de la invocación al método.
[2].id	Invoca al método getId() del tercer elemento.
top	Retorna el elemento superior de la pila.
results.{name}	Retorna el valor de la llamada a getName() de cada uno de los elementos de la colección results.
@com.static.Constants@USER_NAME	Retorna una propiedad estática de la clase Constants.

Results & Result Types

Después de que la Action haya sido procesada, es el turno de enviar la información resultante de vuelta al usuario. En Struts2 esta tarea está dividida en dos partes:



El Result Type es el tipo de información que se retornará al usuario. La mayoría de estos tipos de resultados ya están pre configurados en Struts2 o se proveen mediante Plugins. Pero en el caso de que nos encontremos en una situación en que ninguno de los Results Types disponibles no se adapte a nuestro diseño, siempre podemos crear nuestro Result Type personalizado, al igual que ocurre con los Interceptors.

El Result Type configurado por defecto es el ***dispatcher***, el cual usa un JSP para mostrar la respuesta al usuario.

Result Type	Nombre de la Clase	Descripción
dispatcher	org.apache.struts2.dispatcher.ServletDispatcherResult	Muestra un JSP.
chain	com.opensymphony.xwork2.ActionChainResult	Encadena una Action con otra. (Este tipo de resultado es usado por el interceptor chainning)

freemarker	org.apache.struts2.views.freemarker.FreemarkerResult	Muestra un plantilla de Freemarker
httpheader	org.apache.struts2.dispatcher.HttpHeaderResult	Retorna una cabecera http configurada.
redirect	org.apache.struts2.dispatcher.ServletRedirectResult	Redirige el usuario a una URL ya configurada.
redirectAction	org.apache.struts2.dispatcher.ServletActionRedirectResult	Redirige el usuario a una Action ya configurada.
stream	org.apache.struts2.dispatcher.StreamResult	Retorna un Stream como respuesta al browser, útil para imágenes, pdf, etc…
Velocity	org.apache.struts2.dispatcher.VelocityResult	Muestra una plantilla de Velocity
Xslt	org.apache.struts2.views.xslt.XSLTResult	Muestra un XML en el browser.
plaintext	org.apache.struts2.dispatcher.PlainTextResult	Retorna el contenido como "plain text"

Tag Library

Las librerías de Tags se usan en los .jsp para obtener o añadir información a las Actions, se puede decir que proveen una intersección entre las acciones y las vistas, favoreciendo la mantenibilidad y manteniendo la lógica encapsulada, reduciendo la tentación de cortar y pegar código.

Las principales diferencias entre las librerías de Tags de Struts2 y el resto como pueden ser las de JSTL son las siguientes:

- Mejor integración con el Framework Struts2.
- Mejor uso de la Value Stack.
- Mejor uso del OGNL para evaluar expresiones.

Existen cuatro categorías diferentes de librerías de Tags en Struts2:

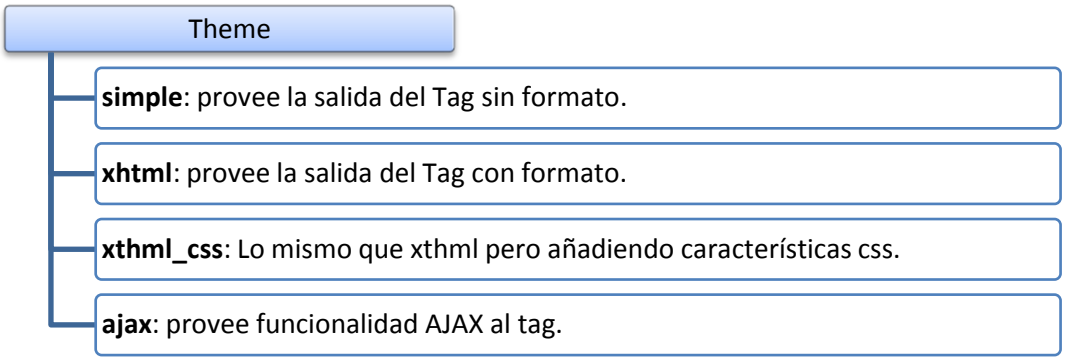
Control Tags
- Controlan que información es mostrada en la vista, como por ejemplo la manipulación de colecciones dentro de un Iterate.

Data Tags
- Usados para la manipulación de datos o creación. Generación de URL´s, enlaces, texto internacionalizado, mostrar datos de la acción ejecutada.

Form Tags
- Este grupo provee Wrappers, formularios, Widgets entre otros.

NonForm Tags
- Estos Tag's son usados en formularios, pero no para la entrada de datos, como son los mensajes de error.

Struts2 provee diferentes Templates para las librerías de Tags, esta característica es llamada:



2.4.5. Configurando los elementos del Framework

Ya hemos hablado de los elementos principales del Framework, ahora es el momento de la configuración. Empezaré por el famoso fichero web.xml y después hablaré sobre la configuración de las Actions y demás componentes por medio de las Annotations y XML utilizando el fichero struts.xml

2.4.5.1. web.xml

El fichero web.xml es un fichero de configuración J2EE que ***determina como los elementos de la petición HTTP son procesados por el contenedor de Servlets*** (Tomcat). No es un fichero de configuración de Struts2 pero hace falta que lo configuremos un poco para que Struts2 pueda correr en nuestra aplicación.

Web.xml

```
<filter>
  <filter-name>action2</filter-name>
  <filter-class>org.apache.struts2.dispatcher.FilterDispatcher
  </filter-class>
</filter>
<filter-mapping>
  <filter-name>action2</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

Ésta es la configuración mínima que requiere el fichero web.xml para que pueda funcionar el Framework Struts2. Más adelante, veremos cómo al fichero web.xml se le van añadiendo configuraciones adicionales para poder usar diversos Plugins.

2.4.5.2. struts.xml

El fichero de configuración struts.xml es el ***fichero principal de configuración*** de Struts2.

Existen dos modos para poder configurar las Actions en Struts2, una es por medio de ***Annotations*** y otra es mediante el fichero Struts.xml mediante ***XML***.

La configuración mediante anotaciones es una opción muy buena para mantener las acciones y su configuración en el mismo fichero. Sin embargo, las Annotations de momento no ofrecen toda la potencia en cuanto a la configuración que aporta el XML. Así que lo recomendable, es usar las dos formas de configuración, más adelante explicaremos dónde usar cada una.

Ahora describiré un poco los Tags disponibles en el fichero de configuración de Struts.xml. Podría describir detalladamente cada uno de los Tags y sus parámetros, pero eso requeriría demasiada documentación, así que solo mencionaré los más importantes y explicaré brevemente que es lo que hacen.

Los Tags mas importantes que están disponibles en el fichero de configuración Struts.xml son los siguientes:

- **Include File**

  La configuración del fichero Struts.xml puede estar dividida en diferentes ficheros, de esta manera conseguimos ganar en manejabilidad y modularidad en la configuración.

Include File

```
...
<include file="struts-modulo1.xml"/>
<include file="struts-modulo2.xml"/>
...
```

- **Packages**

  Dividir la configuración en diferentes archivos ***es un camino para la modularidad***, y los Packages es otra. Los Packages proveen un contenedor para almacenar el mapeo y la configuración de ejecución de las Actions que estén incluidas dentro. Éstos pueden ***extender a otros Packages para heredar configuraciones*** y así ahorrarnos código de configuración.

  Struts2 nos aporta un package por defecto, llamado “Struts-default” que contiene todos los Result Types, interceptores y pilas de interceptores que mencionamos anteriormente. Es recomendable extender este package para poder acceder a estos componentes.

Result Types

Antes de que Struts2 pueda usar los Results, se necesita tener configurados los Result Types correctamente. "Struts-default" ya nos aporta todos los Result Types configurados, así que sólo estaríamos obligados a configurar los Result Types si no extendiéramos el package "Struts-default" o quisiéramos crear nuestros propios Result Types.

Result Types

```
<package name="home-package" extends="struts-default" namespace="/">
  …
  <result-types>
   <result-type name="fotoRSS"
                class="com.visualGate.util.RssFotoResult" />
   <result-type name="fotoRSS2"
                class="com.visualGate.util.RssFotoResult2" />
  </result-types>
  …
</package>
```

Interceptors

Como los Result Types, los Interceptors tienen una configuración muy simple. Cuando hablamos de los Interceptors, mencionamos que existen simples interceptores o pilas de interceptores. En el siguiente ejemplo he creado un interceptor personalizado llamado "security" y lo he adjuntado a una pila de interceptores llamada "paramsPrepareParamsStack" creando una nueva pila llamada "securedStack". Como ocurre también con Result Types, el package "Struts-default" ya nos aporta todos los Interceptors configurados.

Interceptors

```
<interceptors>

  <interceptor name="security"
               class="com.visualGate.util.SecurityInterceptor" >
    <param name="requiresAuthentication">/foto,/admin</param>
  </interceptor>

  <interceptor-stack name="securedStack">
    <interceptor-ref name="security" />
    <interceptor-ref name="paramsPrepareParamsStack" />
  </interceptor-stack>

</interceptors>
```

Una de las ventajas de Struts2 es que soporta diferentes ciclos de vida por Action, éstos están basados en las pilas de interceptores. Como hemos visto, personalizar el ciclo de vida de una Action es relativamente sencillo.

Global Results

Los Global Results son usados por diferentes acciones. Si tenemos varias acciones que requieren del mismo resultado, ésta sería la mejor opción.

Global Results

```
<global-results>
  <result name="unknownError">/WEB-INF/jsp/error.jsp</result>
</global-results>
```

Global Exception Mappings

Describiendo la excepción que puede llegar a ocurrir, podemos recogerla y redirigir el resultado a un .jsp mediante un Result configurado o un Global Result como es el caso del ejemplo anterior. De esta manera nos evitamos que al usuario le aparezca la típica pila de errores de java en pantalla.

Global Exception Mappings

```
<global-exception-mappings>

  <exception-mapping result="LobError"
    exception="javax.persistence.PersistenceException”/>
  <exception-mapping result="dupPK"
    exception="org.hibernate.exception.ConstraintViolationException"/>
  <exception-mapping result="unknownError"
    exception="java.lang.Exception”/>

</global-exception-mappings>
```

Actions

La configuración de las Actions por medio del XML contiene más opciones de configuración disponibles que por medio de Annotations.

Una diferencia de la configuración por medio de las anotaciones es que con XML, se puede declarar una Action por defecto en un package. Esto nos va bien cuando un usuario entra una dirección y no especifica la acción a ejecutar.

Actions

```
<package name="base-package" extends="home-package" >
…
  <default-action-ref name="usuarioR"></action>
  <action name="usuarioR"
        class="com.visualGate.actions.usuario.UsuarioRAction" >
    <result name="success">/WEB-INF/…/usuarioR-success.jsp</result>
    <result name="input">/WEB-INF/…/usuarioR-input.jsp</result>
    <interceptor-ref name="paramsPrepareParamsStack"/>
  </action>
…
</package>
```

2.4.5.3. ZERO Configuration Annotations

Struts2 requiere Java5 para poder funcionar, eso le aporta varias características, entre ellas la de poder ***utilizar las anotaciones como mecanismo de configuración***.

La ventaja real de la utilización de esta característica es que ***se puede reducir la configuración XML*** a favor de la configuración basada en anotaciones. Cada Action o acción en la aplicación necesita tener su propia configuración en el fichero Struts.xml, sólo hay que imaginar cómo llegaría a ser de grande este fichero si tuviéramos más de 20 acciones en la aplicación, y eso, es pensar en una aplicación pequeña. Usando anotaciones la información relacionada con la clase Action se ubica dentro de la misma clase.

Configuración basada en XML – Struts.xml

```
<package name="base-package" extends="home-package" >
…
<action name="usuarioR"
        class="com.visualGate.actions.usuario.UsuarioRAction" >
  <result name="success">
        /WEB-INF/jsp/usuario/usuarioR-success.jsp
  </result>
  <result name="input">
        /WEB-INF/jsp/usuario/usuarioR-input.jsp
  </result>
  <interceptor-ref name="paramsPrepareParamsStack"/>
</action>
…
</package>
```

Configuración basada en Annotations

```
@ParentPackage("base-package")
@Results({
    @Result(name="success",
            value="/WEB-INF/jsp/usuario/usuarioR-success.jsp",
            type=ServletDispatcherResult.class),
    @Result(name="input",
            value="/WEB-INF/jsp/usuario/usuarioR-input.jsp",
            type=ServletDispatcherResult.class)
         })

public class UsuarioRAction extends BaseUsuarioAction {

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

Como vemos, toda la configuración en XML de Struts.xml ya no sería necesaria. Y personalmente prefiero tener la configuración en la misma acción, que estar buscándola en un fichero con miles de configuraciones de diferentes acciones. Más adelante podremos ver cómo nos podemos deshacer de la mitad del código, gracias al plugin Codebehind.

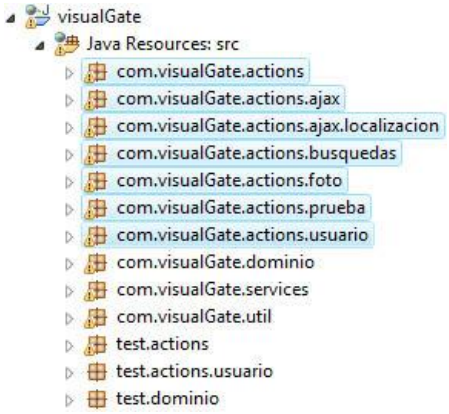
XML y Annotations trabajando juntos!

En una aplicación lo suficientemente grande y compleja, se requiere el uso de XML ya que las anotaciones no cubren todos los tipos de configuraciones posibles para una acción. Así que lo mejor que podemos hacer, es utilizar Annotations siempre que podamos y cuando no nos aporten lo que necesitamos, recurramos a la configuración XML.

Un ejemplo práctico en el que Annotations no nos pueden ayudar, es en la configuración de nuevos tipos de resultados (Result Types) o pilas de interceptores diferentes a las de la configuración por defecto.

Activando "ZERO Configuration"

Para activar la configuración Zero, lo primero que se ha de hacer es decirle a Struts2 qué paquetes con acciones utilizarán las anotaciones, y para eso hay que configurar el FilterDispatcher de Struts2. Struts2 aplicará las anotaciones, también los subpackages del package declarado.



Web.xml

```
<filter>
  <filter-name>action2</filter-name>
  <filter-
class>org.apache.struts2.dispatcher.FilterDispatcher</filter-class>
    <init-param>
      <param-name>actionPackages</param-name>
      <param-value>com.visualGate.actions</param-value>
    </init-param>
</filter>
<filter-mapping>
  <filter-name>action2</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

2.4.6. Integrando Struts2 con otras Tecnologías

Una de las ventajas que tiene Struts2 es la de disponer de una gran cantidad de Plugins que añaden funcionalidades al Framework o que te permiten la integración de éste con otros Frameworks como pueden ser Spring y JSF.

Existen diversas técnicas para integrar tecnologías externas a Struts2, éstas son:

Interceptores	Result Types	Plug-in Packages	Plug-in extensions points
•Permiten cambiar el ciclo de vida de una petición-respuesta, modificar el resultado e inyectar objetos dentro de la acción.	•Permiten post-procesado después de haber ejecutado la acción, y muestran la información retornada por la acción.	•Packages con nuevos interceptores, Result Types, Results y acciones que podemos reusar en diferentes proyectos.	•Permiten nuevas implementaciones de las clases del núcleo del Framework, cambiando el comportamiento original de Struts2.

El objetivo de este proyecto, no es describir cada una de las opciones o técnicas de integración de tecnologías, ni tampoco todas las tecnologías que existen alrededor de Struts2, pero si veremos las principales que se han utilizado en VisualGate.

2.4.6.1. Diseño y decoración de páginas

El desarrollar aplicaciones web, requiere de una serie de normas de diseño. Diseñar las páginas que serán utilizadas por toda la aplicación, no es una tarea tan sencilla, vale la pena dedicarle un poco de tiempo, ya que si no lo hacemos, a la larga resultará peor. Para facilitarnos un poco el trabajo en esto de la decoración de páginas y el diseño, dos Plugins son aportados por Struts2:

- Tiles
- Sitemesh

Ambos tienen sus ventajas y desventajas, pero lo que más destaca es que la versión de Tiles para Struts2 se encuentra en fase de pruebas, y por el otro lado Sitemesh aporta una versión estable, además, ***Sitemesh puede ser usado en cualquier Framework, y eso se traduce en una reducción de acoplamiento al Framework***.

Más sobre Plugins en Struts2, y en concreto Sitemesh:
Ver ANEXO A.1

2.4.6.2. Servicios de Negocio e Inyección de dependencias.

Una pregunta nos viene a la cabeza cuando hablamos de integrar Spring y Struts2, ya que Spring es un Framework MVC al igual que Struts2. Con ambos podríamos construir una aplicación web, pero cada uno de ellos ofrece aspectos que el otro no ofrece o no es tan bueno ofreciéndolos, y ahí es donde entra ***Spring, para aportarnos servicios de negocio e inyección de dependencias.***

Struts2 nos proporciona el Plugin de Spring, que nos permitirá agregar las características de este Framework dentro de Struts2.

No veremos Spring en profundidad, pero sí que hablaremos sobre la inyección de dependencias y cómo se integra con Struts2.

Aun así, en el mercado existen más opciones como Plexus, el cual posee un Plugin para Struts2 pero se encuentra en fase experimental.

Más sobre Plugins en Struts2, y en concreto Spring:
Ver ANEXO A.3

2.4.6.3. Bases de Datos

Struts2 no aporta nada especial que permita la integración con las bases de datos, pero si la manera en que se acceden a ellas.

Por medio de las librerías de Tags

- Esta no es la mejor opción, ya que violaría el principio del MVC de Struts2, pero es posible hacerlo.

Utilizando DAO por medio de la inyección de dependencias

- Las acciones tienen la ventaja de inyectar dependencias como pueden ser los **DAO**, una vez que la acción tiene una instancia del **DAO**, esta puede llamar a los métodos como si fueran suyos, y así acceder a los servicios de negocio aportados por el **DAO**.

Utilizando DAO/ORM por medio de la inyección de dependencias

- Esta aproximación es como la anterior, pero añadiendo un **ORM** como puede ser el caso de **iBatis** o **Hibernate**. Si se usa un **ORM**, se recomienda encarecidamente usar **Spring para la inyección de dependencias**, ya que Spring nos proveerá de todo lo necesario para **configurar y inicializar** los diferentes **DAO** que estemos inyectando dentro de la acción. De esta manera cuando la acción esté a punto de ser ejecutada, todas las intancias DAO estarán preparadas y listas para ser usadas.

Más sobre el patrón DAO:
Ver ANEXO D

Más sobre Hibernate:
Ver ANEXO A.4

Capitulo 3. ANÁLISIS PREVIO

En este capítulo haremos un análisis del estado previo antes de ponernos a diseñar e implementar la aplicación web, desde la metodología usada, a un estudio de costes de los recursos humanos y técnicos que requiere el proyecto.

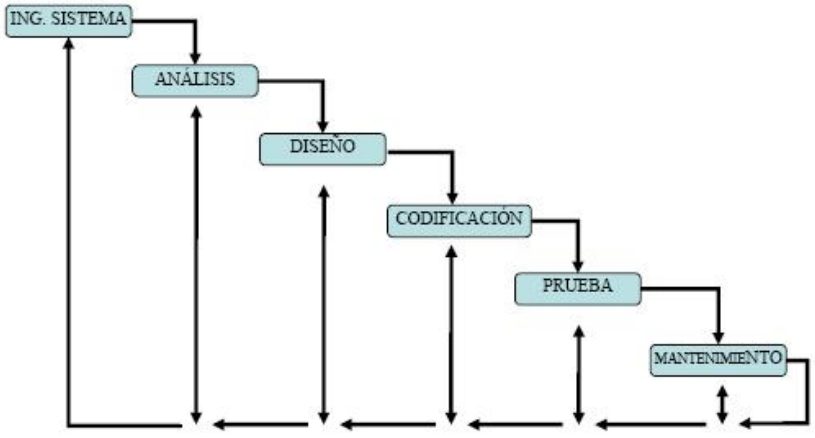
3.1. Metodologías

La metodología a seguir para el desarrollo de un sistema software es uno de los factores más importantes a tener en cuenta. La metodología es una de las cosas más visibles que diferencia a un ingeniero en informática de uno que no lo es.

Las ventajas de seguir la metodología de una ingeniería, en nuestro caso la ingeniería del software, son muchas:

- Se preocupa de la fiabilidad y del rendimiento.
- Trata de reducir costes y complejidad.
- Basa sus modelos en teorías matemáticas sólidas.
- Utiliza diagramas formales.
- Se utilizan técnicas probadas que dan resultados precisos.
- …

En el desarrollo de la aplicación web, se va a tratar de desarrollar un productor basado en el denominado ciclo de vida clásico de un sistema software, el cual trata de determinar las diferentes fases por las que se tiene que pasar a fin de conseguir un buen producto final. A continuación un esquema de las etapas de este ciclo de vida:



Más sobre las etapas del ciclo de vida de un sistema Software: Ver ANEXO H

Ilustración 11 : Ciclo de vida clásico de un sistema Software

Lo más destacable sería la realimentación que se da en casa fase. Cuando se detecta un error o carencia, se en la etapa que sea, se debe rehacer todo lo relacionado con la posible causa desde la fase de origen del problema. El objetivo de esta realimentación es conseguir un

producto sólido y libre de parches. Además, el volver a una fase anterior se traduce en un aumento de costes de tiempo y económicos.

3.2.Procesos a realizar

A continuación se procederá a describir los puntos generales establecidos para la consecución del proyecto:

- •Análisis de los requisitos.
- •Análisis de las posibles alternativas de desarrollo (tecnologías candidatas)
- •Diseño de la aplicación web.
- •Aprendizaje de las tecnologías elegidas.
- •Implementación de la aplicación web.
- •Pruebas exhaustivas de la aplicación desarrollada.
- •Despliegue.

3.3.Estimación y planificación inicial

Una estimación de las horas previstas para la realización del proyecto:

- Esfuerzo previsto:
  - Horas aproximadas de análisis: 100 horas
  - Horas aproximadas de programación: 250 horas
- Número de programadores: 1
- Número de analistas: 1

HORAS TOTALES APROX. = 87 DIAS * 4 HORAS/DIA = 350 HORAS

3.3.1. Descomposición en actividades: duración en horas

La planificación se ha establecido en horas (aproximadas) puesto que es una medida mucho más intuitiva a la hora de tomar conciencia del esfuerzo estimado.

Ref	Descripción	Analista	Programador
1	**Inicio**		
2	**Análisis de los Requisitos**		
3	Requisitos funcionales	8	
4	Requisitos no funcionales	8	
5	Reajustes de los requisitos	8	
6	**Análisis de tecnologías candidatas**	16	
7	**Diseño**		
8	Casos de uso	4	
9	Diagrama de casos de uso	4	
10	Modelo de dominio	8	
11	Diagrama de actividades	8	
12	Diagramas de flujo	16	
13	Diseño del sistema	16	
14	Diseño de las pantallas	8	
15	**Implementación**		
16	Instalación y configuración del software a utilizar		8
17	Aprendizaje de las tecnologías a usar		60
18	Capa dominio		4
19	Capa persistencia		8
20	Capa servicios		8
21	Capa aplicación		60
22	Capa presentación		40
23	**Pruebas**		60
24	**Documentación**		16
25	**Despliegue**		4
26	**Final**		

Diagrama de Gantt:
Ver ANEXO I

3.4.Estudio económico

El estudio económico hace referencia al precio total real que tendría este proyecto. Se ha supuesto que se tuviese que adquirir todo lo necesario (hardware y software), aunque, claro está, cualquier empresa ya dispondría de parte de estos recursos.

Basándonos en la anterior planificación de actividades se han añadido las siguientes suposiciones:

- Distinción entre el sueldo por hora de analista y programador.

- Distinción entre los costes para analista y programador en función del coste de mercado, y el coste interno para la empresa de software.

3.4.1. Coste de los recursos humanos

- **Coste de mercado**

Éste es el coste de los recursos humanos que supondría el hecho de contratar una empresa de software para el desarrollo de este producto. Aquí se define a cuánto se paga de media en el mercado actualmente la hora de analista y programador, cuando se subcontratan por medio de una empresa especializada en desarrollo de aplicaciones software.

Para este apartado tendremos en cuenta que el precio por hora media estándar de un analista es de 60€, mientras que la del programador está a 35€. Si hacemos los cálculos pertinentes:

Recurso	Número	Importe/hora	NºHoras	Importe Total
Analista	1	60€	100	6600 €
Programador	1	35€	250	8750 €
Total	2	-	335	15350 €

- **Coste interno**

Éste es el coste interno que supondría a la empresa de desarrollo el pagar a los empleados encargados de desarrollar el proyecto según la planificación. Para este apartado tendremos en cuenta que el precio por hora medio estándar de un analista en una empresa se paga a 36€, mientras que la del programador está a 21€. Si hacemos los cálculos pertinentes:

Recurso	Número	Importe/hora	NºHoras	Importe Total
Analista	1	36€	100	3600 €
Programador	1	21€	250	5250 €
Total	2	-	335	8850 €

3.4.2. Coste de los recursos técnicos

El coste de los recursos informáticos para el desarrollo, pueden convertirse en uno de los más significativos dentro de un proyecto y a la vez de lo más difíciles a la hora de calcular. Los costes de los productos son caros, pero además no se pueden atribuir exclusivamente a un proyecto concreto puesto que, generalmente, lo podemos emplear para posteriores proyectos.

Una de las cualidades de nuestro proyecto, es el uso de tecnologías Open Source, eso nos reduce los gatos exclusivamente a temas de hardware.

Capitulo 4. REQUISITOS DEL SISTEMA

Una vez estudiado las tecnologías pertinentes, es el momento de realizar la aplicación web que represente en cierta medida a estas tecnologías, la aplicación permitirá a los usuarios poder compartir y ver las fotos de usuarios registrados a través de una página Web. Debido a que este tipo de Webs deben estar siempre a la última en cuanto a servicios que ofrecen y se basan principalmente en la participación del usuario, se ha de crear un entorno amigable y fácilmente ampliable para que, en caso de que el administrador quiera añadir nuevas funcionalidades en el futuro, éstas no tengan un coste elevado.

Tal y como se ha explicado con anterioridad, el proyecto se va a realizar utilizando el lenguaje de programación Java (J2EE). Otra característica importante por no decir la más importante de la implementación, es la utilización del Framework Struts2 basado en el patrón MVC que nos agilizará el desarrollo y el mantenimiento de la aplicación Web, además usaremos otros Frameworks como Spring para la inyección de dependencias e Hibernate para la persistencia.

También se usarán otras tecnologías como AJAX, Mashups, RSS y Google Maps para proveer de funcionalidades a la aplicación.

Por lo tanto, en este capítulo se analizarán los requisitos necesarios, para cumplir los objetivos planteados y se detallará los Roles que existirán dentro de la aplicación Web. Estos requisitos se han sido divididos en dos:

Funcionales	•Los requisitos funcionales son los procesos que se pueden realizar en la plataforma como por ejemplo el registro de usuarios, fotos, autenticaciones…
No Funcionales	•Los requisitos no funcionales son condiciones que debe cumplir el proyecto como puede ser la rapidez, seguridad y coste.

4.1.Roles

Se han definido dos roles o usuarios diferentes que pueden utilizar la aplicación. Se diferencian básicamente por los permisos que poseen y de las acciones que pueden llegar a desempeñar dentro de la aplicación web.

Usuario no Registrado

- Un usuario no Registrado tiene acceso a toda la aplicación exceptuando al registro de Fotos y las opciones como eliminación y modificación de Fotos.

Usuario Registrado

- Un usuario Registrado tiene acceso a todas las acciones disponibles del usuario no registrado exceptuando el registro de usuario, solo que ahora será actualización del perfil.

4.2.Requisitos funcionales

Los requisitos funcionales que debe cumplir la aplicación son los siguientes:

Gestión de usuarios
- Registro de Usuario
- Actualización de Usuario

Gestión de fotos
- Registro de Fotos
- Modificación de Fotos
- Eliminación de Fotos

Motor de búsquedas
- Búsqueda de Fotos básica
- Búsqueda de Fotos Avanzada
- Búsqueda de Fotos Recientes
- Búsqueda de Fotos personales

Control de Acceso
- Autenticación de Usuarios.
- Autorización de Uusarios.
- Restricción de contenido a usuarios no registrados.

Google Maps
- Implementación de un Mashup (Google Maps) para ubicar la localización de la foto.

Sindicación de contenido
- Publicar la información de las fotos de manera externa.

4.3.Requisitos no funcionales

La aplicación también tiene que cumplir las siguientes condiciones o requisitos no funcionales:

Seguridad
- La aplicación web ha de proporcionar cierta privacidad de datos a los usuarios.

Entorno Amigable
- La aplicación web ha de ser fácil e intuitiva de usar.

Velocidad
- La aplicación web ha de ser lo suficientemente rápida.

Escalable
- La aplicación no deberá colapsarse al ampliar el número de usuarios o recursos. Permitiendo ampliaciones sin un gran coste.

Coste
- Software Libre
- Realizar la aplicación en el menor tiempo posible.

Ampliable
- Facilidad para incluir nuevos componentes o añadidos a la aplicación sin que eso suponga un gran coste de trabajo.

MultiThread
- El Sistema puede ser utilizado por más de una persona de manera concurrente.

Documentación
- La aplicación web, ha de estar comentada y documentada al máximo para futuras mejoras o ampliaciones.

Recogida de errores
- El sistema debe de disponer de un sistema para capturar errores e informar al usuario en caso de que se produzcan de una manera “suave”.

Capitulo 5. DISEÑO DE VISUALGATE

Una vez analizados los requisitos del sistema, tanto funcionales como no funcionales, es el momento de diseñar la aplicación. Empezaré describiendo los casos de uso, especificando el objetivo, rol y su descripción detallada. Se desarrollarán los diagramas más comunes y útiles para la posterior implementación del sistema, como puede ser el diagrama de actividades, o el modelo de dominio.

El diseño de la base de datos, no será necesario, ya que usaremos Hibernate para la persistencia de datos, el cual se encargará de crear la base de datos a partir del modelo de dominio. Aprovecharé también para especificar que tecnologías se integrarán dentro de la aplicación web, y que responsabilidades tendrán.

La metodología utilizada para el diseño y modelado de la aplicación se ha basado en la tecnología UML (Unified Modelling Language , lenguaje que nos permite desarrollar en el proceso de la ingeniería con una metodología orientada a objetos.

5.1.Casos de Uso

UC1: Registrar Usuario	Rol: Usuario no Registrado
Objetivo	Permitir al usuario registrar en la aplicación
Descripción	Antes de que los usuarios puedan registrar o añadir una foto, necesitan estar registrados. Para poderse registrarse, tendrán que introducir una serie de datos, entre ellos el Email y el Password en un formulario que será enviado, validado y procesado. Una vez Registrado el usuario en la base de datos, podrá identificarse en la aplicación web usando el UC3 y acceder a opciones que antes no podía.

UC2: Actualizar Usuario	Rol: Usuario Registrado
Objetivo	Permitir al usuario actualizar su perfil en la aplicación
Descripción	Para que un usuario pueda actualizar su perfil, ha de estar identificado en la aplicación web, ya que si no lo está no podrá acceder a la opción de Perfil. Una vez que acceda a dicha opción, se le presentará el mismo formulario que el de registro, a diferencia de que este contendrá la información del usuario.

UC3: Subir Foto de Usuario	Rol: Usuario Registrado
Objetivo	Permitir al usuario actualizar su perfil añadiendo su foto en la aplicación
Descripción	Como su nombre indica, permite subir una foto del usuario y asociarlo a su perfil. Se usará el mismo formulario utilizado para los UC1 y UC2 añadiendo un botón para subir la foto.

UC4: Autenticación o "Login" de Usuario	Rol: Usuario no Registrado
Objetivo	Permitir al usuario autenticarse en la aplicación web.
Descripción	El usuario necesita ser autenticado para poder acceder a ciertas características de la aplicación. Para autenticarse, existe la condición de que el usuario se encuentre registrado, si no, no podrá hacerlo ya que no existirá en la base de datos de la aplicación como usuario válido. El login, se presenta mediante un formulario, donde el usuario ha de introducir su email y su Password, si los valores coinciden con los almacenados durante el proceso de registro, el usuario tendrá acceso a nuevas funcionalidades.

UC5: "Logout" Usuario	Rol: Usuario Registrado
Objetivo	Permitir al usuario autenticarse en la aplicación web.
Descripción	Una vez hecho el login, el usuario ha de ser capaz de desconectarse de la aplicación, esto significa que una vez hecho, no tendrá acceso a las funcionalidades que aporta la aplicación a usuarios autenticados.

UC6: Registrar o Añadir Foto	Rol: Usuario Registrado
Objetivo	Permitir al usuario registrar una foto.
Descripción	Registrar o añadir Foto es una funcionalidad exclusiva de usuarios registrados, así que los usuario tendrá que validarse antes de poder usar esta funcionalidad. Este es posiblemente el caso de uso más complejo, ya que intervienen todas las clases del modelo de dominio para crear una foto: Foto, Usuario, Tipo, Localización, País y Ciudad. El proceso de registro de una foto, ser realizará mediante un workFlow, donde se irán recogiendo y validando los datos en diferentes pasos. Una vez completado el proceso, los datos se almacenarán junto a la foto y su Thumbnail en la base de datos.

UC7: Cambiar el tipo de Foto	Rol: Usuario Registrado
Objetivo	Permitir al usuario cambiar de pública a privada o viceversa su foto.
Descripción	Cambiar el tipo de Foto es una funcionalidad exclusiva de usuarios registrados, y solo se puede realizar cuando la foto es propiedad del usuario. Básicamente, sirve para cambiar los permisos de visualización de la foto en la aplicación web. Una foto pública podrá verla cualquier usuario, esté registrado o no lo esté, en cambio una foto privada solo será visible por el propietario.

UC8: Eliminar Foto	Rol: Usuario Registrado
Objetivo	Permitir al usuario eliminar su propia foto.
Descripción	Al igual que cambiar el tipo de foto, es una funcionalidad exclusiva de usuarios registrados y sólo disponible por el usuario propietario de la foto.

UC9: Búsqueda de Fotos por nombre	Rol: Todos
Objetivo	Permitir al usuario realizar búsquedas rápidas.
Descripción	Cualquier usuario registrado o no registrado, tendrá acceso a esta funcionalidad, su finalidad es encontrar las fotos que coincidan con el nombre introducido.

UC10: Búsqueda de Fotos avanzada	Rol: Todos
Objetivo	Permitir al usuario realizar búsquedas avanzadas.
Descripción	Al igual que la búsqueda de fotos por nombre, cualquier usuario tendrá acceso a esta funcionalidad. La búsqueda avanzada permitirá realizar búsquedas de fotos especificando, el nombre de usuario, el nombre de Foto, país y ciudad. Todos estos campos son opcionales, así que si el usuario no introduce ninguna condición, se mostrarán todas las fotos almacenadas.

UC11: Mostrar Fotos Recientes	Rol: Todos
Objetivo	Mostrar al usuario las últimas fotos subidas.
Descripción	Este caso de uso, es el más utilizado de todos, ya que nada más entrar en la aplicación se mostrarán las fotos subidas recientemente. No hace falta ser un usuario registrad, ya que las fotos mostradas son sólo las fotos públicas de todos los usuarios registrados en la aplicación.

UC11: Mostrar Fotos Recientes	Rol: Todos
Objetivo	Mostrar al usuario las últimas fotos subidas.
Descripción	Este caso de uso, es el más utilizado de todos, ya que nada más entrar en la aplicación se mostrarán las fotos subidas recientemente. No hace falta ser un usuario registrado o no, ya que las fotos mostradas son sólo las fotos públicas de todos los usuarios registrados en la aplicación.

UC12: Descargar Foto	Rol: Todos
Objetivo	Permitir al usuario, poder descargar una foto.
Descripción	Cualquier usuario registrado o no, podrá descargar cualquier foto, exceptuando las fotos privadas, en cuyo caso el usuario deberá estar registrado y ser propietario de la foto.

UC13: Ver Foto	Rol: Todos
Objetivo	Permitir al usuario, poder visualizar una foto.
Descripción	Cualquier usuario registrado o no, podrá visualizar cualquier foto, exceptuando las fotos privadas, en cuyo caso el usuario deberá estar registrado y ser propietario de la foto.

UC14: Mostrar Mis Fotos	Rol: Usuario Registrado
Objetivo	Permitir al usuario, poder visualizar sus fotos.
Descripción	Como su nombre indica, se muestran las fotos del usuario que lo solicita. Pero para acceder a esta funcionalidad, el usuario deberá estar autentificado en la aplicación. Una vez que el usuario acceda a esta funcionalidad, se le mostrarán todas las fotos que sean de su propiedad, tanto las públicas como las privadas.

UC15: Publicar la información de las fotos.	Rol: Todos
Objetivo	Sindicación mediante RSS feed.
Descripción	La información entrada por los usuarios, debería estar disponible externamente, y así poder acceder a ella sin necesidad de un navegador. Todo esto se implementará mediante RSS feed.

5.2.Diagrama de Casos de Uso

En el siguiente diagrama se muestran las diferentes acciones que pueden realizar los diferentes actores en la aplicación web. El usuario registrado podrá realizar las acciones del usuario no registrado.

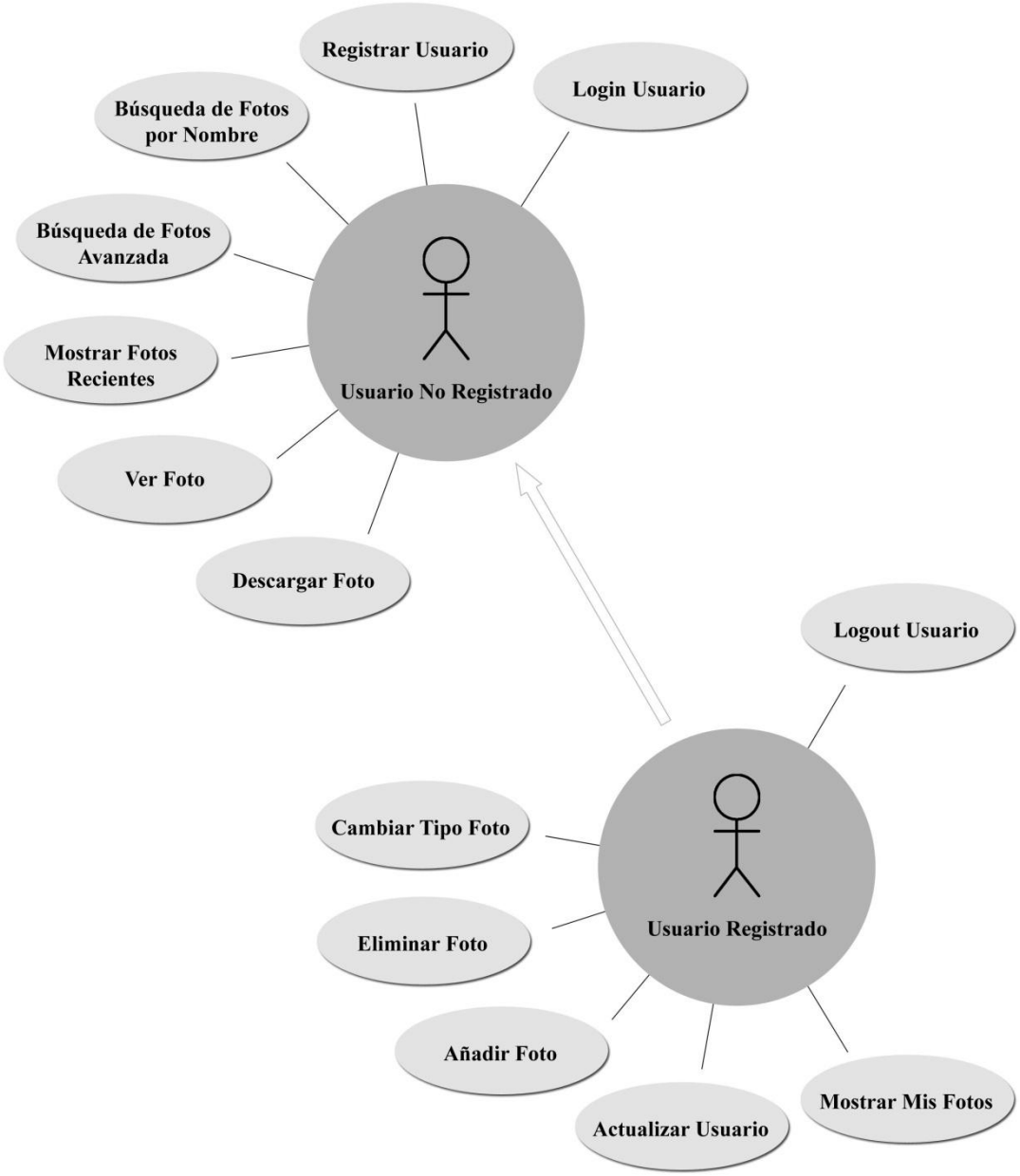
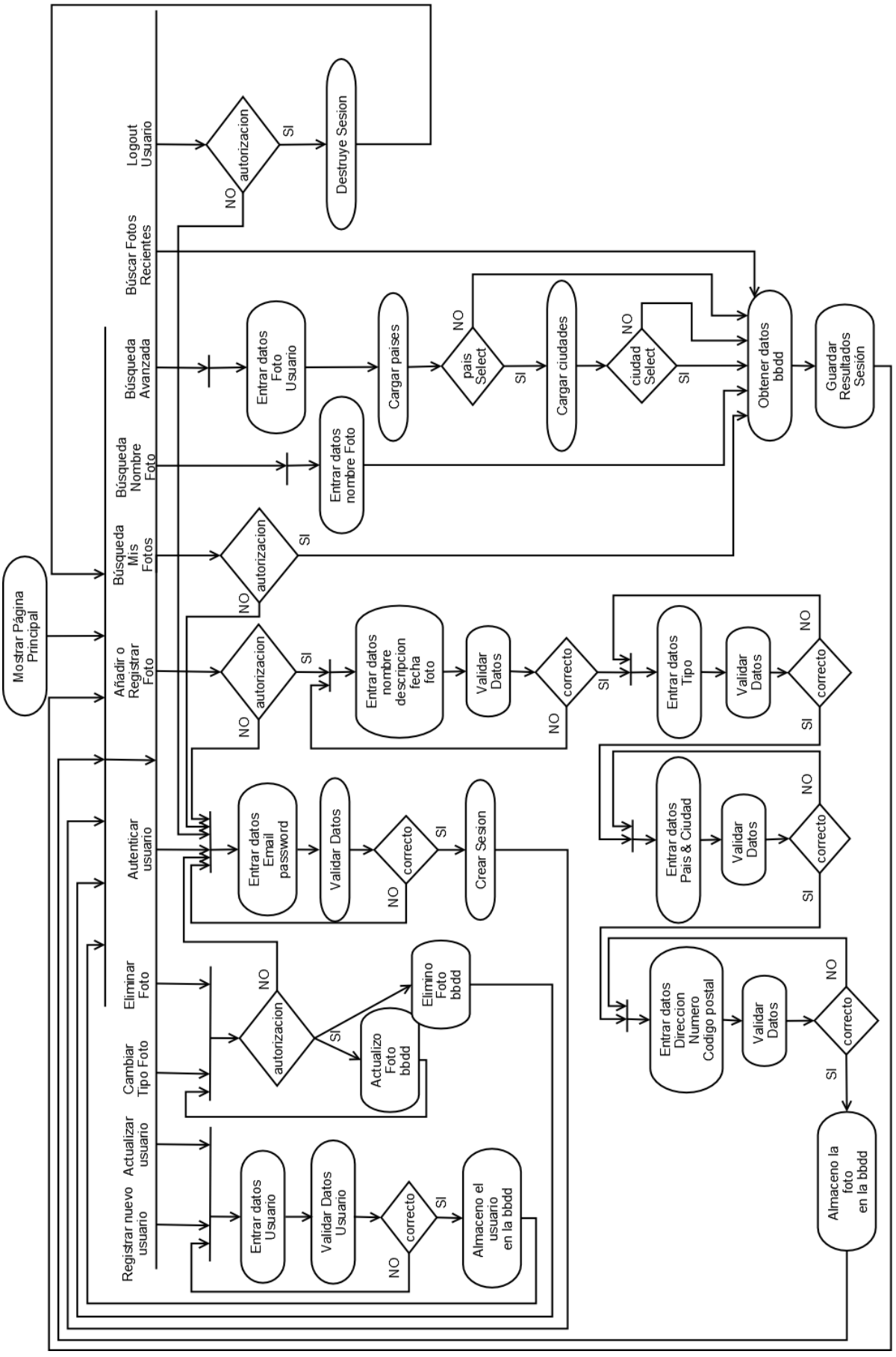


Ilustración 12 : Diagrama de casos de uso

5.3.Diagrama de Actividades



5.4.Integrando Tecnologías a Struts2

Una de las ventajas que tiene Struts2 es la de disponer de una gran cantidad de Plugins que añaden funcionalidades al Framework o que permiten la integración de éste con otros Frameworks como pueden ser Spring y JSF.

Se podría haber hecho una aplicación basada exclusivamente en el Framework Struts2, pero para hacer la aplicación mucho más atractiva y completa, en cuanto a funcionalidades, he integrado Struts2 con varias tecnologías, consiguiendo una aplicación más completa y más cercana a lo que podría ser una aplicación empresarial.

Las tecnologías que he usado para la aplicación Web VisualGate, han sido:

Más sobre Integración de Tecnologías en Struts2:
Ver ANEXO A

5.5.Modelo de Dominio

Cada aplicación necesita clases que representen los conceptos principales de la aplicación que será desarrollada. Estas clases contienen y manejan varias relaciones entre ellas, juntas forman el modelo de dominio de la aplicación.

El modelo de dominio ha sido desarrollado bajo UML – Unified Modeling Language.

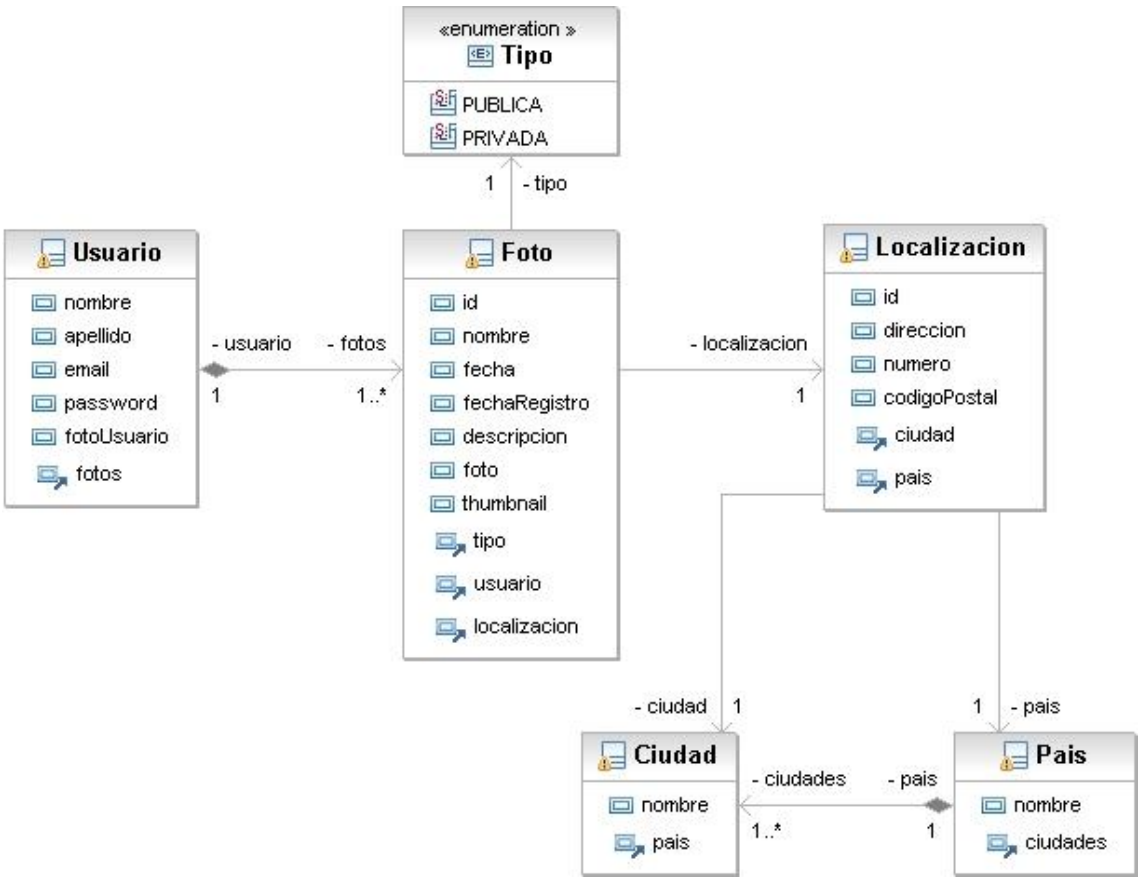


Ilustración 13 : Modelo de Dominio VisualGate

Foto

- Es la clase principal del dominio, contiene toda la información relacionada con la foto, como el usuario y su localización.

Tipo

- Es una "enumeration" y nos es útil para marcar la Foto como pública o privada.

Usuario

- Contiene toda la información del usuario, incluyendo las fotos que ha realizado.

Localización

- Provee información de la localización de la foto, como el país y ciudad donde fue tomada.

Ciudad & País

- Proveen los nombres de países y ciudades con sus relaciones.

Capitulo 6. IMPLEMENTACIÓN DE VISUALGATE

Llega el momento de implementar ***la aplicación web VisualGate***, la cual ***ha sido creada para demostrar el potencial del Framework Struts2*** y como éste se integra con diferentes tecnologías. En este capítulo, describiremos la arquitectura de VisualGate y como las diferentes capas interactúan entre ellas. Nos concentraremos en analizar varios casos de uso con sus respectivos diagramas de flujo, para comprender aún más el funcionamiento interno de Struts2, recorriendo el camino de la petición-respuesta de una acción. También veremos qué nos ofrece Struts2 en cuanto a validaciones, recogida de excepciones, internacionalización de contenido y otras características básicas de un Web Framework. Finalmente veremos cómo hacer que VisualGate siga la tendencia web 2.0 mediante el uso de AJAX, sindicación de contenidos y el uso de Mashups con la API de Google Maps.

6.1. Arquitectura de la aplicación

La arquitectura de la aplicación web VisualGate, se compone de cinco capas, las cuales se muestran en la ilustración 14. A continuación veremos como estas capas se relacionan entre ellas, y que responsabilidades poseen.

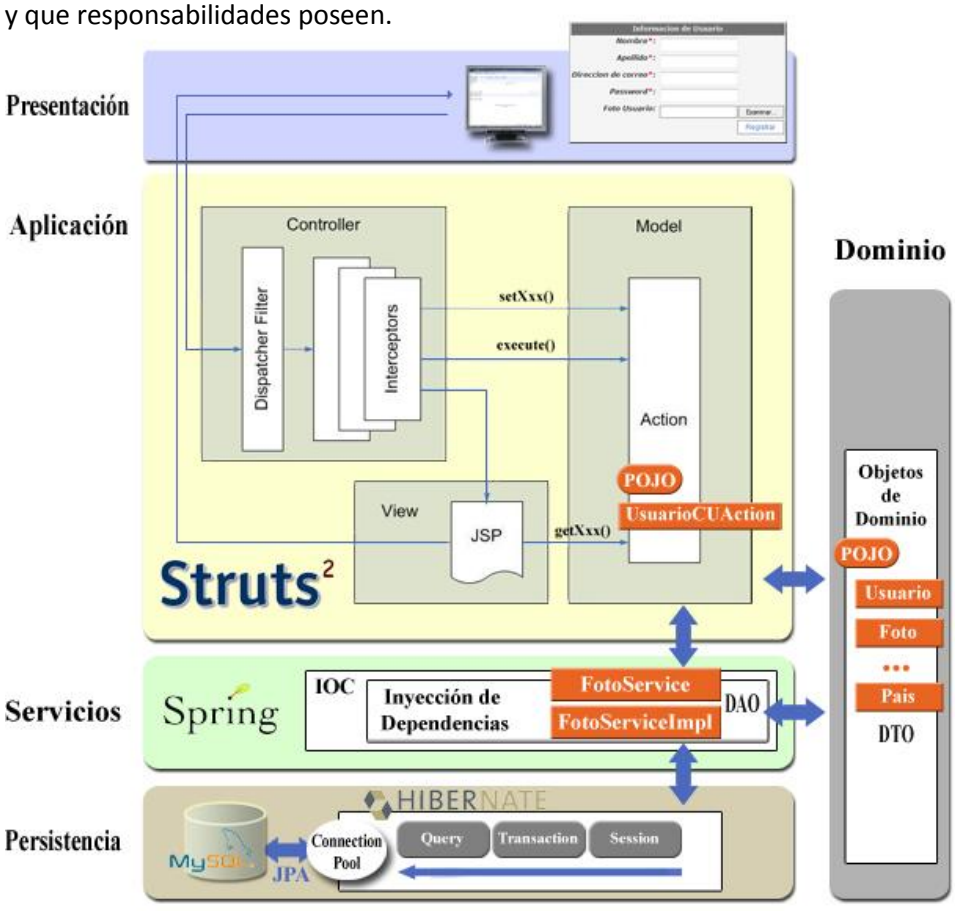


Ilustración 14: Arquitectura de 5 capas - VisualGate

Capa Presentación

- **Responsabilidad**: Interfaz de Usuario
- **Implementación Tecnológica**: JSP / HTML / JavaScript / CSS / Sitemesh

Ésta es la capa con la que interactúa el usuario de la aplicación web, normalmente a través de un browser como puede ser el **Firefox** o el **IE7**. Una de sus tareas, es la de capturar los datos del usuario con los que opera la capa de la aplicación y enviárselos a ésta y por otro lado presentar al usuario los resultados generados.

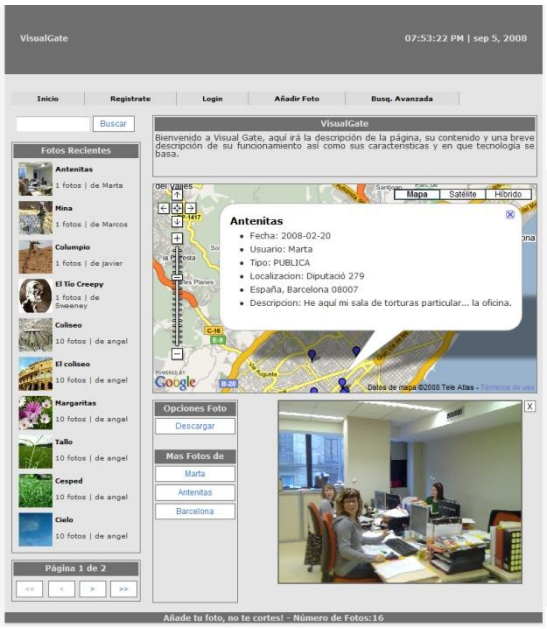


Ilustración 15 : Página principal VisualGate

Esta capa está formada por el conjunto de las páginas que se pueden ver desde el browser. VisualGate dispondrá de una página principal desde donde se podrá ir accediendo a las diferentes opciones dependiendo de los permisos del usuario, en este caso un usuario registrado podrá acceder y realizar acciones que un usuario no registrado no podría hacer.

Cuando un usuario se autentica, se crea una sesión automáticamente con los datos del usuario, esto le permite al usuario acceder a diferentes partes de la aplicación sin tener que preocuparse de autenticarse otra vez, a menos que se desconecte o caduque la sesión según el tiempo fijado.

Para el diseño de las vistas, se han utilizado páginas JSP con etiquetas Struts2, (***TagLibrary***) de las cuales ya hemos hablado con anterioridad, todo esto mezclado con código HTML. Se han utilizado hojas de estilo CSS para formatear el contenido y así aprovechar las ventajas que este sistema nos ofrece, como tener limpio el código HTML y tener en varios ficheros de configuración el aspecto que debe tener la aplicación, permitiéndonos realizar cambios en toda la aplicación únicamente modificando una línea de código en estos ficheros CSS, esto promueve la ***usabilidad y la accesibilidad***.

Volviendo al diseño de la aplicación web VisualGate, se ha utilizado el magnífico Sitemesh para la decoración de las páginas. La necesidad de este Framework de decoración y diseño ha sido requerida para decorar todas las páginas de la aplicación con elementos comunes como pueden ser la cabecera y pie de página, headers, entre otros.

Más sobre Tag Library:
Ver 2.4.4.2

Más sobre Sitemesh:
Ver ANEXO 1.2

Capa Aplicación

- **Responsabilidad**: Flujo de navegación, control de la aplicación, validaciones, conversiones, autentificaciones, interacción con la capa de Servicio y la capa de Presentación
- **Implementación Tecnológica:** Struts2

En esta capa, se encuentra la aplicación en sí. Ésta se encuentra en el servidor a la que acceden los clientes a través de la red utilizando el protocolo http. Podríamos decir que las funciones de la capa Aplicación además de las que nos aporta Struts2 consisten en:

- Recoger los datos enviados desde la capa de presentación.
- Procesar la información, implementar la lógica de la aplicación.
- Acceso a los datos.
- Generar las respuestas para el cliente.
- Interacción con la capa de servicios para obtener inyección de dependencias y acceder a la capa de persistencia.

Es en esta capa donde Struts2 y su patrón MVC entra en juego, así que poco más podemos añadir a lo que ya se explico al principio de la memoria.

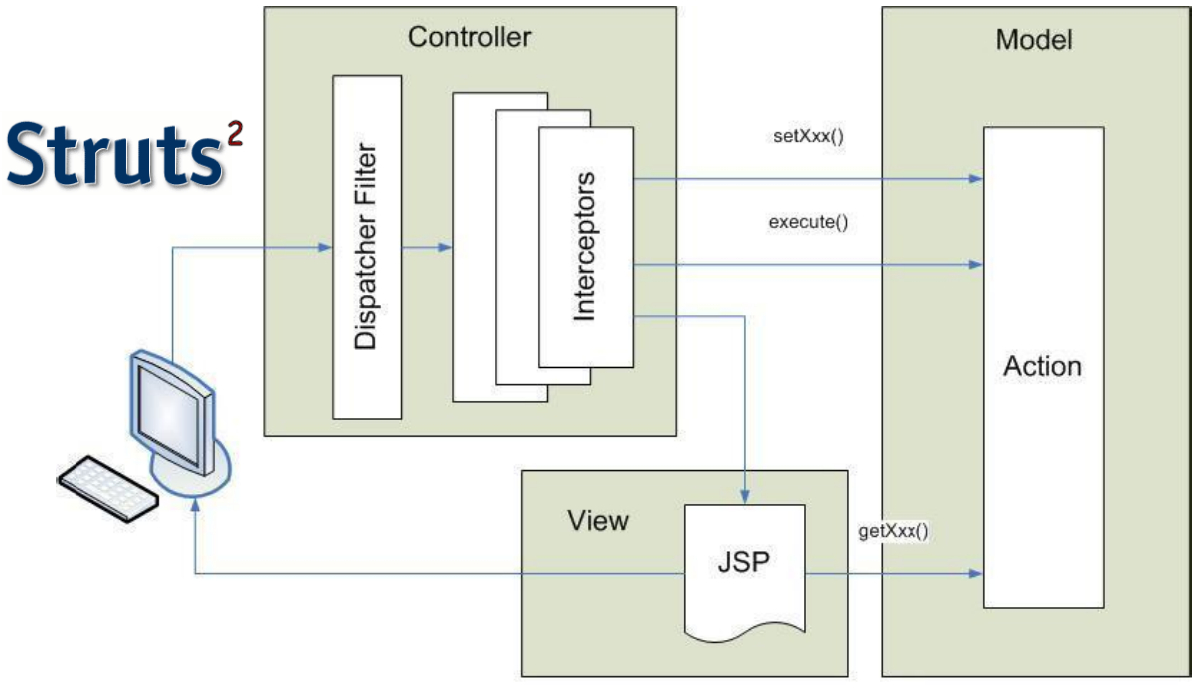


Ilustración 16 : Struts2 MVC

Más sobre el patrón MVC:
Ver 2.3

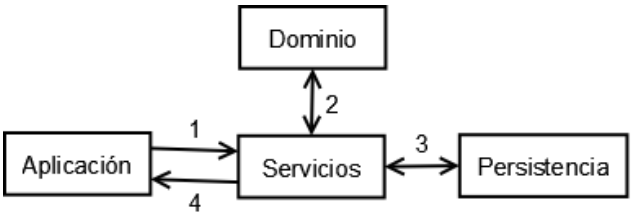
Capa Servicios

- **Responsabilidad**: Proveer servicios de negocio, Inyección de Dependencias.
- **Implementación Tecnológica:** Spring (DAO)

Esta capa actúa como un puente entre las capas de aplicación y persistencia usando la capa de domino como contenedor de información, también llamados DTO (Data Transfer Protocol).

El ***Framework Spring*** es usado para proveernos de estos servicios de negocio, los cuales usan el ***patrón DAO (Data Access Object)*** para acceder a la información de la capa de persistencia mediante una interface, de esta forma conseguimos que nuestra lógica de negocio (aplicación) no sepa nada de Hibernate, y siempre que quiera acceder a los datos lo hará usando ***la interface DAO***, con esto ***conseguimos reducir el acoplamiento*** y así podemos intercambiar la implementación fácilmente si algún día nos cansamos de Hibernate/JPA sin que eso repercuta en el código de la aplicación. La ventaja real de usar Spring en este "puente" de comunicación entre capas, es que los objetos (DTO) instanciados (creados) serán manejados no por Struts2, sino por Spring, con lo que conlleva beneficiarse de la inyección de dependencias.

Un ejemplo básico de cómo funcionaría este sistema, sería lo siguiente:



1. La aplicación necesita recuperar el objeto Usuario de la base de datos, y la capa de servicios nos ofrece una serie de servicios de negocio mediante una interface DAO.

Acción usando la interface DAO (capa de servicios)

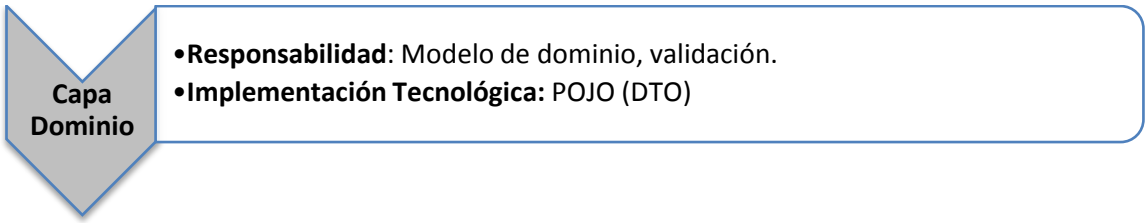
```
…
Resultados = fotoService.buscarFotosPorUsuario(emailId);
…
```

2. Se obtiene (se crea) una instancia del objeto Usuario del modelo de dominio, que ahora se puede considerar un DTO, ya que será usado para transferir los datos de una capa a otra.
3. Se recuperan los datos del objeto Usuario mediante la implementación de la interface DAO (capa de persistencia), y se guardan en el DTO.
4. Finalmente se retorna el DTO a la acción (capa aplicación) que lo invocó, con la ventaja de que este objeto tiene ya todas sus dependencias incluidas.

Más sobre el patrón DAO y DTO:
Ver ANEXO D

Más sobre Spring e Inyección depen.:
Ver ANEXO A.3

Ilustración 17 : Relaciones entre capas



Esta capa, se comunica directamente con las capas de aplicación, servicios y persistencia.

Anteriormente, ya hemos hablado sobre el modelo de dominio, y qué clases lo componen. Ahora hablaremos un poco más en detalle sobre estas clases, y qué peculiaridades tienen, u ofrecen al proyecto, o mejor dicho, a Struts2 en cuanto a validación (Capa Aplicación), y Hibernate en cuanto a persistencia (Capa Persistencia).

Validación

Struts2 usa los objetos de dominio básicamente para dos cosas***, la primera como contenedores vacíos para almacenar información y la segunda para obtener los DTO,*** que sería los objetos de dominio devueltos por la capa de servicios con toda la información y dependencias necesarias.

Como ya hemos dicho, Struts2 usa estos contenedores para seguir el principio ***DRY (Don´t Repeat Yourself)***, como veremos más adelante en el apartado de validaciones de datos. Struts2 hace las validaciones en la acción y no nos interesa añadir ***setters y getters*** que se repetirán en la acción y en el objeto de dominio, es por eso que el objeto de dominio contendrá anotaciones para la validación, ahorrándonos esos ***setters*** en la acción.

Usuario

```
@Entity @Table(name="USUARIO", schema="VISUALGATE")
public class Usuario implements Serializable{

  private String nombre;
…
  @Column(name="NOMBRE")
  public String getNombre(){return nombre;}
  @RequiredStringValidator(message="Error de Validacion", key="..", trim=true)
  public void setNombre(String nombre){this.nombre = nombre;}

…
}
```

Más sobre validaciones en Struts2:
Ver 5.4

Persistencia

Hibernate también usa estos objetos de dominio, ya que serán estos mismos objetos los que serán persistidos en nuestra base de datos relacional, para ello Hibernate ha de conocer como relacionar el objeto con la base de datos, para eso dispone de una serie de anotaciones que utilizará para definir el objeto a persistir.

Más sobre Persistencia (Hibernate):
Ver ANEXO A.4

Capa Persistencia

- **Responsabilidad**: Persistencia de objetos de dominio
- **Implementación Tecnológica:** Hibernate – JPA

La función básica de esta capa, es la persistencia de un modelo de dominio basado en objetos dentro de una base de datos relacional (MySQL) mediante un ORM, que en este caso es Hibernate/JPA.

El uso de una base de datos en el proyecto es esencial para que funcione la aplicación, pero no considero que sea realmente importante incluirla en la memoria como para llegar a explicar cómo funciona y describir la base de datos, ya que este protagonismo se lo lleva Hibernate, el cual será el encargado de acceder, modificar o eliminar registros en la base de datos de Mysql.

La única configuración que requiere MySQL, es su instalación y creación de una base de datos llamada “visualGate”, la cual, se usará para la aplicación VisualGate. La función de creación de tablas y relaciones, está delegada a Hibernate, es por eso, que no se requiere ningún diseño de tablas ni relaciones.

Poco más se puede añadir sobre esta capa, que no se haya dicho ya con anterioridad o que se mencione en los ANEXOS. Así que para saber más en detalle sobre Hibernate y la persistencia de los objetos de dominio en la base de datos relacional ***se aconseja mirar el ANEXO A.4***.

Más sobre Hibernate:
Ver ANEXO A.4

6.2.Implementando los Casos de Uso

VisualGate contiene más de 15 casos de uso, describir en detalle cada uno de ellos llevaría demasiada documentación, así que sólo describiré en detalle un par UC que contenga todo lo necesario como para comprender el funcionamiento completo de la aplicación, desde la llamada a la acción, pasando por sus interceptores, generando el resultado y la comunicación entre las diferentes capas.

6.2.1. Implementando UC1 & UC2

UC1 – Registrar Usuario, aunque parezca muy sencillo, contiene muchas de las características que aporta Struts2 como:

- Acciones.
- Interceptores.
- Results Types.
- Validaciones.
- Excepciones.
- Subida de Ficheros.
- Configuración por medio de Zero Annotations.
- Uso del plugin Codebehind.
- Uso de Tecnologías de integración como Spring y Hibernate.

Además, el caso de uso ***UC2 – Actualizar Usuario*** está muy relacionado con este caso de uso, en realidad ambos usan las mismas clases y JSP´s.

Recordando la descripción de los Casos de Uso 1 y 2, ambos utilizan el mismo formulario para la entrada de datos o modificación en el caso del UC2.

La diferencia será:

UC	Cargar Formulario	Formulario (usuarioR-success.jsp)	Guardando Datos
1	Mostramos el formulario para que el usuario entre los datos necesarios para registrarse.		Validamos, procesamos y almacenamos los datos en la base de datos.
2	Mostramos el formulario con los datos del usuario para que pueda modificarlos.		Validamos procesamos y almacenamos los datos en la base de datos.

6.2.2. Diagrama de Flujo UC1 & UC2

Una vez sabemos qué debe hacer cada caso de uso, podemos diseñar el diagrama de flujo que muestra la relación que tendrán las Actions y las JSP´s.

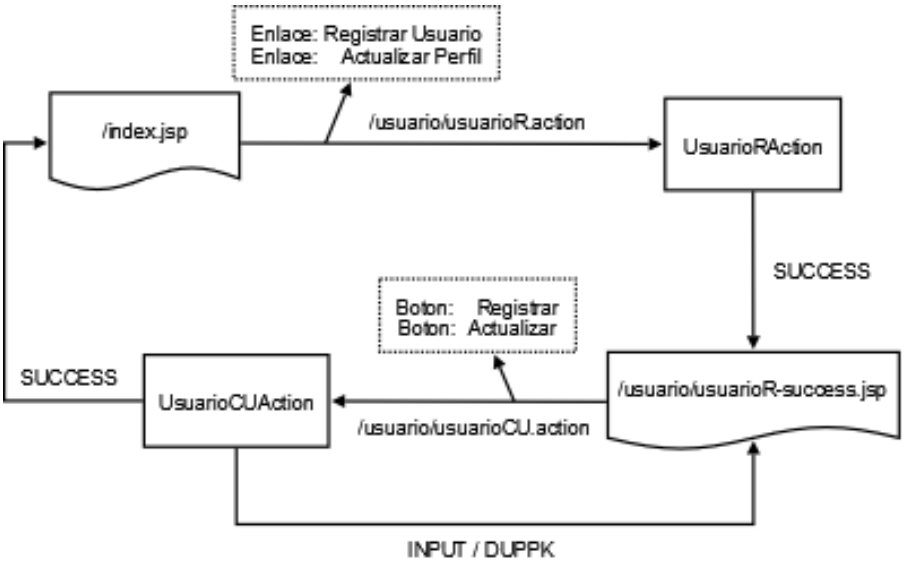


Ilustración 18 : Diagrama de Flujo entre Actions & JSP´s

6.2.3. Caminando a través del UC1 & UC2

He dividido el diagrama de flujo en dos fases, para analizar en detalle el recorrido de la petición-respuesta a través de los interceptores y de las acciones.

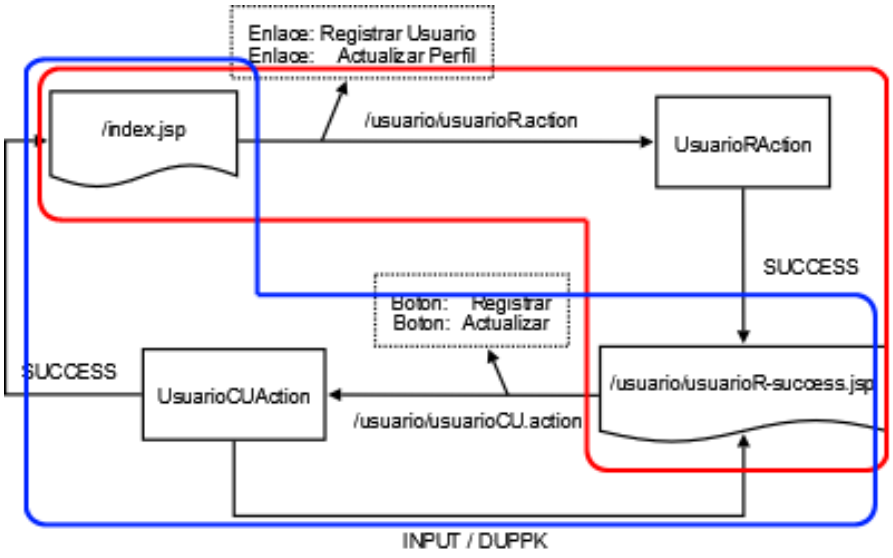
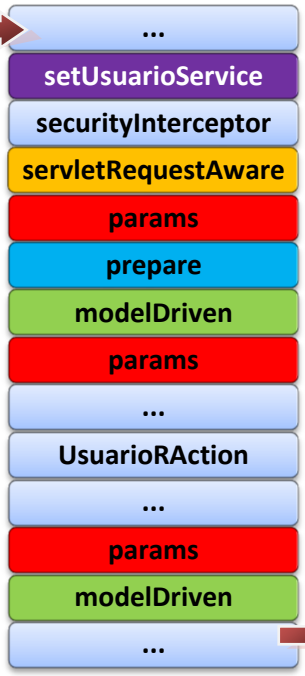


Ilustración 19 : Diagrama de Flujo entre Actions & JSP´s en 2 Fases

Fase1 : Mostrando el Formulario - Registrar Usuario

•El package utilizado para aportar la configuración a la Action es: **base-package**

/usuario/usuarioR.action



Esta fase es muy sencilla, el único interceptor que va a trabajar, será **prepare**, el cual creará una instancia del objeto de dominio **Usuario** en la Action.

Lo siguiente será ejecutar el método **execute()** de la acción **UsuarioRAction**.

Finalmente se devolverá una respuesta y se mostrará el .JSP correspondiente. En este caso **usuarioR-success.jsp**

Más sobre Packages VisualGate:
Ver ANEXO C

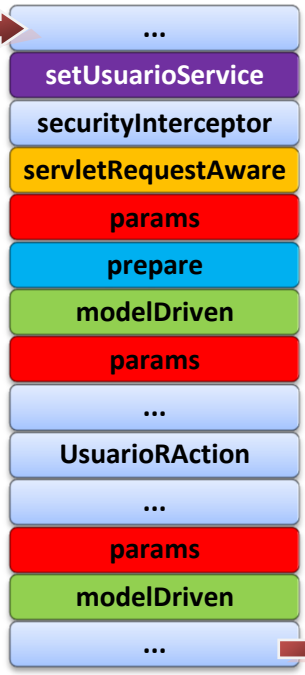
Más sobre Interceptores:
Ver ANEXO B

Más sobre Ciclo Petición-Respuesta:
Ver 2.3.3.1

Fase1 : Mostrando el Formulario - Actualizar Usuario

•El package utilizado para aportar la configuración a la Action es: **base-package**

/usuario/usuarioR.action



setUsuarioService nos crea una instancia del **DAO** mediante **Spring,** el cual le proporcionará **inyección de dependencias**.

servletRequestAware nos proporciona una instancia a **HttpServletRequest** con la cual podemos acceder a la sesión.

El interceptor **prepare**, recupera el email de la sesión del usuario, el cual utilizaremos para recuperar el objeto de la base de datos mediante el **DAO usuarioService.**

El método **execute()** de la acción es ejecutado, y vuelven a ejecutarse los interceptores necesarios, pero en orden inverso.

El objeto usuario obtenido mediante el DAO es colocado en la **ValueStack** por medio del **modelDriven**, y así el formulario será rellenado por los datos del

Más sobre el Patrón de diseño DAO:
Ver ANEXO D

Fase2 : Almacenando los Datos - Registrar Usuario

•El package utilizado para aportar la configuración a la Action es: **base-package**

/usuario/usuarioCU.action

setUsuarioService nos crea una instancia del **DAO** mediante **Spring.**

Prepare, al igual que en la fase 1 creará una instancia del objeto de dominio **Usuario** en la Action.

- ...
- setUsuarioService
- securityInterceptor
- servletRequestAware
- params
- prepare
- modelDriven
- params
- ...
- UsuarioCUAction
- ...
- params
- modelDriven
- ...

Mediante **modelDriven** y **params** se irán insertando los parámetros del formulario en la Action o mejor dicho en el objeto **Usuario** de la Action.

Finalmente el método **execute()** de **UsuarioCUAction** persistirá el objeto **Usuario** mediante el **DAO.**

Finalmente se devolverá una respuesta y se mostrará el .JSP correspondiente. En este caso **usuarioR-success.jsp**

/index.jsp

Fase2 : Almacenando los Datos - Actualizar Usuario

•El package utilizado para aportar la configuración a la Action es: **base-package**

/usuario/usuarioCU.action

Más sobre el Patrón de diseño DAO:

- ...
- setUsuarioService
- securityInterceptor
- servletRequestAware
- params
- prepare
- modelDriven
- params
- ...
- UsuarioCUAction
- ...
- params
- modelDriven
- ...

El proceso será el mismo que el anterior, salvo que el interceptor **prepare**, en vez de instanciar un nuevo objeto usuario, recupera el email de la sesión del usuario, el cual utilizaremos para recuperar el objeto de la base de datos mediante el **DAO usuarioService.**

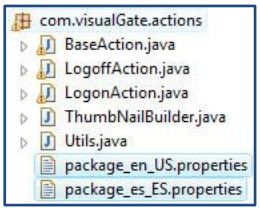
Y mediante **execute()** de **UsuarioCUAction** actualizará el objeto **Usuario** mediante el **DAO**.

/index.jsp

6.3. Internacionalización i18n

La internacionalización provee a la aplicación soporte para diferentes idiomas, mediante Struts2 se puede realizar la internacionalización de forma rápida y fácil.

Lo único que tenemos que hacer es crear un fichero de texto por cada idioma que vayamos a usar y ubicarlo en el package donde se encuentren las acciones, Struts2 lo buscará automáticamente.



Cada fichero de texto, contendrá claves y valores:

package_es_ES.propertes

```
…
usuario.datos=Datos Usuario
usuario.nombre=Nombre
usuario.apellido=Apellido
usuario.email=Direccion correo
usuario.password=Password
usuario.fotoUsuario=Foto
…
```

package_en_US.properties

```
…
usuario.datos=User data
usuario.nombre=Name
usuario.apellido=Surname
usuario.email=Mail address
usuario.password=Password
usuario.fotoUsuario=Photo
…
```

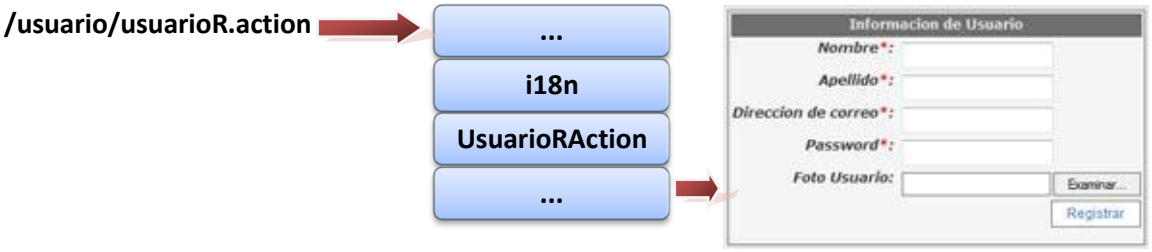
El objetivo es que, dependiendo de la configuración de idioma del browser, se cargue el fichero correspondiente al idioma usado. Otra posibilidad es ofrecer en la aplicación web la selección de idioma.

A la hora de crear los Templates .jsp no escribiremos el texto directamente, sino que usaremos las claves que luego serán traducidas al idioma correspondiente, para eso Struts2 nos provee un atributo que podemos usar en la mayoría de sus Tags, este atributo es **key** :

usuarioR-success.jsp

```
…
<s:textfield key="usuario.nombre"   name="nombre" required="true"/>
<s:textfield key="usuario.apellido" name="apellido" required="true" />
<s:textfield key="usuario.email" name="email"  required="true"/>
…
```

La última condición para poder usar la internacionalización, es que la petición de un .jsp ha de pasar a través de un interceptor, llamado **i18n**. Esto significa que necesitaremos de una Action para poder mostrar el .jsp con internacionalización.

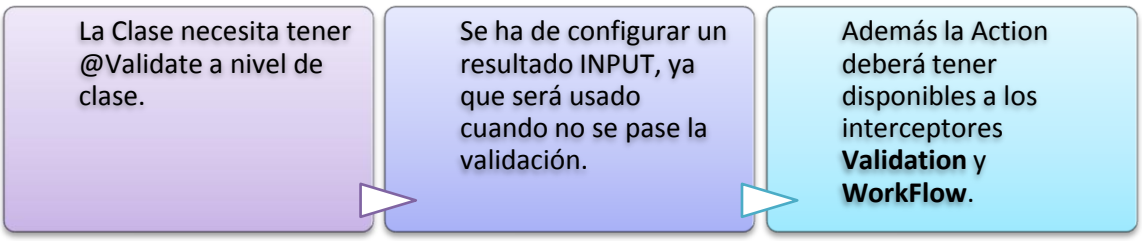


6.4.Validando los datos

Tener almacenada la información en la base de datos es una cosa, pero tener la información almacenada y que sea válida es otra. La importancia de validar los datos antes de almacenarlos, es quizás más importante que tenerlos almacenados.

Struts2 nos provee un sistema de validación robusto y fácil de utilizar, este sistema de validación está basado en Annotations, aunque también podemos usar el sistema mediante ficheros XML.

Para usar las anotaciones en una Action, la clase ha de cumplir una serie de requisitos:



En este ejemplo, veremos cómo se ha de configurar una clase Action y como se usan las anotaciones para validar el campo email.

UsuarioCUAction

```
@ParentPackage("base-package")
@Results({
    @Result(name="success", value="index",…),
    @Result(name="input", value="/WEB-INF/jsp/usuario/usuarioR-success.jsp"…),
    @Result(name="dupPK", value="/WEB-INF/jsp/usuario/usuarioR-success.jsp"…)
})
@Validation
public class UsuarioCUAction extends BaseUsuarioAction {

    @RequiredStringValidator(message="Error de Validacion",
                        key="validate.notEmpty", trim=true, shortCircuit=true)
    @StringLengthFieldValidator(message="Longitud demasiado corta",
                        key="validate.minLength.6", trim=true, minLength="6")
    public void setPassword(String password){this.password = password;}

    public String execute() throws Exception {
        …
        return SUCCESS;
    }
}
```

Más sobre VisualGate Packages: Ver ANEXO C

Más sobre Results: Ver 2.3.3.2

Más sobre Zero Conf: Ver 2.3.4.3

Recuperando el diagrama de flujo entre acciones y jsp´s del **UC1**, nos fijamos en la parte donde ocurre la validación, si ocurre algún problema, la acción retornará **INPUT** y se re direccionará al .jsp o acción que haya sido configurada, en este caso al mismo formulario, mostrando los errores.

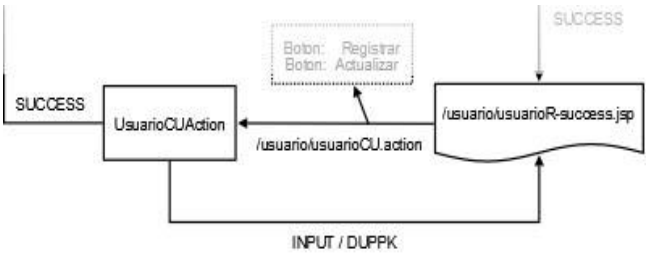


Ilustración 21 : Diagrama de Flujo parcial del UC1 & UC2

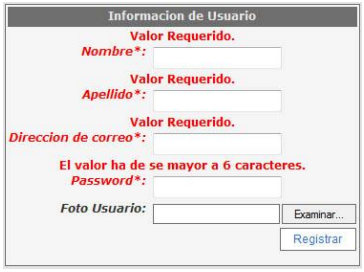


Ilustración 20 : Formulario usuarioR-success.jsp mostrando los errores

Como vemos la Action **usuarioCUAction**, retornará **INPUT** si ocurre algún problema con la validación.

Ahora sabemos que si queremos validar un campo, solo tenemos que incluir una anotación en el setter de la Action, pero qué ocurre cuando estamos usando un objeto del modelo de dominio que contiene **getters** & **setters** de las propiedades como pueden ser las del objeto de dominio **Usuario**.

Como sabemos, incluir estos **setters** por duplicado en la Action sería violar el principio **DRY** (Don´t Repeat YourSelf), por lo tanto lo único que tenemos es una instancia al objeto de dominio, cómo podemos validar entonces estos **setters**?

La solución está en usar **@VisitorFieldValidator,** es uno de los validadores más complejos aportados por Struts2, nos permite usar para cada propiedad del objeto de dominio su propia validación, así que siguiendo el principio **DRY**, colocamos las anotaciones de validación en el objeto Usuario y éstas podrán ser usadas por cualquier caso de uso o acción sobre dicho objeto.

UsuarioCUAction

```
@VisitorFieldValidator(message="Mensaje por defecto",
                       fieldName="model",appendPrefix=false)
public String execute() throws Exception {
    …
    return SUCCESS;
}
```

Usuario

```
public class Usuario implements Serializable{

    @RequiredStringValidator(message="Error de Validacion",
                             key="validate.notEmpty", trim=true)
    public void setNombre(String nombre){this.nombre = nombre;}
    @RequiredStringValidator(message="Error de Validacion",
                             key="validate.notEmpty", trim=true)
    public void setApellido(String apellido){
        this.apellido = apellido;
    }
```

6.5.Recogiendo Excepciones

En cualquier aplicación, las excepciones pueden ocurrir por diferentes razones y circunstancias, estas se dividen en las siguientes categorías:

- Algo inesperado ocurre y sólo el administrador lo puede arreglar.
- Una excepción es usada para cambiar el flujo de trabajo del usuario.
- Un error puede ser solventado interactuando con el usuario.

Como muchas otras características de Struts2, las excepciones son manejadas por medio de los interceptores, más concretamente por el **interceptor exception**. Si queremos utilizar las características de este interceptor, tendremos que tenerlo implementado en nuestra configuración del package.

Errores inesperados

Los errores inesperados, pueden ser definidos globalmente en **struts.xml** usando:

- **global-results**
- **global-exception-mappings.**

De esta manera, conseguimos "disimular" el error a la vista del usuario, mediante un re direccionamiento.

Más sobre struts.xml: Ver 2.3.4.2

Cambiando el flujo de trabajo

En ocasiones nos puede llegar a interesar cambiar el flujo de trabajo del usuario según las circunstancias. Los casos más típicos son las autentificaciones, estas pueden lanzar excepciones como:

- **UsuarioNoAutentificadoException**
- **ActionNoPermitidaException**

Las excepciones corrientes, son reemplazadas por excepciones personalizadas, con nombres de resultados personalizados y JSP´s que son mostradas para marcar el fin del WorkFlow o bien proveer al usuario la oportunidad de continuar, pero por otro camino. Como por ejemplo, mostrando un formulario para la validación del usuario si éste accede a una zona restringida.

Recuperando el error mediante el usuario

Cuando el usuario introduce un email duplicado, una excepción es lanzada por parte de **Hibernate**, en este caso, capturamos la excepción y le asignamos un resultado **DUPPK** y redirigimos el resultado. De esta manera podemos aprovechar y preguntar al usuario si quiere cambiar la dirección del email, o notificarle el error y darle la posibilidad de volver a introducir otra dirección de email.

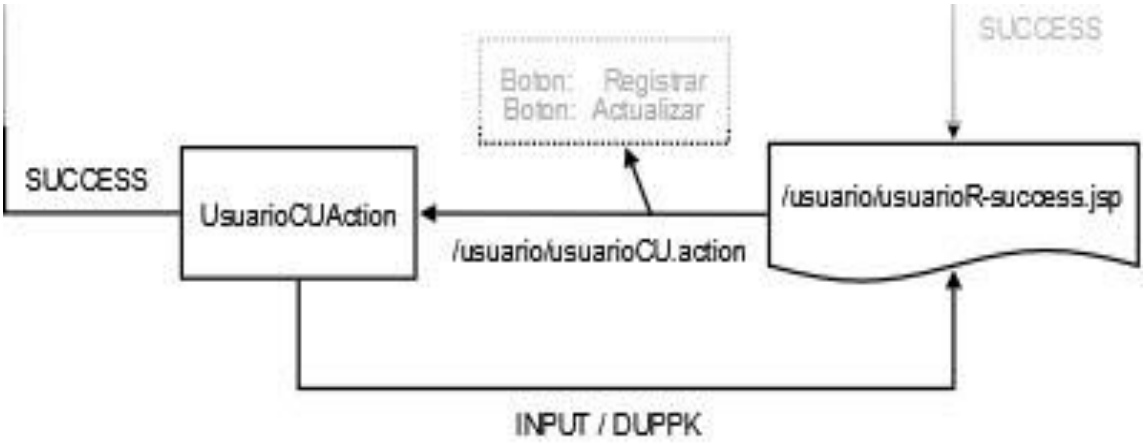


Ilustración 22 : Diagrama de Flujo parcial entre Actions & JSP´s del UC1 & UC2

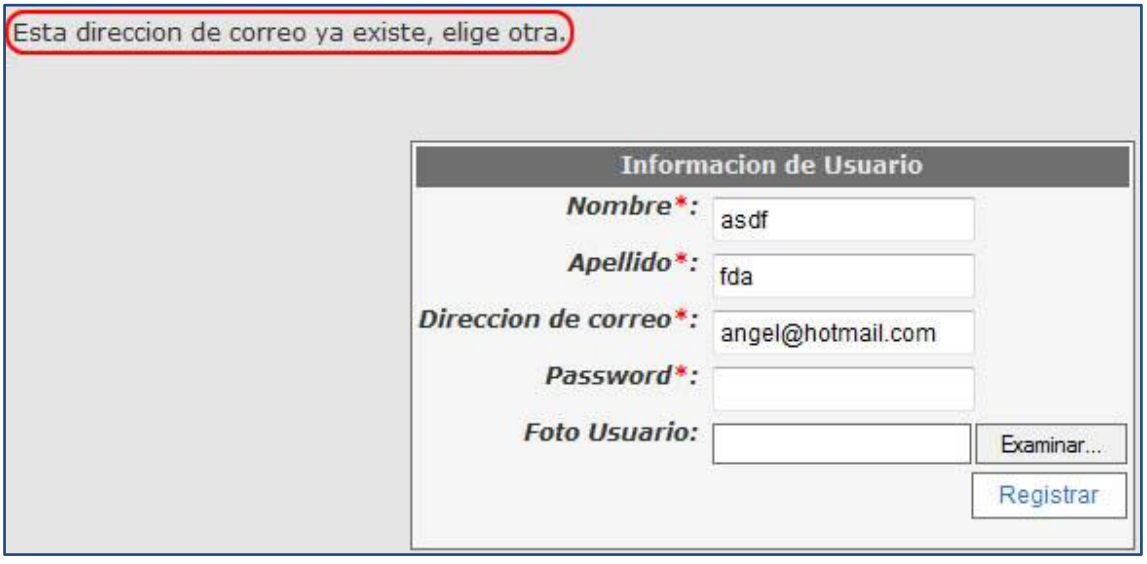


Ilustración 23 : Formulario usuarioR-success.jsp mostrando excepción por Email duplicado

6.6.Wizards & WorkFlows

El Caso de Uso **UC6 – Registrar o Añadir Foto**, se trata básicamente de almacenar en la base de datos una foto añadida por parte de un usuario, con su respectiva información. Como ya comenté éste es uno de los casos de uso más complejos debido a su relación con todas las clases del dominio, eso implica que se tendrá que insertar una cantidad de información más grande de lo normal y en un orden adecuado.

Cuando llegó el momento de diseñar la interfaz del usuario, para hacer realidad este caso de uso, tenía varias opciones:

Almacenar toda la información en un único formulario.

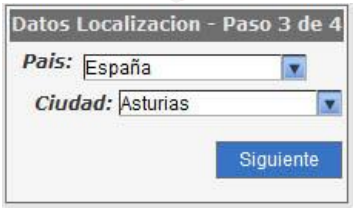
- El problema de este diseño, es que si tenemos campos que han de ser actualizados, todo el formulario será actualizado. Esto se traduce en una pérdida de rendimiento.

Almacenar toda la información en un único formulario, pero usando AJAX.

- Se soluciona el problema del anterior punto.

Una série de formularios pequeños para almacenar la información usando AJAX.

- Se reduce la carga de grandes formularios, y parece ser que los usuarios finales prefieren este tipo de interfaz.



La opción que elegí fue la tercera, ya que me permitía ir construyendo poco a poco el objeto final que sería persistido e ir haciendo las validaciones de cada formulario por separado y sobretodo utilizar el **interceptor scope** que me aportaba Struts2.

Así que finalmente el UC6 se tradujo en un Wizard que consta de 4 pasos.

Más sobre el interceptor scope:
Ver ANEXO B

Más sobre el diagrama de Flujo:
Ver ANEXO E

6.7.Listados y Paginación

VisualGate es una aplicación donde se almacenará una gran cantidad de fotos con su respectiva información, es lógico pensar, que tiene que haber un modo para realizar búsquedas y mostrar los resultados obtenidos. Estos resultados pueden llegar a ser lo suficientemente grandes como para notar una gran pérdida de rendimiento en la aplicación, es por eso que se recurrirá a la paginación de resultados.

Existen varias maneras de implementar la paginación, la opción escogida ha sido:

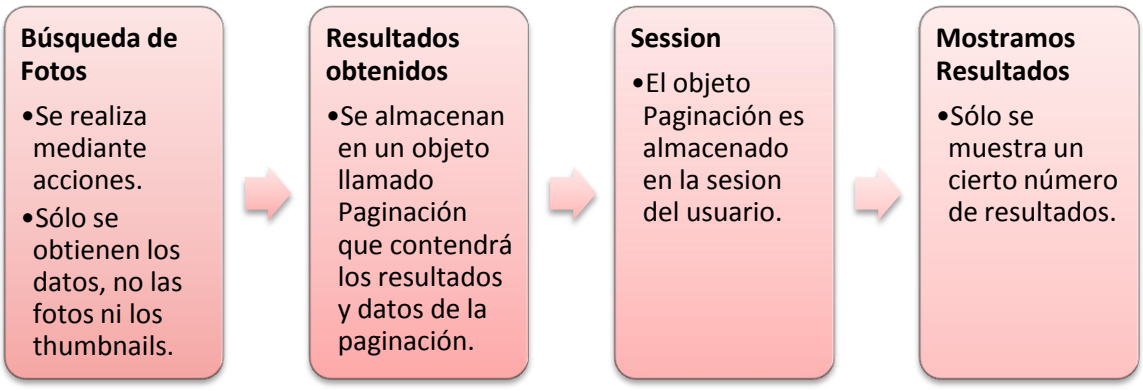
Almacenar los resultados en Sesión

- Los resultados obtenidos de la base de datos son ubicados dentro de "HTTP session". De esta manera el JSP, puede tener acceso a la lista de resultados y decidir qué elementos deberán ser mostrados.

Ventajas	Se reduce considerablemente los accesos a la capa de servicios o la capa de persistencia.
Desventajas	Aumento del uso de recursos en la aplicación del servidor.

En este punto, podemos pensar que al tratarse de búsquedas de fotos en una base de datos, esto conlleve una saturación en los recursos de la aplicación, ya que si nos retornan 1000 resultados con 1000 fotos y se tienen que almacenar en sesión, está claro que algo va a fallar. Este problema ha sido solventado, debido a que solo se recupera y almacena en sesión los datos de la foto y no la imagen (foto). Solo se recuperará la foto cuando sea requerido por el usuario mediante la paginación.

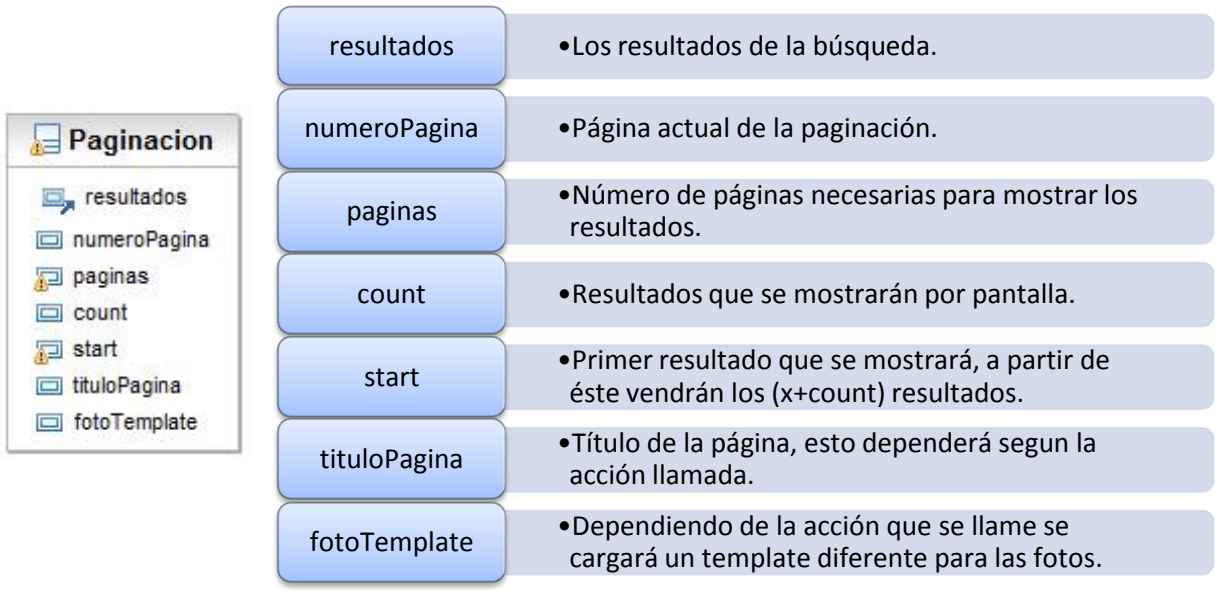
Toda búsqueda realizada en VisualGate seguirá el siguiente proceso:



Lo primero, se realiza una búsqueda con unos parámetros por defecto o fijados por el usuario. Una vez realizada la búsqueda mediante la acción apropiada, obtenemos los resultados que serán almacenados en un objeto especial creado para esta tarea llamado ***Paginación***, un dato importante es que ***no se recuperan las imágenes (fotos) de la base de datos, solo los datos***. El ***objeto Paginación, será almacenado en la sesión*** del usuario, con lo cual podremos acceder a los resultados de la búsqueda y datos sobre la paginación en cualquier parte de la aplicación.

Finalmente mostramos un número concreto de resultados por pantalla fijado por la configuración de la aplicación web.

La clase paginación, constará de las siguientes propiedades:



6.7.1. Modularizando la lista a dibujar usando Templates

Ahora veremos cómo se dibujan los resultados de la búsqueda en la aplicación web. Se ha intentado ***modularizar la página principal index.jsp*** para poder reutilizar el código y también poder ***usar diferentes templates de renderización***. Si observamos el objeto Paginación, contiene una propiedad que será fijada por la acción encargada de recoger los resultados de la base de datos, dependiendo de la acción, el template a usar cambiará y por lo tanto los resultados serán dibujados dependiendo del template. Es el caso del **UC14 – Mostrar Mis Fotos**, el cual utiliza un template diferente para mostrar la información de la foto.

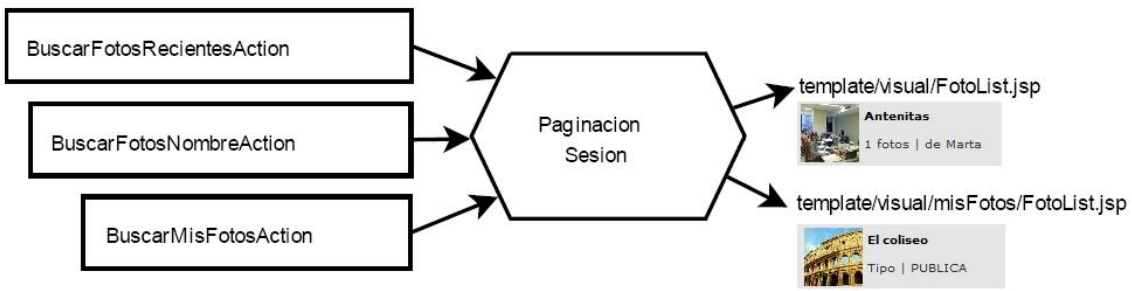
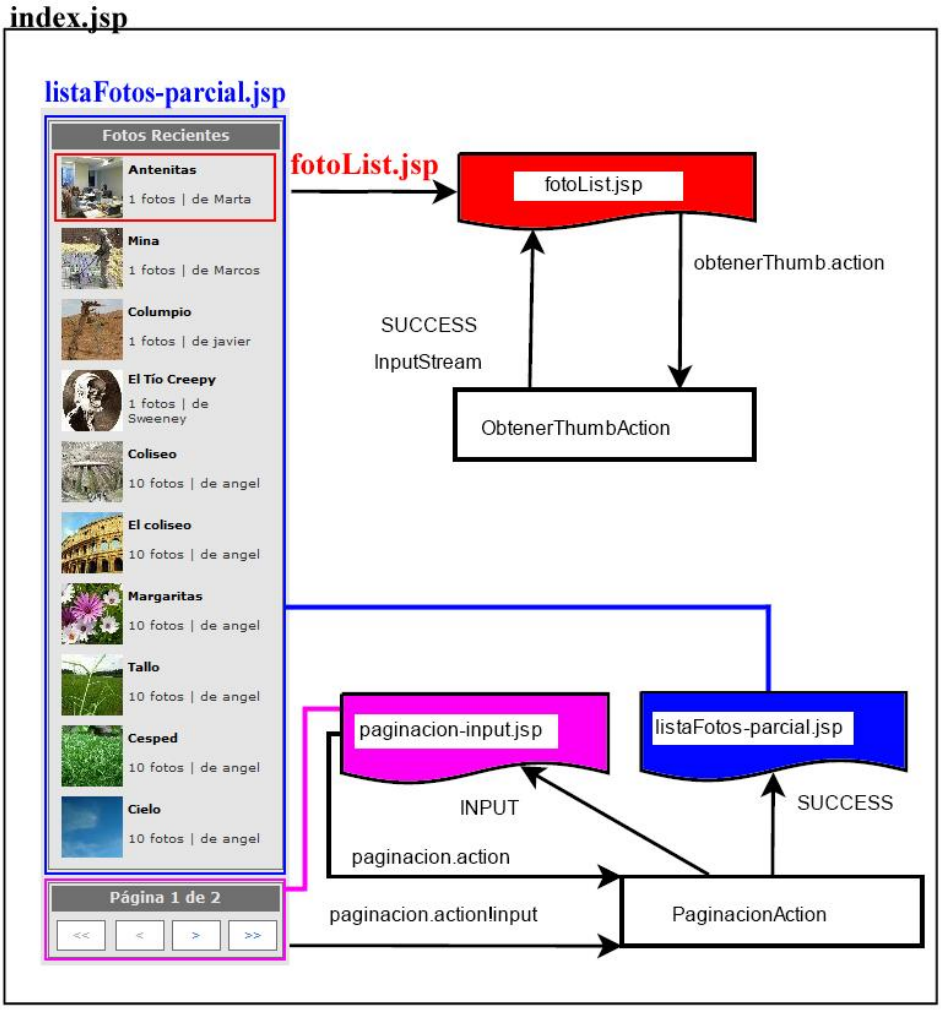


Ilustración 24 : Usando Templates según la Action

Este diagrama de flujo, muestra cómo se relacionan los .jsp y acciones cuando se renderizan los resultados obtenidos por la acción y cómo interviene la paginación en la navegación de los diferentes resultados.



listaFotos-parcial.jsp

- Es el encargado de acceder y mostrar la lista de resultados almacenados en sesión mediante un iterador, cada vez que se acceda a un objeto de la lista, se delegará la responsabilidad de dibujarlo a **fotoList.jsp**

fotoList.jsp

- Su responsabilidad será la de mostrar por pantalla la información del objeto recibido y mediante una acción acceder a la base de datos para obtener el **thumbnail** para mostrarlo junto a la información del objeto foto.

paginacion-input.jsp

- Mostrará una barra de navegación, para poder acceder a las diferentes páginas con resultados. Cada vez que se acceda a una nueva página, PaginacionAction es ejecuta, la cual modificará los datos del objeto paginación, como por ejemplo, aumentar el número de página. Finalmente se redibujará listaFotos-parcial.jsp mediante una llamada asíncrona (AJAX).

6.8.Seguridad

La seguridad es una de esas características que se consideran desde un principio de la aplicación, pero que no se implementan hasta al final del desarrollo, o al menos es lo que suele suceder.

En este apartado se discutirán los atributos de autorización y autenticación, ya que son los atributos necesarios cuando queremos integrar un sistema de seguridad dentro de la aplicación.

Struts2 no proporciona ninguna característica especial para la seguridad, pero sí que tenemos las herramientas como para crearlo por nosotros mismos.

Para implementar un sistema de seguridad, podemos hacerlo de tres maneras diferentes:

Mediante el Contenedor Tomcat

- El servidor de la aplicación o el contenedor de servlets sería el encargado de la autenticación.

Usando librerías externas

- Existen librerías que nos pueden ayudar en la implementación de un sistema de seguridad, como es el caso de ACEGI, el cual nos provee de una amplia gama de configuraciones para la seguridad.

Creando nuestro propio sistema de autenticación y autorización

- La última opción es desarrollar nuestro propio sistema, tiene sus ventajas y sus contras, pero ésta sería la opción que más se ajustaría a nuestro proyecto.

La seguridad no es uno de los principales objetivos del proyecto, y añadir más librerías y depender de mas configuración de integración con otras tecnologías como lo puede ser ACEGI, se distanciaría un poco del objetivo de aprovechar al máximo las cualidades que nos aporta Struts2. Así que la opción que he elegido ha sido la de implementar un sistema de seguridad propio aprovechando las características de Struts2 y sus magníficos interceptores.

Así que la idea será crear un interceptor, que modifique o redireccione el ciclo de vida de una petición de Struts2, dependiendo de si el usuario tiene o no acceso al recurso solicitado, esto solventaría el problema de la autorización.

Para implementar la autenticación, habrá la opción de poderse autenticar mediante el típico formulario de login.

Más sobre el interceptor security y su funcionamiento:
Ver ANEXO B

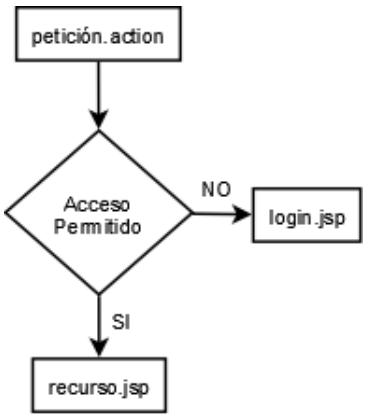


Ilustración 25 : Autorización VisualGate

Ilustración 26 : Autenticación VisualGate

6.9.Google Maps

Hasta ahora, sabemos que un usuario registrado puede añadir o registrar una foto en VisualGate aportando información como su localización. Esta información es visualizada por cualquier usuario que entre en la aplicación.

Para hacer más atractiva la representación de la información de las fotos, se ha incluido un mapa donde se representan cada una de ellas ubicadas en el lugar donde se tomaron.

Para poder llevar a cabo esta funcionalidad, varias tecnologías son requeridas. Por un lado tendremos que hacer servir la **API de Google Maps** que nos permite integrar **Google Maps** en la aplicación web por medio de **JavaScript**. Esta **API** nos proporciona una serie de utilidades para la manipulación de los mapas y, lo más importante, la capacidad de añadir contenido al mapa consumiendo un **RSS feed**, lo que nos permitirá mostrar toda la información necesaria de la foto en el mapa.

Un **RSS feed** es un formato para la sindicación de contenidos de páginas web.

Está basado en XML conforme especificaciones publicadas por el (W3C).

Es relativamente fácil coger un RSS feed y combinarlo con un Google Map para proveerle una representación geográfica de lo que fue originalmente información textual, pero para ello hay que haber generado el RSS feed con anterioridad.

Ahora es cuando Struts2 entra en acción. Como ya sabemos, el ciclo de vida de una petición simple en Struts2 es, básicamente, la petición (.action) y la respuesta que normalmente es un .jsp, pero en este caso tendremos que generar una salida XML, o mejor dicho, un RSS feed y Struts2 no provee ningún tipo de **Result Type** que nos lo genere. Así que habrá que crear uno nuevo.

La Api de Google Maps nos provee de una función especial JavaScript que es capaz de leer el contenido de un XML si le indicamos una URL:

scriptGoogleMaps.js

```
…
// Leo primero el XML y luego genero el mapa.

var url = "http://localhost:8080/visualGate/rss.action";
url = url  + "?" + new Date().valueOf() ;

// Accedo al .xml generado por la action.
GDownloadUrl(url, function (doc, responseCode) {
…
```

La URL será la dirección hacia una acción de la misma aplicación web, donde esta acción será la encargada de generar en el momento de la carga del mashup el XML necesario para que se puedan representar los resultados en el mapa.

Volviendo al **Result Type** personalizado, que he llamado **RssFotoResult** contendrá básicamente la información de la foto y, lo más importante, la longitud y latitud de la foto tomada.

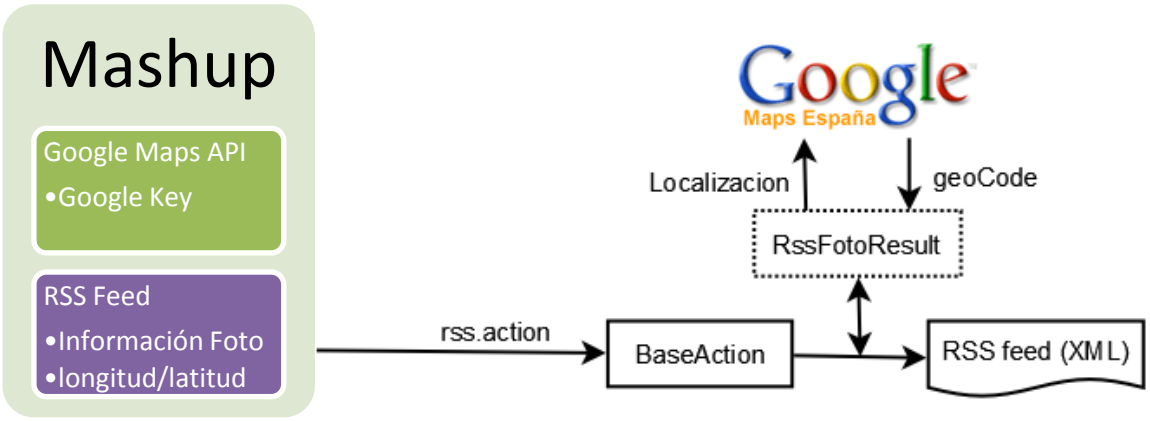
Para obtener la **longitud** y **latitud** de la foto, únicamente necesitamos saber la dirección de la foto, como el país, ciudad, dirección… cuanto más precisa sea la información aportada por el usuario a la hora de registrar la foto, más precisa será su localización en Google Maps.

Para facilitarnos un poco más la vida a la hora de generar el nuevo Result Type, utilizaré las librerías de Rome para la generación del RSS feed (XML).

Más sobre Rome:
Ver ANEXO A

Ejemplo de una Foto - RSS feed (XML)

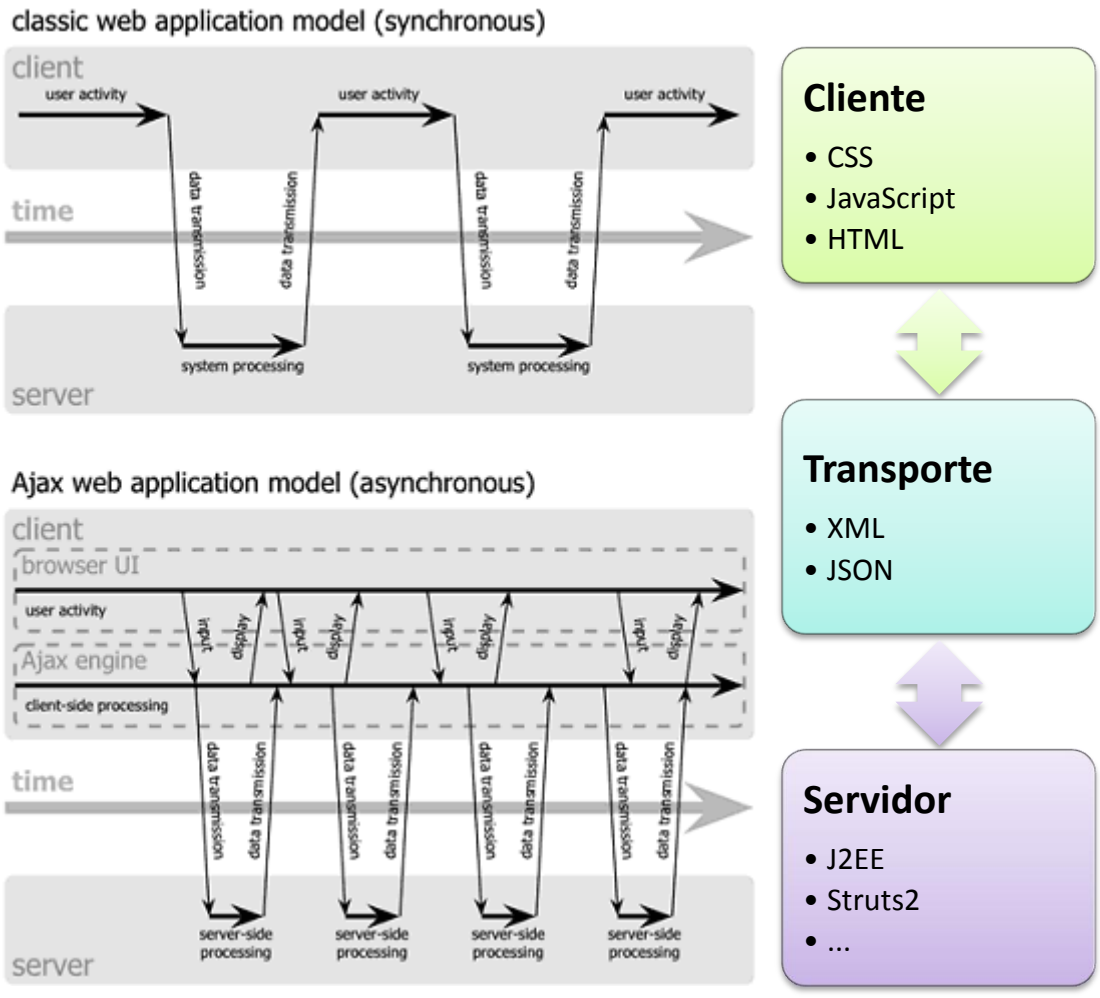
```
…
<item>
  <title>Antenitas</title>
  <link>http://localhost:8080/visualGate/api/foto/18</link>
  <description…</description>
  <pubDate>Mon, 04 Aug 2008 08:38:37 GMT</pubDate>

<guid>http://localhost:8080/visualGate/api/foto/18</guid>
   <dc:date>2008-08-04T08:38:37Z</dc:date>
    <geo:lat>41.391254</geo:lat>
    <geo:long>2.168176</geo:long>
</item>
…
```



6.10. AJAX

AJAX es una técnica de desarrollo web para crear aplicaciones interactivas. Éstas se ejecutan en el lado del cliente, o mejor dicho en el navegador del usuario, y mantiene una comunicación asíncrona con el servidor en segundo plano. Esto tiene la ventaja de realizar cambios sobre la misma página sin necesidad de recargar todo el contenido de la misma. Esto puede traducirse como un aumento de la interactividad, velocidad y usabilidad de la aplicación web.

Esta tecnología está basada en el objeto XMLHttpRequest, el cual es un objeto suministrado por los navegadores webs, accesible mediante JavaScript, usado para transferir datos asíncronos entre el navegador y el servidor Web.



Una de las mayoras mejoras que tiene Struts2 en comparación con su versión anterior Struts y otros Frameworks, es su integración con AJAX. Todos los componentes básicos de Struts2 se han desarrollado de manera que se puedan integrar perfectamente con AJAX, sólo hace falta declarar el Tag con el ”Theme” AJAX. Estos Tags, permiten desde simples actualizaciones a llamadas asíncronas al servidor, y todo ello sin necesidad de programar ninguna línea de código en el cliente.

6.10.1. Usando el Theme AJAX

El Theme de AJAX se implementa usando el mismo mecanismo como cuando usábamos el Theme xhtml que viene por defecto en Struts2. Para proveer características AJAX, Struts2 usa la tecnología Dojo Toolkit. Dojo Toolkit es una de las muchas y diferentes librerías AJAX que pueden llegar a usarse.

Para optimizar la aplicación web VisualGate, se ha utilizado esta tecnología en la mayoría de casos de uso, sobre todo los que están relacionados con las búsquedas y muestra de listados.

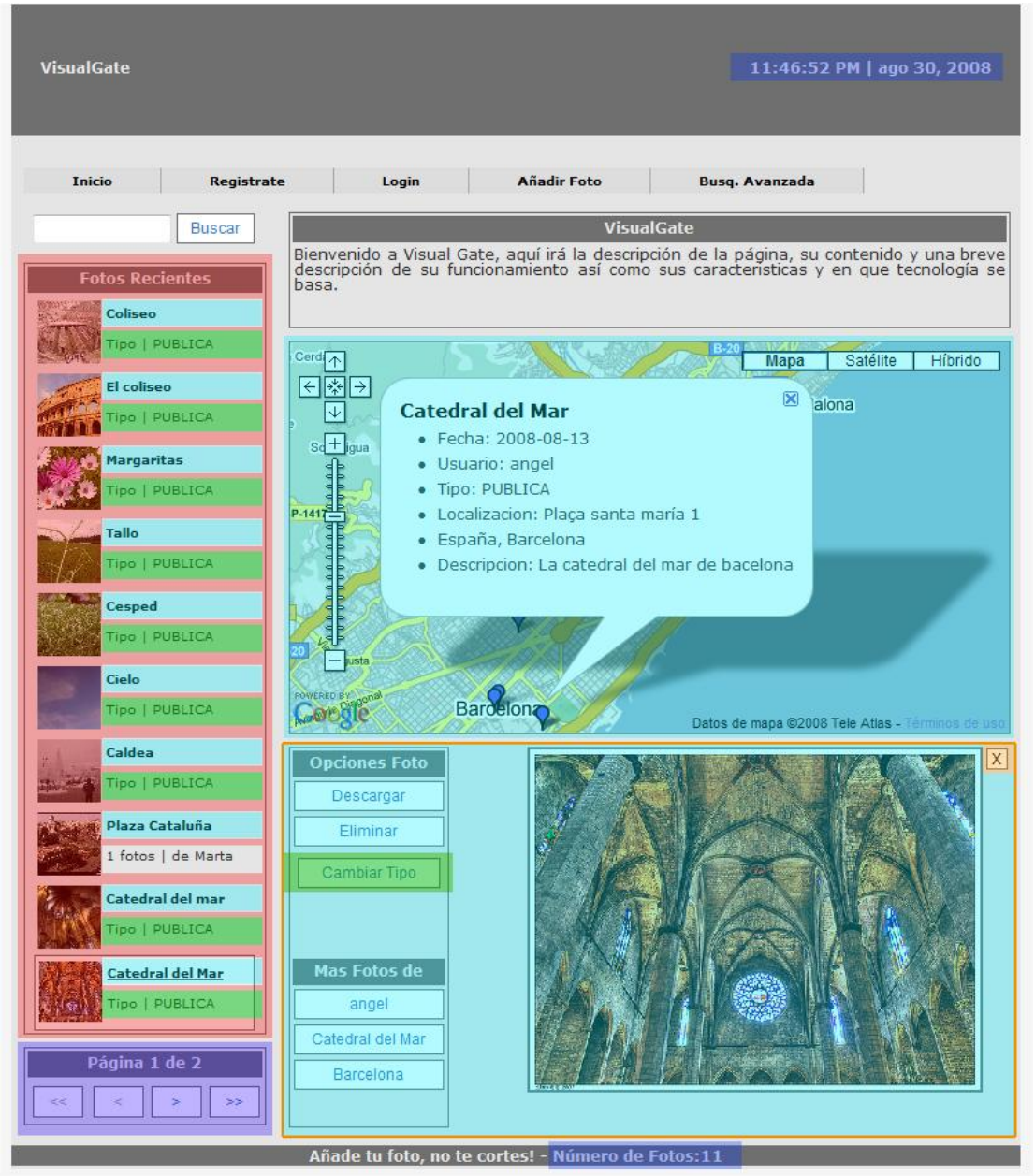


Ilustración 27 : VisualGate con funcionalidades AJAX

A continuación se describen cada uno de los módulos con contenido AJAX, y cuáles son sus funciones y responsabilidades.

Reloj & Fotos bbdd

- Usado para actualizar el contenido cada cierto tiempo, lo he aplicado a un reloj en la parte superior de la página y en la inferior para mostrar el número de fotos que contiene la base de datos.

Paginación

- Actualiza asíncronamente la lista de fotos y a si mismo. Según el botón presionado, se mostrarán las 10 fotos siguientes o anteriores. También se puede acceder al final y al principio de la paginación. En el caso de que no se encuentren mas fotos o se haya llegado al final de la paginación los botones se desactivarán.

Lista de Fotos

- Muestra 10 fotos de la página actual, cada foto provee un enlace para conocer en detalle más información sobre la foto. Este módulo es actualizado asíncronamente por la Paginación.

Nombre Foto

- El nombre de la foto es un enlace que actualiza el módulo inferior derecho donde se muestra la foto con un tamaño más grande y las opciones disponibles. Y por otro lado, muestra en el módulo superior (mapa) dónde está ubicada la foto y su información.

Cerrar Módulo divBuscarFotoDetalles

- Cierra el módulo donde se muestra la foto y sus opciones.

Cambiar Tipo Foto (Privada - Publica)

- Actualiza la propiedad de la foto "tipo" asíncronamente pasando de privada a pública o viceversa. Opción solo disponible si se trata del usuario poseedor de la foto y autenticado.

Además de lo anterior, se ha implementado el **UC10 – Búsqueda Avanzada** por medio de **AJAX,** para proveer del típico formulario de **selección país/ciudad**, pero con una característica añadida por parte de un Tag especial de Struts2 que nos aporta la función de "**autocompletar**".

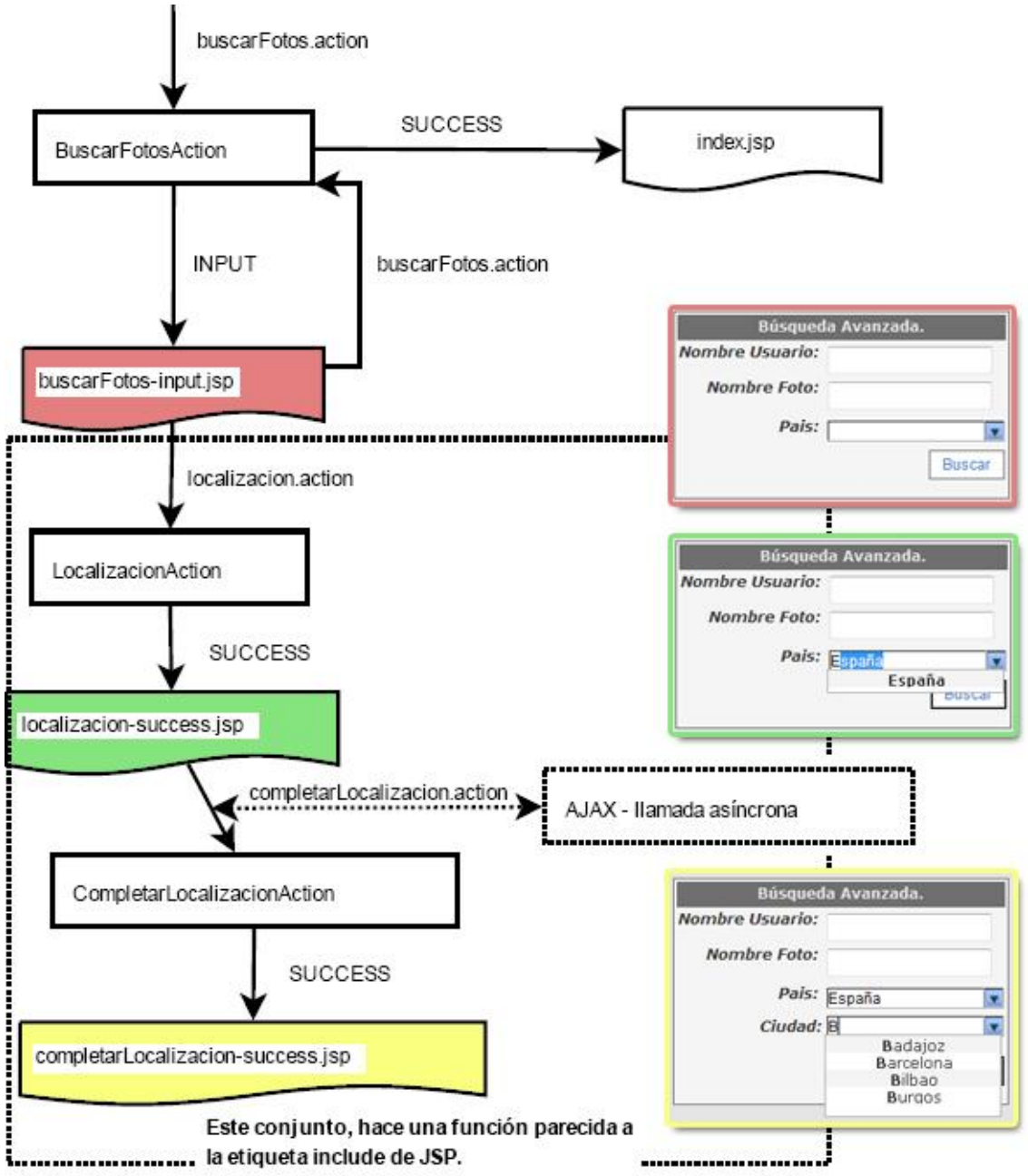


Ilustración 28 : Diagrama de Flujo UC10 - Búsqueda Avanzada + AJAX

El recuadro del diagrama, engloba toda la funcionalidad de "autocompletar + AJAX" de país/ciudad en una acción, llamada "**localización.action**". He utilizado una etiqueta especial de Struts2 que hace una función parecida al típico include de JSP, para que pueda reutilizar el código en otros casos de uso, como es el caso del **UC6 – Añadir o Registrar Foto,** donde también se ha de seleccionar una ciudad y país para poder registrar la foto correctamente y almacenarla en la base de datos.

6.11. Sindicación

RSS es un sencillo formato de datos para redifundir contenidos a suscriptores de un sitio web. El formato permite distribuir contenido sin necesidad de un navegador, utilizando un software diseñado para leer estos contenidos RSS, aunque es posible utilizar el propio navegador para leer estos contenidos.

Ilustración 29 : RSS Feed

La sindicación pertenece al caso de uso, UC15 – Sindicación de contenido web, en VisualGate ya habíamos utilizado un RSS para que fuera consumido por un Mashup junto a la api de Google Maps, para poder representar la localización de la foto mediante un mapa. Ahora aprovecharemos ese RSS Result Type que habíamos creado expresamente, para la sindicación.

Ilustración 30 : Subscripción de RSS a VisualGate

Cualquier usuario que entre en nuestra aplicación web, podrá subscribirse a nuestro contenido de fotos, y así estar informado de las nuevas fotos añadidas en la aplicación sin necesidad de acceder directamente a la página web.

Lo último que necesita el usuario final, es disponer de un "reader" que lea este contenido, en mi caso uso un gadget para la sidebar de Windows Vista, capaz de leer las fuentes RSS.

Ilustración 31 : Reader accediendo a las fuentes de VisualGate desde el escritorio

Capitulo 7. PRUEBAS Y COPIAS DE SEGURIDAD

7.1.Pruebas Realizadas

Este punto es de vital importancia en cualquier proyecto, pese a que en ocasiones hay quien no les atribuye la importancia que merecen. Las pruebas pueden determinar el éxito o fracaso de un producto final, ya que deben proporcionarnos garantías sobre el funcionamiento correcto de la aplicación.

Para asegurar la estabilidad y robustez, se han realizados pruebas funcionalidad por funcionalidad, tanto en el desarrollo individual de cada una como en una etapa específica en la que se ha probado todo el sistema en conjunto. Por cada operación se ha intentado tratar con todos los casos posibles, a fin de garantizar que el resultado no permita situaciones que escapen al control del usuario final, como ya comentamos en el apartado de recogida de excepciones de la aplicación.

Conociendo la operación a probar, se han introducido datos para producir excepciones, o mejor dicho, que nosotros sabemos que deben producirlas, y se ha comprobado que la aplicación las controla y no se produce ninguna situación extraña. A continuación, se han probado toda una serie de casos posibles y reales observando el comportamiento y corrigiéndolo cando ha sido necesario.

A continuación vamos a mostrar un par de ejemplos de pruebas realizadas:

- Cuando un Usuario Registrado, trata de añadir una foto a la aplicación, deberá realizar una serie de pasos donde se le mostrarán varios formularios para la entrada de datos. Algunos de estos campos son campos obligatorios, los cuales han de ser rellenados obligatoriamente, además de esto, se controlará que la fecha insertada sea la correcta. Si ocurriera algún problema, la aplicación trataría el problema devolviendo al usuario, al formulario donde introdujo los campos erróneos mostrando el error producido, de una manera correcta y suave.
- Si hay un usuario que intenta acceder a un recurso, al cual no tiene suficientes permisos para acceder, el ciclo de vida de la petición será modificado, enviándole a la página de login, donde podrá autenticarse para poder acceder a dicha zona.

Para poder llevar a cabo todas estas pruebas, se ha utilizado JUNIT, el cual permite realizar pruebas unitarias.

Más sobre JUNIT – Pruebas Unitarias:
Ver ANEXO G

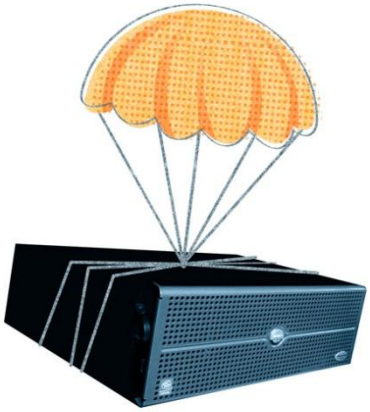
7.2.Copias de Seguridad

Otra parte de vital importancia, aunque no por los mismos motivos, es el sistema de backup utilizado.

La finalidad del backup es guardar una copia periódicamente de una serie de datos, por ejemplo, durante un proyecto podría ser el trabajo desarrollado, con tal de evitar tener que rehacer cosas en caso de un fallo en el sistema (un virus que elimine los datos del disco duro de la máquina donde trabajamos, una subida de tensión que dañe físicamente el disco duro, etc).

En este proyecto se ha establecido un sistema de backup que consiste en lo siguiente:

- Regularmente se copiaban los ficheros del código de las aplicaciones y la documentación en un disco duro secundario, totalmente independiente del sistema donde se trabajaba.
- Cada vez que se hacían cambios en la base de datos o había una cantidad considerable de datos para las pruebas se copiaba también el contenido y la Base de datos en sí en dicho disco duro.
- Se realizan periódicamente copias de seguridad de la base de datos de MySql, programadas en el mismo gestor de MySql para cada día por la noche, en horas de mínima carga para el servidor. De esta manera en caso de desastre o similar se podrían recuperar los datos de la base de datos.



Capitulo 8. CONCLUSIONES Y OBJETIVOS ALCANZADOS

Con este proyecto se ha querido diseñar, una aplicación web que siguiera la tendencia web 2.0. Para ello se ha estudiado la manera de realizar un proyecto con una puesta en marcha lo más rápida posible, utilizando tecnologías de desarrollo web actuales y fáciles de usar, o mejor dicho, que ayudaran al desarrollador y no lo contrario.

La verdadera razón de crear este proyecto, era la de acercarme a las nuevas tecnologías de desarrollo de aplicaciones web, ya que siempre me ha atraído bastante este tema o si se puede decir, este mundillo.

Inicialmente, no conocía mucho sobre este mundo tecnológico, aunque ya había tenido experiencias con PHP antes de empezar la ingeniería técnica, pero sin seguir ningún tipo de "buenas prácticas". Esto fue una de las razones por las cuales me decidí a realizar este proyecto. Quería demostrar todo lo aprendido durante el desarrollo de la carrera, aplicando mis conocimientos en un proyecto el cual representara en mayor medida todo lo aprendido, y una vez acabado, no se tuviera un sentimiento de desorden o de vagancia a la hora de realizar modificaciones por no haber utilizado correctamente buenos diseños o no seguir principios como el DRY (Don´t Repeat YourSelf).

Para la realización completa del proyecto se ha tenido que dedicar mucho tiempo en la investigación, pero sobretodo en el aprendizaje de las múltiples tecnologías que se han implementado. El tiempo total invertido en esta parte, ha sido muy superior a la realización práctica del proyecto, es por eso, que quiero remarcar el esfuerzo empleado en aprender y hacer funcionar estas tecnologías conjuntamente, más que en el contenido de la aplicación web, ya que ésta solo ha sido un medio para poder utilizarlas.

El comienzo fue muy duro, ya que como se ha comentado, existen varias tecnologías que me solucionaban el problema, lo único que tenía claro era el lenguaje de desarrollo J2EE. Inicialmente, no empecé con Struts2, si no con Java Server Faces, y seguidamente por Struts1.x, además de nuestros amigos los Servlets, hasta que finalmente conocí a Struts2. Aunque gasté mucho tiempo en estudiar y realizar prácticas con diferentes Frameworks antes de empezar con Struts2, me aportó una visión más clara de cómo cada Framework veía de manera distinta el desarrollo de aplicaciones web, unos lo basaban en acciones y otros en componentes, aunque al final todos resolvieran el mismo problema.

La intención de usar Hibernate junto a Spring para las capas de negocio y persistencia, fue un poco mareante al comienzo, sobre todo por falta de conocimientos. También me gustaría comentar el quebradero de cabeza que me ha producido implementar la Api de Google Maps en la aplicación, no por su complejidad, sino por incompatibilidades que existen entre los diferentes browsers a la hora de consumir los RSS feed, aunque esto también ocurre en la tecnología CSS.

Finalmente, mirando el proyecto una vez acabado, puedo decir, que me ha aportado una gran cantidad de conocimientos en tecnologías, patrones y buenas prácticas que antes desconocía totalmente, a esto hay que unir la utilización del Framework Struts2, una tecnología que pese a

tener una curva de aprendizaje moderada, ésta se convierte en alta cuando juntamos varios Frameworks y tecnologías para que trabajen conjuntamente. Aún así, cuando consigues hacerlo funcionar correctamente, y lo más importante, entiendes el funcionamiento, el sentimiento de autorrealización aumenta considerablemente.

Quizás la única cosa negativa que saco de todas estas tecnologías de Frameworks, es la cantidad de formas que existen para poder afrontar un mismo problema, en mi opinión esto es un atraso a la hora de desarrollar una aplicación empresarial en grupo, ya que si cada desarrollador lo hace de una manera diferente, esto repercutirá en la mantenibilidad del proyecto.

8.1.Líneas Futuras

El futuro de VisualGate, depende de su publicación en Internet. Según la decisión tomada, habría que valorar diferentes aspectos como son:

- Contratar un dominio.
- Contratar un servicio de Hosting.
- Acabar de perfeccionar la seguridad, incluyendo encriptación de passwords.
- Negociar la inclusión de publicidad en la web.
- Incorporar nuevas secciones y contenidos que mejoren la experiencia del usuario.
- Incorporar un sistema de votaciones para las fotos.
- Añadir compatibilidad de la web a dispositivos móviles.

Éstos son algunos ejemplos, aunque seguramente hay muchos más.

BIBLIOGRAFÍA

Wikipedia, .J2EE, Wikipedia.org

http://es.wikipedia.org/wiki/J2EE

Wikipedia, .W3C, Wikipedia.org

http://es.wikipedia.org/wiki/W3C

Wikipedia, .Web 2.0., Wikipedia.org

http://es.wikipedia.org/wiki/Web_2.0

Wikipedia, .Hojas de estilo en cascada, Wikipedia.org

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

Wikipedia, .JavaScript, Wikipedia.org

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

Wikipedia, .AJAX, Wikipedia.org

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

Wikipedia, .Apache Struts, Wikipedia.org

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

Struts2 – The Apache Software Foundation.

http://struts.apache.org/2.x/

Hibernate – Hibernate Home page

http://www.hibernate.org/

proactiva-calidad.com, Patrón DAO

http://www.proactiva-calidad.com/java/patrones/DAO.html

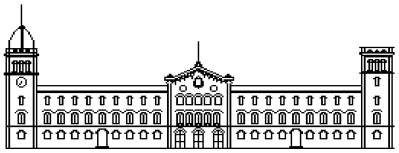
Libros Consultados

Ian Roughley, 2007, Starting Struts 2, InfoQ.

Ian Roughley, 2007, Practical Apache Struts2 Web 2.0 Projects, Apress.

Christian Bauer, Gavin King, 2008, Java Persistence with Hibernate, Manning.

Donal Brown, Chad Michael Davis, Scott Stanlick, 2008, Struts2 In Action, Manning.



Trabajo final de carrera

INGENIERÍA TÉCNICA EN INFORMÁTICA DE SISTEMES

Facultad de Matemáticas
Universidad de Barcelona

ANEXOS

Ángel Gómez García

Director: Jesús Cerquides Bueno
Realizado a: Departamento de Matemáticas Aplicada i Análisis. UB
Barcelona, 25 de septiembre de 2008

ANEXO A. Integrando Tecnologías a Struts2

1. Codebehind Plugin

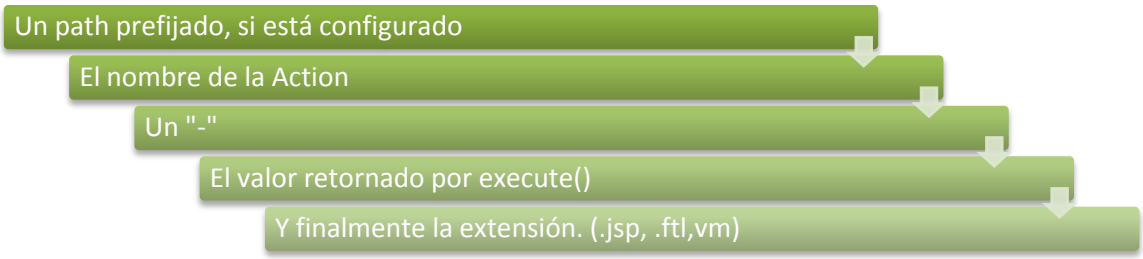
Codebehind es un Plugin muy útil para Struts2, ya sabemos que uno de los objetivos de Struts2 es hacer la vida mas fácil a los desarrolladores web, y uno de los grandes problemas que tienen la mayoría de Frameworks, es la gran cantidad de archivos de configuración necesarios para hacer correr la aplicación. Struts2 pone remedio a esto, utilizando algunas técnicas de ahorro de código como "Zero Annotations" entre otras.

Codebehind Plugin

- **Publisher**: Apache Software Foundation
- **License**: Open Source
- **Version**: Bundled with Struts2 - struts2-codebehind-plugin-2.0.11.2.jar
- **Homepage**: http://struts.apache.org/2.x/docs/codebehind-plugin.html

El Plugin Codebehind nos propone una reducción de código considerable si seguimos una convención para la generación de resultados de las acciones.

Cuando el Plugin está activado y no existe un resultado configurado para la acción como puede ser una configuración XML o basada en "Annotations" el nombre de la plantilla (JSP, FreeMarker, Velocity …) a mostrar es construido usando una convención especifica.



En cuanto a la configuración necesaria para poder usar Codebehind, se han de añadir dos nuevas constantes al fichero de configuración Struts.xml.

- El primero provee el nombre del package que contendrá las acciones que usarán este Plugin.
- Y el segundo, es especificar el path donde se deberán buscar los Templates.

struts.xml

```
…
<constant name="struts.codhind.defaultPackage" value="base-package" />
<constant name="struts.codebehind.pathPrefix" value="/WEB-INF/jsp/"/>
…
```

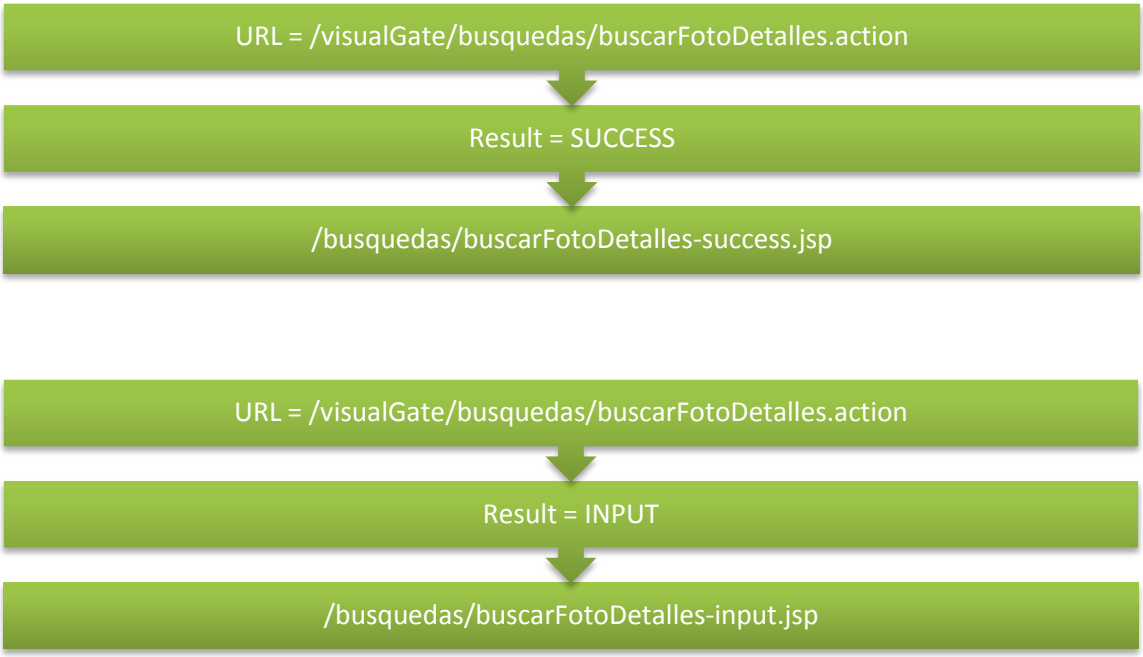
Ejemplo SIN Codebehind:

```
@ParentPackage("base-package")
@Results({
      @Result(name="success",
                  value="/WEB-INF/jsp/busquedas/buscarFotoDetalles-
success.jsp",
                  type=ServletDispatcherResult.class)
          })
public class BuscarFotoDetallesAction extends BaseBusquedasAction {
      …
}
```

Ejemplo CON Codebehind:

```
@ParentPackage("base-package")
public class BuscarFotoDetallesAction extends BaseBusquedasAction {
      …
}
```

Así que siguiendo la convención…



2. Sitemesh Plugin

Sitemesh es un Framework de decoración para páginas web, recomendado para la creación de grandes sitios web que contengan una gran cantidad de páginas, las cuales requieren reproducir los mismos contenidos en diferentes páginas como sistemas de navegación.

Sitemesh Plugin

- **Publisher**: Apache Software Foundation
- **License**: Open Source
- **Version**: Bundled with Struts2 - struts2-sitemesh-plugin-2.0.11.2.jar
- **Homepage**: http://struts.apache.org/2.x/docs/sitemesh-plugin.html

Una de las ventajas que tiene Sitemesh en cuanto a competidores como Tiles para Struts o Faceletes para JSF, es sin lugar a dudas su sencilla configuración. Otra de las ventajas que aporta Sitemesh es que es totalmente independiente del Framework usado, tanto se puede usar en Struts, Stripes, JSF, Struts2 entre otros.

Esta última característica se asemeja mucho a uno de los principios de Struts2, que es reducir el acoplamiento.

SiteMesh utiliza un Filter que se encargará de interceptar la respuesta de la Action y añadirle el HTML adicional al resultado generado por Struts2. Imaginemos que obtenemos un simple .jsp que retorna un "Hello World", por lo general, la petición de la página viene del servidor, la página es dibujada y a continuación los resultados son devueltos al browser del cliente. Es en este preciso momento es cuando actúa Sitemesh, realizando un proceso adicional antes de devolver el .jsp al browser.

Ilustración 32: Funcionamiento Sitemesh

Finalmente, podemos ver la diferencia de usar Sitemesh en la aplicación web VisualGate, además de aplicar los CSS.

Ilustración 33 : Formulario sin Sitemesh

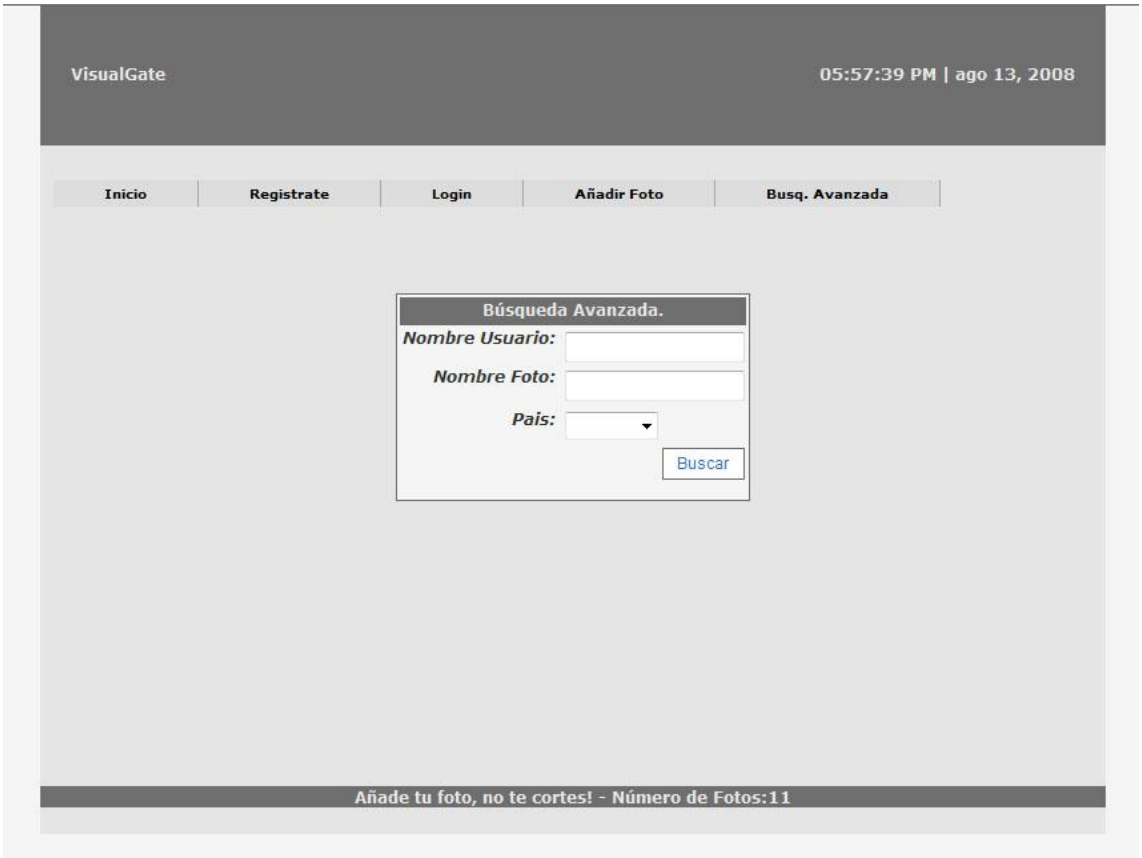


Ilustración 34 : Formulario con Sitemesh

3. Spring Plugin

Spring Plugin

- **Publisher**: Apache Software Foundation
- **License**: Open Source
- **Version**: Bundled with Struts2 - struts2-spring-plugin-2.0.11.2.jar
- **Homepage**: http://struts.apache.org/2.x/docs/spring-plugin.html

Lo que nos interesa más de este plugin, es la inyección de dependencias que nos ofrece Spring.

Un ejemplo de inyección de dependencias sería el siguiente:

Imaginemos que necesitamos representar un coche en nuestra aplicación, el coche constará de un volante, asientos, ruedas, etc. Al crear un objeto de tipo coche tendríamos que asignarle mediante código, estos componentes. Utilizando el contenedor IOC de Spring, en lugar de hacer todas estas creaciones y asignaciones de dependencias, le pedimos muy amablemente que nos instancie un objeto de tipo coche y el solito se encargará de crear y asignarle las dependencias oportunas.

El objetivo del contenedor IOC (Inversion Of Control) es encargarse de instanciar los objetos de nuestro sistema, denominados beans, y asignarles sus dependencias.

Para obtener las ventajas que nos aporta Spring, a la hora de crear las acciones, deberemos enlazarlas con el bean de Spring.

Creación de una Acción por parte de Struts2

```
<action                                                    name="holaMundo"
class="com.visualGate.actions.HolaMundoAction" >
  <result name="success">/WEB-INF/jsp/holaMundo-success.jsp</result>
  <result name="input">/WEB-INF/jsp/holaMundo-input.jsp</result>
</action>
```

Creación de una Acción por parte de Spring

```
<action name="holaMundo" class="holaMundoAction" >
  <result name="success">/WEB-INF/jsp/holaMundo-success.jsp</result>
  <result name="input">/WEB-INF/jsp/holaMundo-input.jsp</result>
</action>
```

```
<beans>
  <bean                                              id="holaMundoAction"
class="com.visualGate.actions.HolaMundoAction"/>
</beans>
```

El Plugin de Spring es el camino perfecto para crear y manejar los servicios de negocio, ya que facilita la inyección de dependencias a las acciones, sin necesidad de configuración adicional, con este plugin crearemos las acciones normalmente y cuando necesitemos usar los servicios que nos proporciona Spring, solo deberemos usar el interceptor "**setXxxService**" en la Action.

Más sobre setXxxService: Ver ANEXO B.5

Más sobre DAO: Ver ANEXO D – DAO

Para poder usar el Plugin de Spring, Struts2 aporta el objeto **ObjectFactory**, que es el encargado de instanciar todos los objetos creados por el Framework, esto proporciona los medios necesarios para integrar el Framework con los contenedores IOC como Spring.

struts.properties

```
struts.objectFactory = org.apache.struts2.spring.StrutsSpringObjectFactory
```

La integración con Spring requiere de un Listener en web.xml para que pueda acceder a los archivos de configuración necesarios.

web.xml

```
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>classpath*:applicationContext*.xml</param-value>
</context-param>
…
<listener>
  <listener-class>
    org.springframework.web.context.ContextLoaderListener
  </listener-class>
</listener>
```

Finalmente utilizaremos el fichero applicationContext.xml que es el fichero por defecto que tiene Spring para declarar los beans que serán usados por el contenedor IOC para la inyección de dependencias.

applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<beans … >
  <bean id="usuarioService"
        class="com…services.UsuarioServiceImpl"/>
  <bean id="fotoService"
        class="com.visualGate.services.FotoServiceImpl" />
  <bean id="paisService"
        class="com.visualGate.services.PaisServiceImpl" />
</beans>
```

4. Hibernate

Hibernate es una herramienta ORM para la plataforma Java que facilita el mapeo de atributos entre una base de datos relacional (MySql) y el modelo de objetos de una aplicación, esto se puede hacer mediante archivos XML, o por medio de anotaciones en los POJO.

a. Características

- No intrusivo (estilo POJO) no obliga a implementar interfaces determinadas ni heredar de una superclase.
- Muy buena documentación.
- Comunidad activa.
- Permite, transacciones, caché, asociaciones, polimorfismo, herencia, lazy loading, persistencia transitiva, estrategias de fetching.
- Potente lenguaje de consulta HQL
- Fácil testeo.
- Open Source

b. Por qué necesitamos Hibernate?

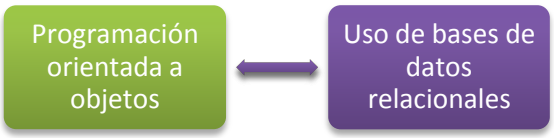


Ilustración 35: ORM - Object-Relacional Mapping

La programación orientada a objetos y las bases de datos relacionales son dos paradigmas diferentes, el modelo relacional trata con relaciones, tuplas y conjuntos, sin embargo la programación orientada a objetos trata con objetos, atributos y relaciones entre estos. Si estamos usando objetos en nuestra aplicación y queremos que sean persistentes normalmente abrirremos una conexión JDBC, crearemos una sentencia SQL y copiaremos todos los valores en la cadena SQL que estemos construyendo, esto se complica cuando el objeto tiene muchas propiedades, asociaciones o contiene otros objetos dentro del mismo objeto a persistir.

Es por eso que surge la necesidad de usar un mapeador ORM que nos ayudará a evitar estos problemas o diferencias entre estos dos paradigmas.

Con un buen ORM, solo tendremos que definir la forma en la que establecemos la correspondencia entre las clases y las tablas una sola vez, como indicar que propiedad se corresponde con que columna y que clase con que tabla.

Hibernate nos proporciona además un lenguaje para el manejo de consultas a la base de datos llamado HQL, similar al SQL que se utiliza para obtener objetos de la base de datos.

La ventaja de usar este lenguaje, es que nos permite usar cualquier base de datos que utilize el lenguaje SQL.

c. Java Persistence API

Desde la versión 3.2.0, Hibernate desarrolla la especificación JPA, con lo que ahora es posible desarrollar una capa de acceso a datos compatible con los estándares de Java en Hibernate, y desplegarla en cualquier servidor de aplicaciones que soporte las especificaciones JEE5.

Esta API busca unificar la manera en que funcionan las utilidades que proveen un mapeo objeto-relacional. ,El objetivo que persigue el diseño de esta API es no perder las ventajas de la orientación a objetos al interactuar con una base de datos, como sí pasaba con EJB2, y permitir usar objetos POJO.

Por otro lado, esta especificación tiene una desventaja, y es que estaremos perdiendo algunas características que ofrece Hibernate.

EntityManagerFactory

Nos proporciona instancias **EntityManager**, todas estas instancias están configuradas para conectarse a la misma base de datos. Esta interface es similar a **SessionFactory** del Hibernate nativo.

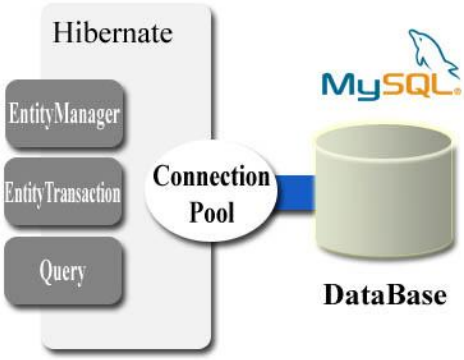


Ilustración 36 : Capa de Persistencia

EntityManager

Es usado para acceder a una base de datos como si fuera una unidad de trabajo, sería lo equiparable al objeto sesión del Hibernate nativo. Se usa para crear, eliminar y buscar instancias de entidades persistentes mediante su clave primaria o mediante una query para acceder a todas sus propiedades.

EntityTransaction

Nos aporta todas las características de una transacción:

Query

Permite realizar peticiones a la base de datos y controla cómo se ejecuta dicha petición (query). Las peticiones se escriben en HQL o en el dialecto SQL nativo de la base de datos que estamos utilizando. Una instancia Query se utiliza para enlazar los parámetros de la petición, limitar el número de resultados devueltos por la petición y para ejecutar dicha petición.

d. Persistiendo los objetos del modelo de dominio

A la hora de hacer persistir un objeto del modelo de dominio, Hibernate ha de conocer como relacionar el objeto con la base de datos, para eso dispone de una serie de anotaciones que utilizaré para definir el objeto a persistir, como indicar que propiedad se corresponde con que columna, que clase con que tabla, la primary key o si tiene alguna relacion de una a muchas como es el caso del ejemplo que sigue.

Este es el caso del objeto Usuario, donde se muestran las anotaciones necesarias para hacer persitir el objeto correctamente:

Usuario

```
@Entity @Table(name="USUARIO", schema="VISUALGATE")
public class Usuario implements Serializable{

  private String nombre;
  private String apellido;
  private String email;
  private String password;
  private byte[] fotoUsuario;
  private List<Foto> fotos = new ArrayList<Foto>();

  @Column(name="NOMBRE")
  public String getNombre(){return nombre;}
  public void setNombre(String nombre){this.nombre = nombre;}

  @Column(name="APELLIDO")
  public String getApellido(){return apellido;}
  public void setApellido(String apellido){this.apellido = apellido;}

  @Id @Column(name="EMAIL")
  public String getEmail(){return email;}
  public void setEmail(String email){this.email = email;}

  @Column(name="PASSWORD")
  public String getPassword(){return password;}
  public void setPassword(String password){this.password = password;}

  @Column(name="FOTO", columnDefinition="LONGBLOB",nullable=true)
  public byte[] getFotoUsuario(){return fotoUsuario;}
  public void setFotoUsuario(byte[] fotoUsuario){
     this.fotoUsuario = fotoUsuario;
  }

  @OneToMany(mappedBy="usuario",cascade=CascadeType.ALL)
  public List<Foto> getFotos() {
    return fotos;
  }
  public void setFotos(List<Foto> fotos) {
    this.fotos = fotos;
  }
  public void addFoto(Foto f){
    f.setUsuario(this);
    fotos.add(f);
  }
}
```

5. Rome

Rome es un conjunto de librerías Open Source que nos aporta las herramientas necesarias para la generación y publicación de Feeds RSS y Atom.

Rome soporta todas las versiones y formatos RSS, además de Atom, aunque en VisualGate solo se utilizará los RSS 2.0

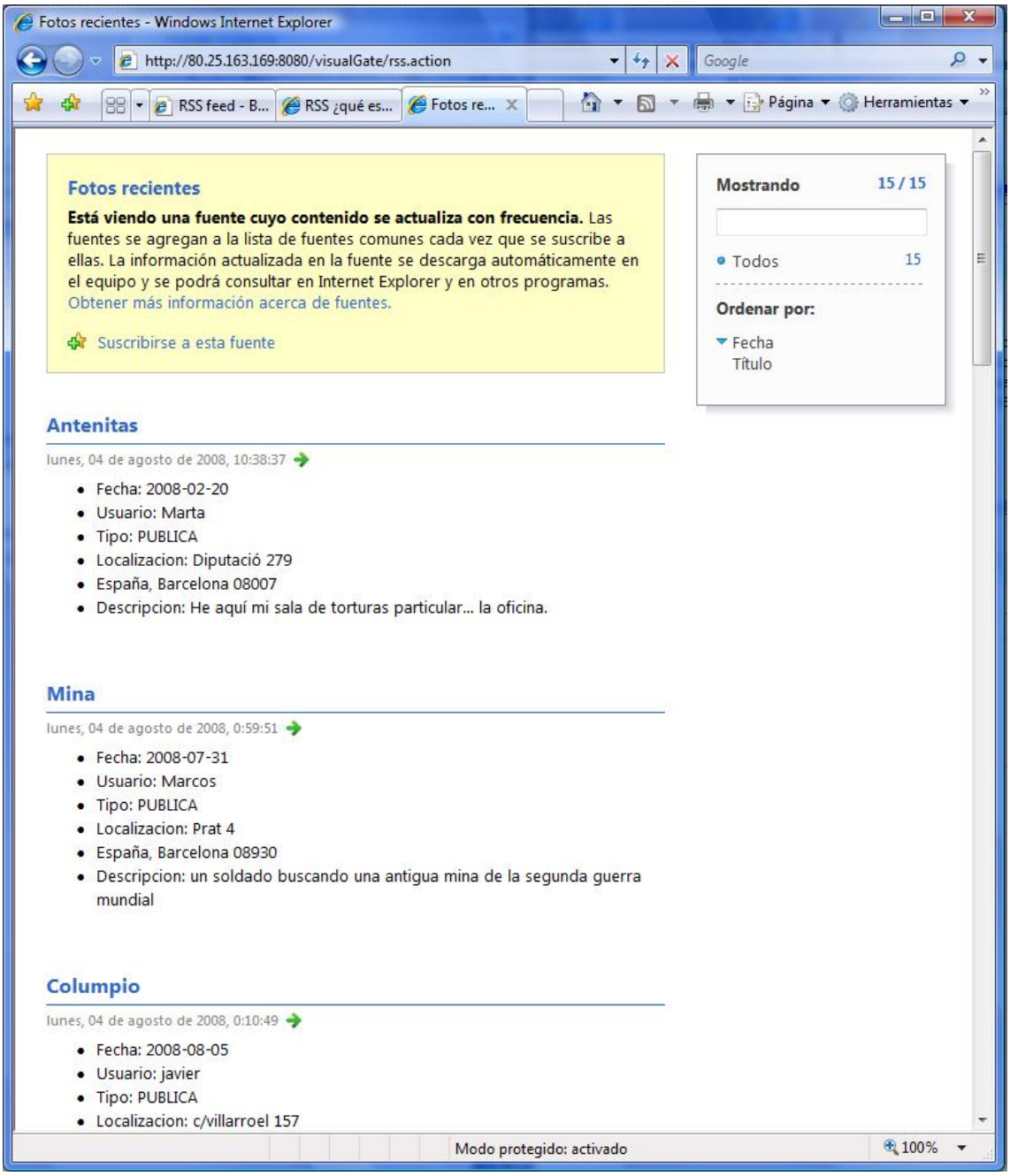


Ilustración 37 : RSS de Fotos, generado usando Rome

ANEXO B. Interceptores

En la siguiente tabla podemos encontrar los interceptores disponibles en Struts2

Interceptor	Nombre	Descripción
Alias Interceptor	alias	Permiten a los parámetros tener nombres diferentes como aliases.
Chaining Interceptor	chaining	Permite que las propiedades de la action ejecutada anteriormente estén disponibles en la action actual. Se usa conjutamente con el Result Type Chain.
Checkbox Interceptor	checkbox	Añade un valor de False a los check boxes que no están chequeados.
Create Session Interceptor	createSession	Se crea automáticamente una sesión http si no ha sido creada aun.
Debugging Interceptor	debugging	Provee algunas opciones de debugging al desarrollador.
Execute and Wait Interceptor	execAndWait	Envía al usuario una página de espera mientras la action se está ejecutando.
Exception Interceptor	exception	Mapea las excepciones que son lanzadas de una action, así las podemos recoger y redirigir la salida.
File Upload Interceptor	fileUpload	Facilita la subida de ficheros.
Internationalization Interceptor	i18n	Proporciona internacionalización.
Logging Interceptor	logger	Provee un simple logging, donde se muestra las acciones que están siendo ejecutadas.
Message Store Interceptor	store	Almacena y recupera los mensajes, los campos con errores, errores en la Action en la sesión para las Actions que implementen a la interface ValidationAware.
Model Driven Interceptor	modelDriven	Ubica el objeto del modelo dentro de la Value Stack para las acciones que implementen a modelDriven.
Scoped Model Driven Interceptor	scopedModelDriven	Lo mismo que modelDriven, pero en este caso almacena y recupera la información según el alcance utilizado (Scope)
Parameters Interceptor	params	Pone los parámetros de la petición en la Action.
Parameter Filter	n/a	Provee control sobre cualquier parámetros que la acción tenga acceso.

Interceptor		
Prepare Interceptor	prepare	Invoca al método prepare() de la action que implemente a la interface Prepare.
Scope Interceptor	scope	Almacena y recupera el estado de la action en la sesión o la aplicación.
Static Parameters Interceptor	staticParams	Pone los parámetros estáticos ya definidos dentro de la action.
Roles Interceptor	roles	Permite que la action sea solo ejecutada si el usuario es uno de los roles configurados.
Timer Interceptor	timer	Provee información en el formularios de cuánto tiempo tardará en ejecutarse la Action.
Token Interceptor	token	Nos asegura que no se produzca un doble click en un submit de un formulario.
Token Session Interceptor	tokenSession	Los mismo que token pero para tokens invalidos.
Validation Interceptor	validation	Provee soporte de validación a las Actions.
Workflow Interceptor	workflow	Redirige a la vista INPUT sin ejecutar la action (execute()) cuando la validación falla.

Cada interceptor proporciona una funcionalidad para la Action, pero lo lógico será que necesitemos más de uno para cada action.

Struts2 nos permite disponer de pilas de interceptores. En la siguiente tabla se muestran las pilas pre configuradas y listas para usar por las Actions. El orden en que se llama a cada interceptor es muy importante, sino se respetara el orden o se creara una pila de interceptores personalizados con un orden no adecuado se podrían producir errores.

Nombre de la pila de interceptores configurados	Interceptors incluidos	Descripción
basicStack	Exception, servletConfig, prepare, checkbox, params, conversionError	Los interceptors que son esperados en una situación mínima
validationWorkflowStack	basicStack, validation, workflow	Añade validación y workFlow a las características básicas.
fileUploadStack	fileUpload, basicStack	Añade subida de ficheros a las características básicas.
modelDrivenStack	modelDriven, basicStack	Añade funcionalidad del modelo a las características básicas.

chainStack	Chain, basicStack	Añade encadenamiento a las características básicas.
i18nStack	I18n, basicStack	Añade internacionalización a las características básicas.
paramPrepareParamsStack	Exception, alias, params, servletConfig, prepare, i18n, chain, modelDriven, fileUpload, checkbox, staticParams, params, conversionError, validation, workflow	Provee una pila complete de interceptors incluyendo preparación para la action. El interceptor params es usado 2 veces, una para proveer los parámetros antes de prepare() y la siguiente para aplicar los parámetros a los objetos que deberían haber sido cargados durante el prepare()
defaultStack	Exception, alias, servletConfig, prepare, i18n, chain, debugging, profiling, scopedModelDriven, modelDriven, fileUpload, checkbox, staticParams, params, conversionError, validation, workflow	Provee una pila complete, incluyendo debugging y información.
executeAndWaitStack	execAndWait, defaultStack, execAndWait	Util para la subida de ficheros, donde el usuario tiene que esperar a que el fichero se suba correctamente.

1. Interceptor params

params

El interceptor **params** establece los parámetros de la petición en la Action.

```
public class Usuario implements Serializable{

    private String nombre;
    private String apellido;
    private String email;
    private String password;
    private byte[] fotoUsuario;

    public String getNombre(){return nombre;}
    public void setNombre(String nombre){this.nombre
    public String getApellido(){return apellido;}
    public void setApellido(String apellido){this.
    public String getEmail(){return email;}
    public void setEmail(String email){this.email = email;}
    public String getPassword(){return password;}
    public void setPassword(String password){this.password = password;}
    public byte[] getFotoUsuario(){return fotoUsuario;}
    public void setFotoUsuario(byte[] fotoUsuario){this.fotoUsuario = fotoUsuario;}
}
```

Además tiene la funcionalidad de **conversión de datos**. En el caso de que se preguntara la edad, el mismo interceptor se encargaría de la conversión.

<s:textfield label="Cual es tu edad?" name="edad" />

↓

params interceptor (conversión)

↓

public void setEdad(int edad) { .. }

2. Interceptor Model-Driven

modelDriven

Sabemos que los parámetros introducidos mediante el interceptor params desde un formulario, se insertan dentro de los setters de la Action, pero por otro lado tenemos el objeto de dominio con sus getters y setters, así que proveer en la acción las mismas propiedades violaría el principio DRY (Don´t Repeat Yourself) así que la solución sería proporcionar a la Action una manera de acceder al objeto de dominio o mejor dicho a las propiedades del objeto de dominio, como si el objeto de dominio estuviera dentro de la misma Action.

El interceptor Model-Driven proporciona acceso al objeto de dominio (propiedades) directamente desde el .jsp o mejor dicho desde la Vista. (setters & getters)

Para poder usar el interceptor Model-Driven, necesitamos cumplir dos requisitos:

La Action ha de implementar a la interface ModelDriven

- Esto nos obliga a implementar un método que nos permite acceder a la instancia del objeto de dominio.

```
public class BaseUsuarioAction extends BaseAction
                                implements ModelDriven<Usuario> {

        protected Usuario usuario;
        …

        /* INTERCEPTOR MODELDRIVEN */
        public Usuario getModel() {
                return usuario;
        }

        …
}
```

La Action ha de disponer del interceptor ModelDriven

- Tenemos que asegurarnos que la Action esté dentro de un package de configuración en struts.xml o mediante annotations que incluya el interceptor ModelDriven

El interceptor se encargará de recoger el modelo de dominio y colocarlos en la ValueStack, permitiendo a los Tags de struts2 acceder directamente al modelo de dominio.

Utilizando el interceptor ModelDriven	Utilizando la implementación pero no el interceptor ModelDriven
`<s:textfield …name="nombre" />`	`<s:textfield …name="model.nombre" />`

3. Interceptor Prepare

prepare

En ocasiones necesitamos configurar la Action o prepararla de algún modo antes de que se ejecute su método principal **execute().** Imaginemos que estamos usando una Action que contiene el interceptor **modelDriven** que hace disponible el objeto de dominio al .JSP incluyéndolo dentro de la **ValueStack**. Sabemos que el orden de los interceptores es importante, y también sabemos que el interceptor **modelDriven** se ejecuta antes que el método del Action **execute().** Así que estaremos devolviendo un **null** cuando el interceptor **modelDriven** coloque el modelo de dominio dentro de la **ValueStack** porque aun no lo hemos instanciado.

La solución está en el interceptor Prepare, que es el encargado de llamar a un método de la Action llamado prepare() para configurar la Action antes del método execute().

Para poder usar el interceptor Prepare, necesitamos cumplir dos requisitos:

La Action ha de implementar a la interface Preparable

- Esto nos obliga a implementar un método que nos permite configurar la Action antes de su método principal **execute()** sea llamado.

```
public class BaseUsuarioAction extends BaseAction
                              implements ModelDriven<Usuario>,
                                         Preparable {

      protected Usuario usuario;
      …

      /* INTERCEPTOR PREPARABLE */
      public void prepare() throws Exception {
            usuario = new Usuario();
      }

      /* INTERCEPTOR MODELDRIVEN */
      public Usuario getModel() {
            return usuario;
      }

      public String execute() throws Exception{
            return SUCCESS;
      }

      …
}
```

La Action ha de disponer del interceptor Preparable

- Tenemos que asegurarnos que la Action esté dentro de un package de configuración en struts.xml o mediante annotations que incluya el interceptor Preparable

4. Interceptor ServletRequestAware

servletRequestAware

En cualquier aplicación web que necesita conservar el estado entre peticiones, necesita recurrir a las sesiones. Struts2 nos provee un interceptor que proporciona acceso a los objetos de petición mediante el HttpServletRequest.

Para poder usar el interceptor ServletRequestAware, necesitamos cumplir dos requisitos:

La Action ha de implementar a la interface ServletRequestAware

- Esto nos obliga a implementar un método que nos permite acceder al objeto HttpServletRequest y así poder acceder a los objetos de la petición.

```
public class BaseUsuarioAction extends BaseAction
                                implements ModelDriven<Usuario>,
                                           Preparable,
                                           ServletRequestAware {

  protected Usuario usuario;
  protected HttpServletRequest httpServletRequest;
  …

  /* INTERCEPTOR PREPARABLE */
  public void prepare() throws Exception {
      usuario = new Usuario();
  }

  /* INTERCEPTOR MODELDRIVEN */
  public Usuario getModel() {
      return usuario;
  }
  /* INTERCEPTOR SERVLETREQUESTAWARE */
  public void setServletRequest(HttpServletRequest
                                          httpServletRequest){
      this.httpServletRequest=httpServletRequest;
  }

  public String execute() throws Exception{
      return SUCCESS;
  }

   …
}
```

La Action ha de disponer del interceptor ServletRequestAware

- Tenemos que asegurarnos que la Action esté dentro de un package de configuración en struts.xml o mediante annotations que incluya el interceptor ServletRequestAware

5. Interceptor setXxxService

setUsuarioService

Llega el momento en el cual, necesitamos recuperar un objeto de la base de datos, como por ejemplo el **objeto Usuario**. El encargado de recuperarlo es **Spring**, ya que nos proporciona la **inyección de dependencias**. Nosotros solo le pedimos una instancia y él nos devuelve el objeto Usuario con todas sus dependencias.

Para las funcionalidades de Usuario, he definido una interfaz llamada **UsuarioService**, que nos proporciona la funcionalidad **CRUD** (Copy, Read, Update & Delete) mediante el patrón DAO, esto nos permitirá hacer persistir un objeto Usuario en la base de datos, leerlo (recuperarlo), actualizarlo o eliminarlo, sin tener que conocer como está implementado.

Más sobre Spring:
Ver ANEXO A.3

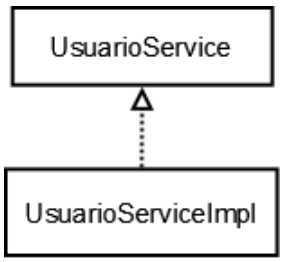
Más sobre DAO:
Ver ANEXO D

La persistencia de objetos en la base de datos sigue el patrón de diseño DAO.

UsuarioService.java (DAO)

```
public interface UsuarioService {

  public Usuario buscaUsuarioPorEmail(String email);
  public void persist(Usuario usuario, boolean actualiza);
  public void eliminar(String email);

}
```

Junto con la interfaz, he creado una clase **UserServiceImpl**, que implementa los métodos de **UsuarioService**. Esta clase usa el **JPA (Java Persistence API)** para persistir la instancia del objeto de dominio a la base de datos.



Más sobre JPA:
Ver ANEXO A.4.3

Más sobre Persistencia:
Ver ANEXO A.4

La magia es la forma en la que el objeto definido en la configuración de Spring (applicationContext.xml) se inyecta en la Action.

applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans … >
  <bean id="usuarioService" class="com…services.UsuarioServiceImpl"/>
  <bean id="fotoService"
        class="com.visualGate.services.FotoServiceImpl" />
  <bean id="paisService"
        class="com.visualGate.services.PaisServiceImpl" />
</beans>
```

Y para eso necesitamos un setter del objeto en la Action, este setter deberá llamarse como el nombre del atributo id en la configuración de Spring (applicationContext.xml).

```
public class BaseUsuarioAction extends BaseAction … {

	…
	protected UsuarioService usuarioService;
	…

	/* INYECCION DEL OBJECTO A LA ACCION */
	public void setUsuarioService(UsuarioService usuarioService) {
		this.usuarioService = usuarioService;
	}


	public String execute() throws Exception{
		return SUCCESS;
	}

	…
}
```

Una vez inyectado, podremos acceder a los servicios de negocio desde la acción.

6. Interceptor Scope

El método más simple para crear un **WorkFlow** en una aplicación Struts2, es almacenar la información en un sistema donde esté disponible en cualquier momento del **WorkFlow** y esta se elimine en el momento de finalizar. Para optimizarlo aun más, sería conveniente no usar la base de datos para almacenar esta información, solo se usaría cuando el **WorkFlow** finalizara.

El **interceptor Scope** de Struts2, nos almacena la información en las variables de **sesión**, aunque también podemos elegir diferentes "**scopes**" como el de **aplicación**.

El funcionamiento de **Scope** es sencillo, cuando se inicia el WorkFlow se inician las variables y cuando finaliza se eliminan. Para ello se han de declarar dos acciones que realicen estas tareas:

Y lo más importante es que las acciones que participen en el **WorkFlow** incluyan el **interceptor Scope**. Con que ninguna pila de interceptores incluye por defecto este interceptor, tendremos que crear nuestra propia pila de interceptores dentro de un package personalizado que usarán las acciones.

struts.xml

```
<package name="enterFoto" namespace="/foto" extends="base-package" >
  <interceptors>
    <interceptor-stack name="fotoStack">
      <interceptor-ref name="security" />
      <interceptor-ref name="scope">
        <param name="session">Model</param>
        <param name="key">partialFoto</param>
       </interceptor-ref>
       <interceptor-ref name="paramsPrepareParamsStack"/>
     </interceptor-stack>
  </interceptors>
  <default-interceptor-ref name="fotoStack" />
  <action name="startWorkFlowFoto"
          class="com.visualGate.actions.BaseAction">
    <interceptor-ref name="fotoStack">
      <param name="scope.type">start</param>
    </interceptor-ref>
      <result>/WEB-INF/jsp/foto/entrarFotoDetalles-input.jsp</result>
  </action>
  <action name="endWorkFlowFoto"
          class="com.visualGate.actions.BaseAction">
    <interceptor-ref name="fotoStack">
      <param name="scope.type">end</param>
    </interceptor-ref>
      <result>/WEB-INF/jsp/foto/saveFoto-success.jsp</result>
  </action>
</package>
```

Para tener una visión más clara del funcionamiento, estaría bien echar un vistazo al **diagrama de flujo del UC6**, donde se usa este interceptor.

Más sobre el diagrama de Flujo: Ver ANEXO E

7. Interceptor Security

A diferencia del resto de interceptores mostrados anteriormente, este es un interceptor personalizado, únicamente creado para aportar autorización de usuarios a VisualGate.

El acceso a recursos lo he basado en la idea del contenedor de Servlets como es el Tomcat, donde hay que especificar que directorios de la aplicación web requerirán de un acceso especial, la diferencia es que este interceptor leerá estos directorios en el fichero de struts.xml

Struts.xml

```
…
<interceptor name="security" class="com…SecurityInterceptor" >
    <param name="requiresAuthentication">/foto,/admin</param>
</interceptor
…
```

También es posible que según la circunstancia haya un recurso que no esté contenido en esos directorios, pero que necesitemos marcar como "protegido", así que las acciones que necesiten ser tratadas de manera diferente las marcaré con una anotación especial.

Action

```
@RequiresAuthentication
public class LogoffAction extends BaseAction …{
…
```

Para permitir el acceso a un recurso solicitado por un usuario, se valorarán estos tres parámetros:

Usuario
- TRUE = Si el usuario NO se encuentra en sesión, eso quiere decir que el usuario no ha sido autenticado.

Anotacion
- TRUE = La acción requiere autorización para poderla ejecutar.

Package
- TRUE = El package donde se accede está protegido y requiere autorización.

User	Anotación	Package	Acceso
FALSE	FALSE	FALSE	PERMITIDO
FALSE	FALSE	TRUE	PERMITIDO
FALSE	TRUE	FALSE	PERMITIDO
FALSE	TRUE	TRUE	PERMITIDO
TRUE	FALSE	FALSE	PERMITIDO
TRUE	FALSE	TRUE	NO PERMITIDO
TRUE	TRUE	FALSE	NO PERMITIDO
TRUE	TRUE	TRUE	NO PERMITIDO

El interceptor **security** estará disponible en la pila de interceptores **securedStack** contenida en el package **home-package,** así que las acciones que necesiten este interceptor deberán utilizar **home-package** o utilizar un package que extienda de **home-package**.

Más sobre Packages VisualGate:
Ver ANEXO C

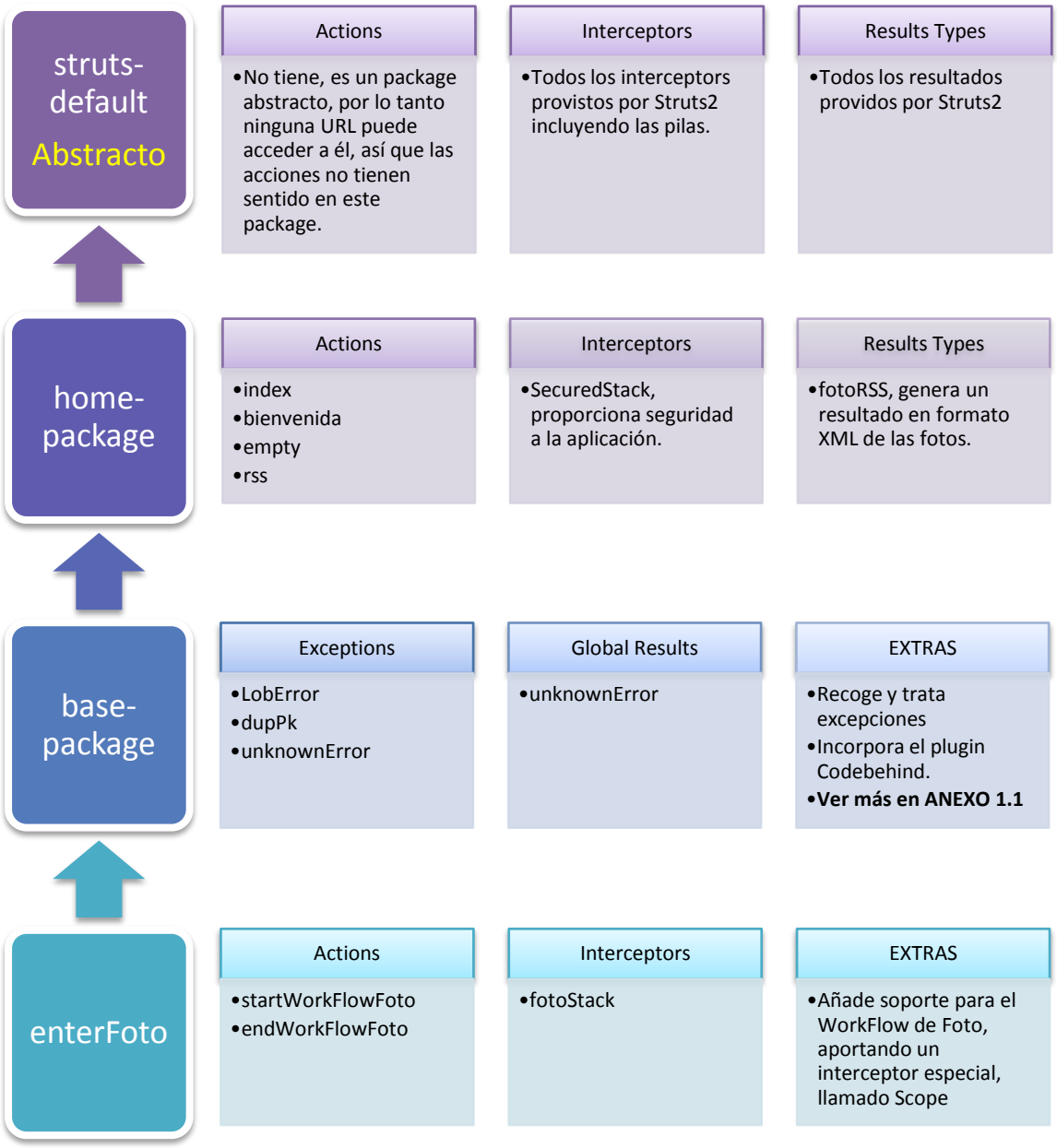
ANEXO C. Packages VisualGate

Cuando estuvimos hablando de la configuración de los elementos del Framework vimos el fichero de configuración struts.xml, el cual permitía poder configurar acciones, resultados y interceptores mediante diferentes Packages por medio del XML y así mejorar la modularidad de la aplicación.

Más sobre Conf. Elementos Framework: Ver 2.3.4

Más sobre Packages: Ver 2.3.4.2

Ahora llega el momento de crear los Packages que tendrá la aplicación web VisualGate, que características tendrán a quien extenderán y que aportarán a las acciones que lo implementen.



ANEXO D. Patrón de diseño DAO – Data Access Object

Cuando realizamos una aplicación como es el caso de VisualGate, surge la necesidad de obtener datos de varias fuentes distintas, como pueden ser bases de datos.

El objetivo de este patrón es encapsular la fuente de datos, de manera que no necesitemos saber cómo se acceden a ellos, y que podamos consumir los datos sin preocuparnos de la fuente que los suministra. De esta forma conseguimos que nuestra lógica de negocio no sepa nada de Hibernate, y siempre que quiera acceder a los datos lo hará usando la interface DAO, con esto conseguimos reducir el acoplamiento y así podemos intercambiar la implementación fácilmente si algún día nos cansamos de Hibernate/JPA.

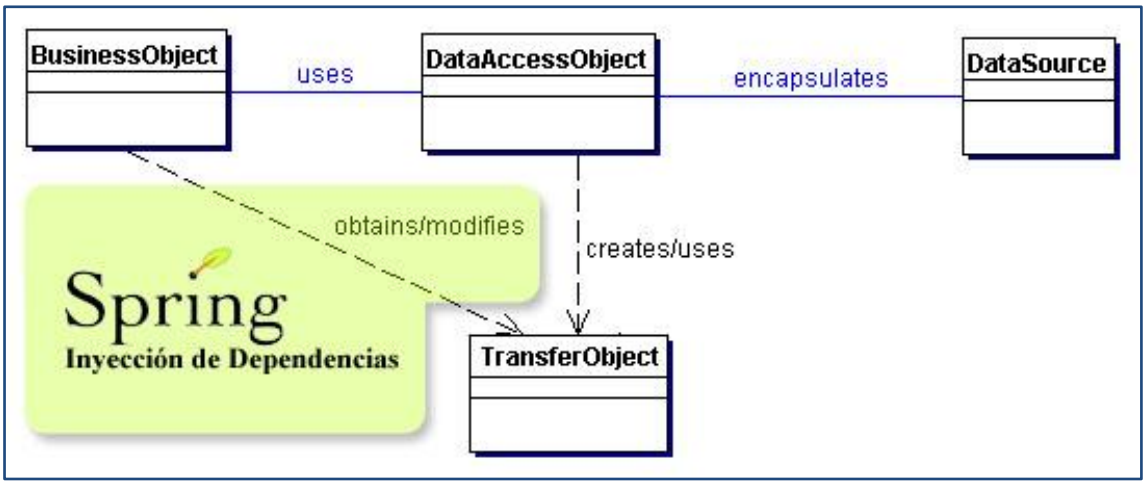


Ilustración 38 : Patrón DAO + Spring

Spring es el camino perfecto para crear y manejar los servicios de negocio, ya que facilita la inyección de dependencias a las acciones (BusinessObject).

BusinessObject

- Es la clase que va a **obtener** y **almacenar** los datos que obtendremos con el patrón DAO. En nuestra aplicación, los businessObjects serían las Action. A continuación un businessObject que obtiene y almacena datos utilizando el patrón DAO.

Action

```
…
    public String execute() throws Exception {

    //usuarioService es la interface DAO que nos proporciona
    //servicios para obtener o almacenar objetos en la base de datos.

        usuarioService.buscaUsuarioPorEmail(email);
        usuarioService.persist(usuario,actualiza);

        return SUCCESS;
    }
…
```

DataAccesObject o DAO

- Un DAO es un objeto que provee una interface para algún tipo de base de datos o mecanismo de persistencia como puede ser Hibernate, permitiendo operaciones especificas sin exponer los detalles de cómo se accede a la base de datos.
- Una buena práctica, es que estas interfaces incorporen la funcionalidad CRUD.

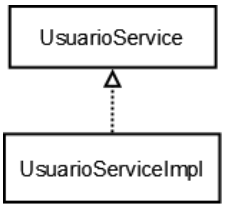
Interface DAO – UsuarioService.java

```
public interface UsuarioService {

 public Usuario buscaUsuarioPorEmail(String email);
 public void persist(Usuario usuario, boolean actualiza);
 public void eliminar(String email);
}
```

Esta interface simplemente define los principales métodos de servicios que nuestra aplicación usará. La implementación de esta interface podríamos hacerlo con Hibernate Nativo, Hibernate/JPA, XML…

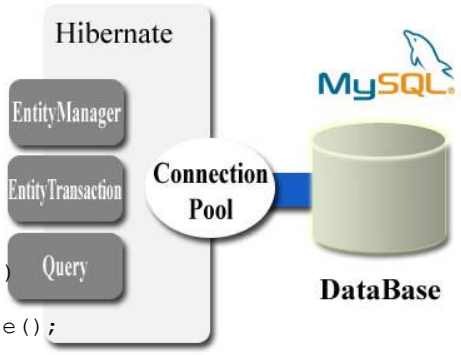
DataSource

- Contiene la implementación específica para cada fuente de datos. En este caso usamos Hibernate - JPA.



DAO – UsuarioServiceImpl.java

```
public class UsuarioServiceImpl implements UsuarioService {

    private EntityManagerFactory emf;
    public UsuarioServiceImpl() {
        emf = Persistence.createEntityManagerFactory("visualGate");
    }
    public Usuario buscaUsuarioPorEmail(String email) {
        EntityManager entityMgr = emf.createEntityManager();
        return entityMgr.find(Usuario.class,email);
    }
    /*Persiste el objeto usuario insertandolo o actualizandolo*/
    public void persist(Usuario usuario,boolean actualiza) {
        EntityManager entityMgr = emf.createEntityManager();
        EntityTransaction tx = null;
        try {
            tx = entityMgr.getTransaction();
            tx.begin();
            if( actualiza == false ) {
                entityMgr.persist(usuario);
            } else {
                entityMgr.merge(usuario);
            }
            tx.commit();
        } catch (Exception e) {
            if ( tx != null && tx.isActive() )
                tx.rollback();
            throw (RuntimeException)e.getCause();
        }
    }
}
```



TransferObject o DTO (Data Transfer Object)

- Son objetos para transferir datos, su función es pasar los datos de una capa a otra.
- Proporcionan una representación de otro objeto, o pueden ofrecer una representación de elementos de datos de varios objetos, que son por lo general, objetos relacionados (dependencias)
- Pueden ser Mutable o no mutables
- Pueden o no ofrecer todas las propiedades del objeto que representan.

Usuario.java

```
public class Usuario implements Serializable{

  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;}
  …
}
```

Ventajas

- Los DAO son un patrón de diseño J2EE y es considerada como buena práctica.
- Los objetos de negocio no conocen como se va a manipular la información, ellos solo tienen acceso a unos servicios para almacenar o obtener información de la base de datos.
- Separación entre la capa de persistencia y la aplicación.
- Los cambios realizados en la capa de persistencia no afectan a los clientes que usan DAO.

Desventajas

- Al agregar un DAO, la complejidad de usar otra capa aumenta la cantidad de código ejecutado durante el tiempo de ejecución.

ANEXO E. Diagramas de Flujo

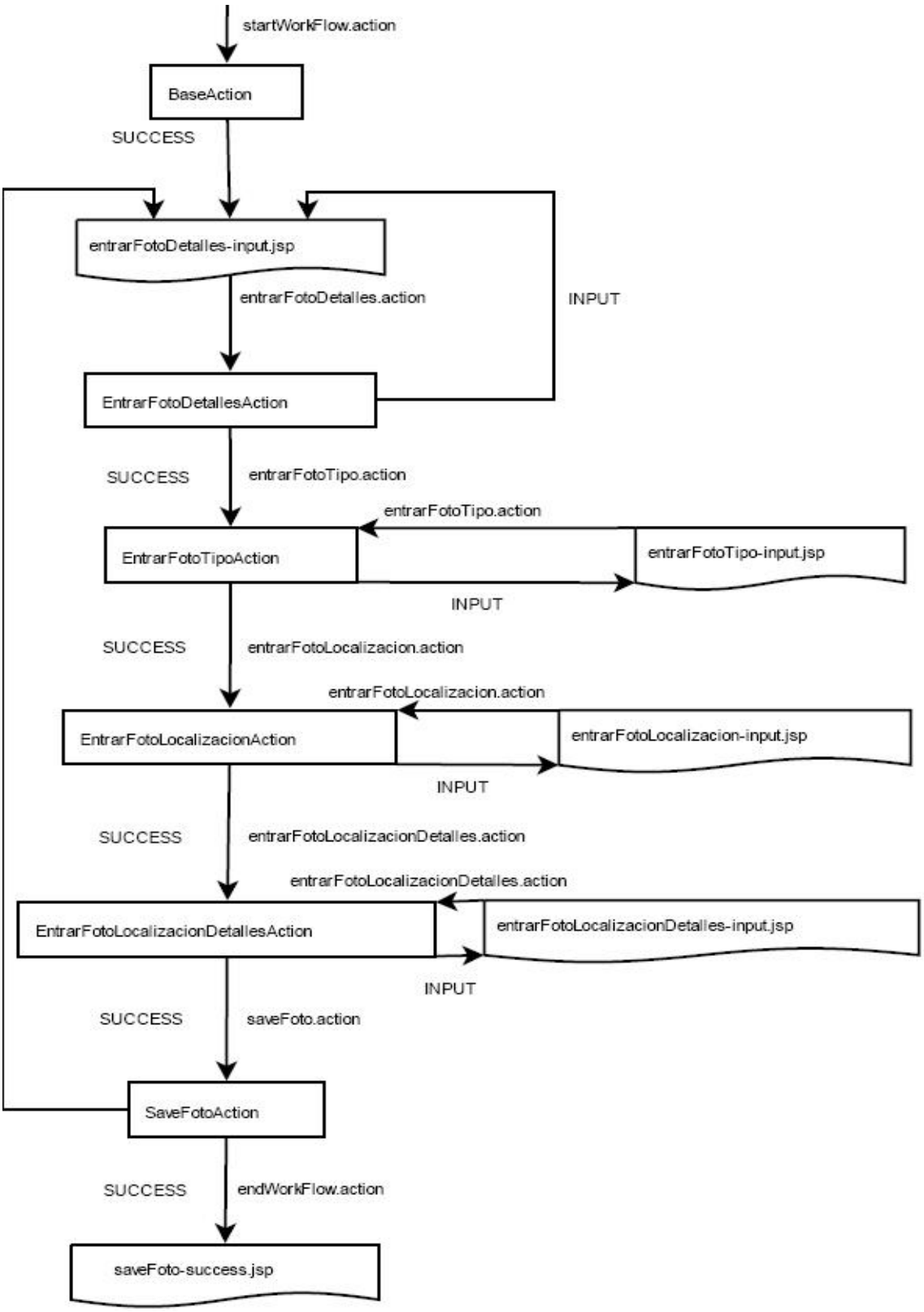


Ilustración 39 : Diagrama de Flujo UC6 - Registrar o Añadir Foto

ANEXO F. Comparativas de Java Web Frameworks

Java Server Faces

Ventajas

- Estándar Java Enterprise Edition.
- Es muy popular y existe mucha demanda de trabajo.
- Fácil y rápido de desarrollar inicialmente.
- Muchas librerías para añadir componentes.
- Validaciones fáciles de configurar.
- Soporta Internacionalización.
- Puede usar Tiles & SiteMesh para la decoración de páginas.

Desventajas

- No trabaja bien con REST.
- Se requiere de varias fuentes para la implementación.
- No soporta AJAX, se tiene que recurrir a librerías como Ajax4JSF.
- Usa un único fichero para la internacionalización.
- No se recomienda el uso de SiteMesh por incompatibilidades.

Spring MVC

Ventajas

- Integración con diferentes opciones para la vista como JSP/JSTL, Tiles, Velocity, FreeMarker, Excel, PDF
- Facilidad en la testeabilidad.
- Soporta Internacionalización.
- Puede usar Tiles & SiteMesh para la decoración de páginas.
- En cuanto a salida laboral, es muy conocido pero no como Framework MVC sinó por otras características.

Desventajas

- Intensa configuración y montones de XML
- Demasiado Flexible.
- No está pensado para soportar AJAX, no tiene librerías, se ha de usar DWR y extras para poder implementarlo.
- Usa un único fichero para la internacionalización.

Struts2

Ventajas

- Arquitectura simple y fácil de extender. POJO´s
- Framework especializado en controlador.
- Librerías de Tags fáciles de personalizar.
- AJAX integrado mediante Dojo Tookit.
- Plugins para integrar GWT y resultados JSON.
- OGNL ayuda en la conversión y validación en la parte del cliente.
- Facilidad en la testeabilidad.
- Soporta la internacionalización y apuesta por la separación de ficheros por cada página y acción en la internacionalización.
- Puede usar Tiles & SiteMesh para la decoración de páginas.

Desventajas

- Documentación mal organizada.
- Struts 1.x abarca la mayoría de resultados en las búsquedas de internet.
- En cuanto a salida laboral, Struts2 está ganando terreno poco a poco.

Tapestry

Ventajas

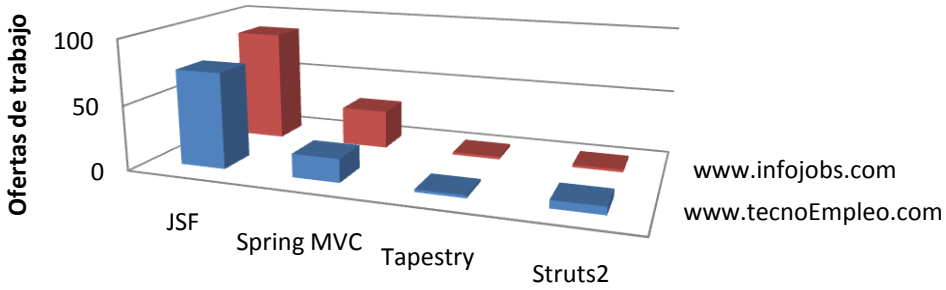
- Muy productivo una vez se aprende como funciona.
- Muchas actualizaciones y innovaciones entre versiones.
- Integra AJAX mediante Dojo Toolkit
- Buen sistema de validación.
- Soporta la internacionalización y apuesta por la separación de ficheros por cada página y acción en la internacionalización.
- Puede usar Tiles & SiteMesh para la decoración de páginas.
- Está ganando popularidad en el mundo laboral.

Desventajas

- Mala documentación.
- Curva de aprendizaje lenta.
- No se recomienda el uso de SiteMesh por incompatibilidades.
- El tiempo entre actualizaciones es muy largo.

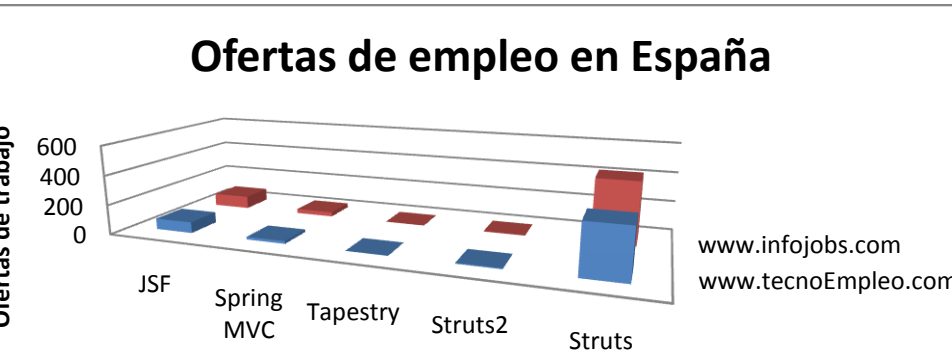
1. Demanda laboral

Ahora veremos una serie de gráficos donde se muestran las ofertas de trabajo disponibles en este mes de septiembre.



	JSF	Spring MVC	Tapestry	Struts2
www.tecnoEmpleo.com	74	18	2	6
www.infojobs.com	87	30	2	2

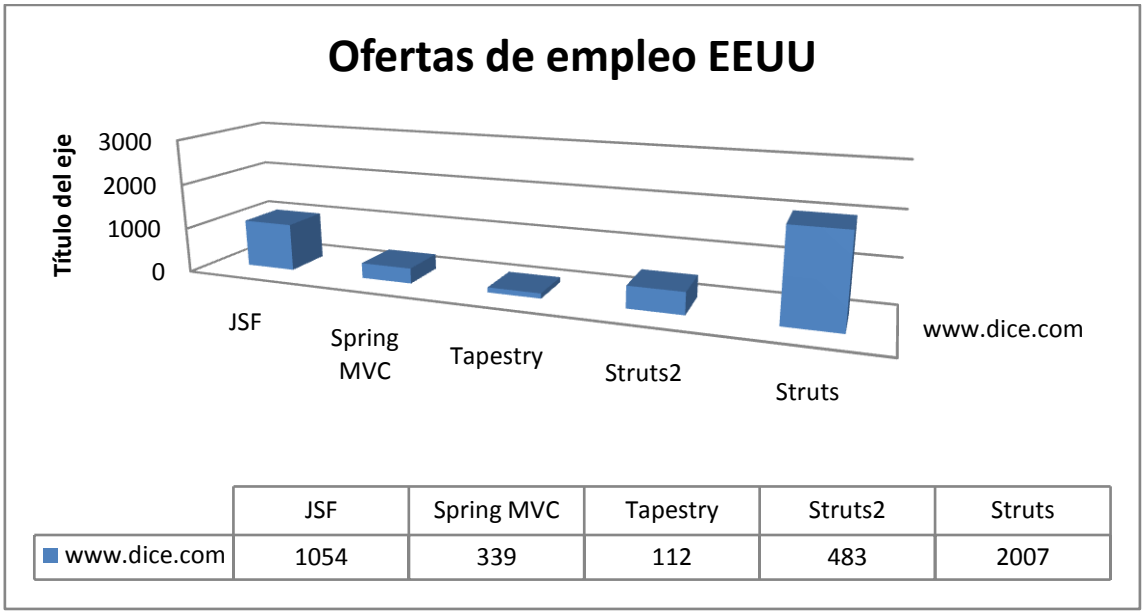
Parece ser que Struts2 aun no ha calado mucho en España. Veamos que sucede si hacemos la comparativa añadiendo el Framework Struts.



	JSF	Spring MVC	Tapestry	Struts2	Struts
www.tecnoEmpleo.com	74	18	2	6	333
www.infojobs.com	87	30	2	2	425

Bueno, parece ser que Struts aun está siendo usado mayoritariamente por las empresas y es por eso que Struts2 aun no ha despegado, espero que con el tiempo las empresas que estén usando Struts pasen a usar Struts2 por afinidad a la misma tecnología, aunque poco se parezcan.

Veamos ahora, como se desarrolla Struts2 en EEUU.



	JSF	Spring MVC	Tapestry	Struts2	Struts
www.dice.com	1054	339	112	483	2007

Si tomamos como referencia a EEUU, pronto Struts2 se impondrá en España a Tapestry y Spring MVC, haciendo frente a su competidor directo JSF.

ANEXO G. JUNIT – Pruebas Unitarias

JUnit permite realizar ejecución de clases, en nuestro caso Acciones, de manera controlada para poder controlar y evaluar el funcionamiento correcto de cada uno de los métodos de la Acción (clase). De esta manera, podemos llegar a evaluar todos los casos posibles y provocar intencionadamente errores para poder llegar a controlarlos correctamente.

Para la aplicación web VisualGate, se han realizado varios test unitarios para los principales casos de uso de la aplicación, como es el caso de la autorización controlada por el interceptor personalizado que creamos específicamente para la aplicación. Además de esto también se han realizado varios test de persistencia de datos, usando la funcionalidad CRUD.

Test Unitario que comprueba la funcionalidad CRUD del usuario.

```
public class UsuarioCRUDTestCase extends MockObjectTestCase {

  private Usuario usuario;

  public UsuarioCRUDTestCase(){
    super();
  }
  protected void setUp() throws Exception {
    super.setUp();
  }
  protected void tearDown() throws Exception {
    super.tearDown();
  }
  public void crearUsuario(){
    …
  }
  public void testCrearUsuario() throws Exception {

    Mock service = new Mock(UsuarioService.class);

    service.expects(once()).method("persist").with(isA(Usuario.class),
    eq(false));

    UsuarioCUAction action = new UsuarioCUAction();
    crearUsuario();
    action.setUsuarioService((UsuarioService)service.proxy());
    action.prepare();
    assertEquals(Action.SUCCESS,action.execute());
    service.verify();
    }
  public void testActualizarUsuario() throws Exception {
    …
    service.verify();
  }
  public void testEliminarUsuario() throws Exception {
    …
    service.verify();
  }
}
```

ANEXO H. Etapas del ciclo de vida de un Sistema Software

- **Ingeniería del sistema (requerimientos y especificación)**

Esta etapa tiene como objetivo la consecución de un primer documento en que queden reflejados los requerimientos y funcionalidades que ofrecerá al usuario del sistema a desarrollar (qué, y no cómo, se va a desarrollar). Es de gran importancia puesto que unos requerimientos bien definidos suponen una base sólida sobre la que construir la solución. Se formalizarán los requerimientos, se definirá con precisión el sistema requerido (empleo de los casos de uso, use cases, para llevar a cabo la especificación del sistema).

- **Análisis**

Es necesario determinar qué elementos intervienen en el sistema a desarrollar, así como su estructura, relaciones, evolución en el tiempo, detalle de sus funcionalidades, ... que van a dar una descripción clara de qué sistema vamos a construir, qué funcionalidades va a aportar y qué comportamiento va a tener.

- **Diseño**

Tras la etapa anterior ya se tiene claro que debe hacer el sistema, ahora tenemos que determinar cómo va a hacerlo (¿cómo debe ser construido el sistema?), se pasará de casos de uso esenciales a su definición como casos expandidos reales, se seleccionará el lenguaje más adecuado, el Sistema Gestor de Bases de Datos a utilizar en su caso, librerías, configuraciones hardware, redes, etc. Se establecerá la arquitectura del software, la estructura y la caracterización de la interfaz. Esta fase es un paso intermedio entre lo que son las funcionalidades a alto nivel y la codificación.

- **Codificación (implementación)**

Llegado este punto se empieza a codificar algoritmos y estructuras de datos, definidos en las etapas anteriores, en el correspondiente lenguaje de programación y/o para un determinado sistema gestor de bases de datos. Es la traducción del diseño en una forma legible para la máquina. Si se ha hecho un buen diseño la codificación puede realizarse de manera totalmente mecánica.

- **Prueba**

El objetivo de estas pruebas es garantizar que el sistema ha sido desarrollado correctamente, sin errores de diseño y/o programación. Es conveniente que sean planteadas al menos tanto a nivel de cada módulo (aislado del resto), como de integración del sistema (según sea la naturaleza del proyecto en cuestión se podrán tener en cuenta pruebas adicionales, p.ej. de rendimiento). También tiene como objetivo la verificación de que el sistema desarrollado cumple con los requisitos expresados inicialmente por el cliente y que han dado lugar al presente proyecto (para esta fase también es interesante contar con los use cases, generados a través de las correspondientes fases previas, que servirán de guía para la verificación de que el sistema cumple con lo descrito por estos).

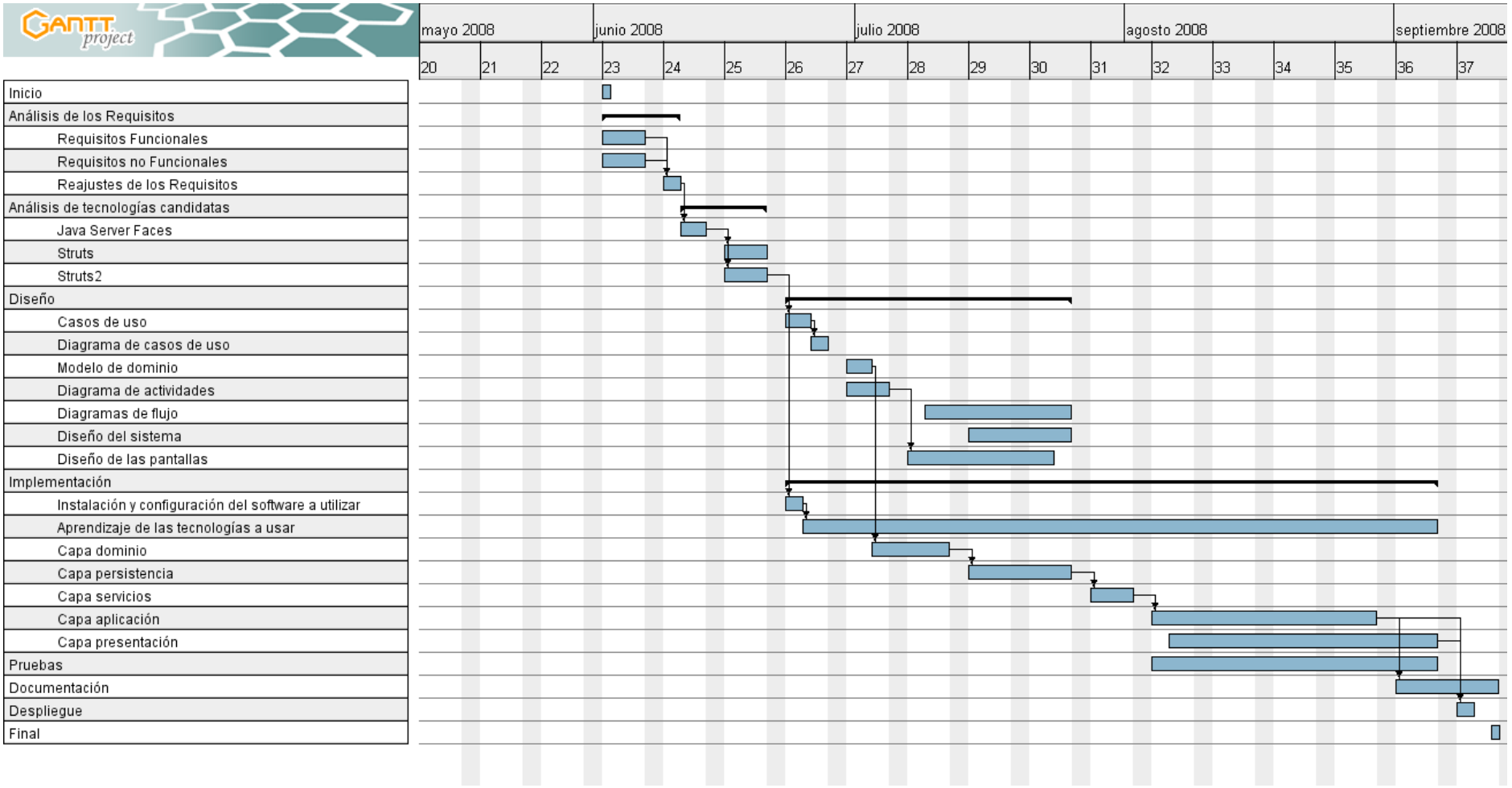
Mantenimiento

Finalmente la aplicación resultante se encuentra ya en fase de producción (en funcionamiento para el cliente, cumpliendo ya los objetivos para los que ha sido creada). A partir de este momento se entra en la etapa de mantenimiento, que supondrá ya pequeñas operaciones tanto de corrección como de mejora de la aplicación (p. ej. mejora del rendimiento), así como otras de mayor importancia, fruto de la propia evolución (p.ej. nuevas opciones para el usuario debidas a nuevas operaciones contempladas para el producto).

La mayoría de las veces en que se desarrolla una nueva aplicación, se piensa solamente en un ciclo de vida para su creación, olvidando la posibilidad de que esta deba sufrir modificaciones futuras (que tendrán que producirse con casi completa seguridad para la mayor parte de los casos).

Con respecto a las fases, a destacar que uno de los problemas que causa más fracasos en los proyectos de ingeniería es el hecho de no atribuir la suficiente importancia a las dos primeras etapas (Ingeniería del sistema y Análisis), que son la base sobre las que se sustentarán las demás. La causa principal es que se pretenden tener resultados de manera inmediata por diferentes motivos, y esto hace que se tengan prisas por comenzar a programar.

ANEXO I. Diagrama de Gantt



ANEXO J. Contenido adicional CD-VisualGate y despliegue

A continuación explicaré el contenido del CD-ROM adjunto a la memoria del proyecto, y como desplegar la aplicación en un servidor. Si se desea acceder a la aplicación web, sin tener que instalarla, he habilitado un servidor para poder acceder desde internet por medio de la URL:

- http://80.25.163.169:8080/visualGate

(El servidor, estará disponible desde las 10h-24h)

8.2.Contenido CD-VisualGate

- **visualGate.war**
  - Fichero que contiene el proyecto listo para ser desplegado en el contenedor de Servlets Apache Tomcat.
- **eclipseProject.rar**
  - Contiene el proyecto de eclipse.
- **VisualGate.pdf**
  - La memoria en formato PDF.

8.3.Despliegue de VisualGate

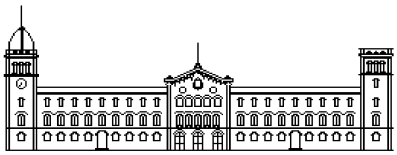
Para poder desplegar correctamente la aplicación VisualGate en el servidor, se han de cumplir una serie de requisitos previos:

- **Tener instalada, la máquina virtual java**
  - Para el proyecto se ha usado la versión JDK 6
  - http://java.sun.com/javaee/downloads/index.jsp
- **Tener instalado Apache Tomcat.**
  - Para el proyecto se ha usado la versión 6.0
  - http://tomcat.apache.org/download-60.cgi
- **Tener instalado MySQL**
  - Para el proyecto se ha usado la versión 5.0
  - http://dev.mysql.com/downloads/
- **Haber creado una base de datos llamada visualGate en MySQL**
  - “create database visualGate;”

Finalmente, solo tenemos que copiar el fichero visualGate.war dentro de nuestro Contenedor de Servlets Tomcat en la carpeta webapps:

- C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps

Iniciamos Apache Tomcat y desde el browser escribimos: http://localhost:8080/visualGate



ENGINYERIA TÈCNICA EN INFORMÀTICA DE SISTEMAS

UNIVERSITAT DE BARCELONA

Treball fi de carrera presentat el dia de de 200
a la Facultat de Matemàtiques de la Universitat de Barcelona,
Amb el següent tribunal:

Dr. President

Dr. Vocal 1

Dr. Secretari

Amb la qualificació de: