Licencia

Xavier Garcia Soto

Framework J2EE Capa de presentación

Tutor: Osca Escudero Sanchez

Alumne: Xavier Garcia Soto

UNIVERSITAT OBERTA DE CATALUNYA (UOC)

Contenido

Índice de imágenes

1. Introducción

Os presento el proyecto final de carrera de Java J2EE. Este proyecto comprende el estudio y análisis de diferentes Frameworks de presentación Java J2EE existentes en la actualidad. En este estudio veremos el patrón Model View Controller (MVC) que se ha convertido en un estándar para las aplicaciones Java.

Una vez analizados y aprovechando los conocimientos adquiridos, diseñaremos y crearemos un nuevo Framework para el desarrollo de aplicaciones Java J2EE.

Este Framework ha de ser el punto de partida para crear con la máxima celeridad posible y con una misma metodología aplicaciones web J2EE.

2. Agradecimientos

Quiero dar mi agradecimiento a mi familia. En especial a mi novia Raquel que durante estos últimos años ha tenido que aguantar y sufrir mis conversaciones sobre la entrega de prácticas, exámenes, pruebas de validación y la falta de tiempo libre que eso ha generado. A mis padres, José Antonio y Joanna, por animarme siempre a estudiar en estos tiempos difíciles que corren. También agradecer a mis amigos por su paciencia conmigo y apoyarme en los momentos bajos.

También agradecer a mis compañeros de trabajo, por echarme algún cable cuando alguna practica se encallaba y a mis superiores directos por ser comprensivos cuando en alguna jornada he podido salir antes para completar alguna entrega.

No me olvido de los profesores, consultores y tutores de la UOC, gracias por su dedicación y ganas de crear foros de debate en torno a este mundo complicado y muchas veces incomprendido.

Gracias también a Oscar Escudero, mi tutor de proyecto. Por animarme a continuar con sus comentarios en cada una de las entregas y guiarnos con sus aportaciones en la elaboración de todos los puntos del proyecto.

3. Descripción del PFC

3.1. Contexto del proyecto

Este proyecto surge a partir de la necesidad de evaluar el mercado actual de los Frameworks de la capa de presentación en el mundo Java J2EE. Una vez evaluados, crearemos un Framework para desarrollar aplicaciones web con celeridad y demostraremos su funcionamiento creando una aplicación web de Test.

J2EE (traducido informalmente y no sé hasta qué punto, como Java Empresarial), es una plataforma de programación para desarrollar y ejecutar software de aplicaciones en el lenguaje de programación Java. Permite utilizar arquitecturas de N capas distribuidas y se apoya ampliamente en componentes de software modulares ejecutándose sobre un servidor de aplicaciones.

Su modularidad ha hecho posible que sea una herramienta muy útil para realizar todo tipo de aplicaciones web. También hemos de tener en cuenta la cantidad de herramientas y documentación existente para implementar Java. Todo este cumulo de factores hace que el desarrollo de aplicaciones en tecnología J2EE esté creciendo día a día.

3.2. Objetivos generales y específicos

El objetivo general del proyecto es doble, inicialmente nos centraremos en el análisis y estudio de los Frameworks de la capa de presentación Java J2EE, para posteriormente diseñar un Framework propio con el fin de poder crear un conjunto de aplicaciones web J2EE para una pequeña empresa familiar llamada SignArt.

Si entramos mas al detalle, este proyecto abarca los siguientes aspectos:

- Estudio de los Frameworks J2EE más extendidos en la actualidad para el desarrollo de aplicaciones web:
    - Java Server Faces
    - Struts
    - Spring

- Análisis de los patrones de diseño J2EE: Emplearemos un alto porcentaje de la memoria a analizar y entender cómo funcionan los principales patrones de diseño J2EE. Alguno de ellos son el Intercepting Filter, Front Controller, Applicaion Controller, Service To Worker, Composite View,…

- Estudio del MVC (Modelo vista controlador): Es el patrón o modelo de referencia para aplicaciones web y por la tanto también para el Framework que crearemos.

- Diseño de nuestro Framework propio SingartFw para la capa de presentación. Una vez analizados los Framewoks existentes, los patrones de diseño y en qué consiste el modelo MVC aplicaremos lo aprendido para crear un Framework de trabajo que nos permita desarrollar aplicaciones J2EE de una manera sencilla.

- Implementación de una aplicación web utilizando el nuevo Framework de prestanción, SingartFw. En este caso nuestra aplicación servirá como punto de partida para que des de una pequeña empresa de rótulos de Barcelona, pueda desarrollar sus propias aplicaciones de gestión. Por ello llamaremos a la aplicación SignartApp.

- Documentación y guía de instalación para utilizar el nuevo Framework SingartFw y poder desarrollar nueva aplicaciones J2EE.

3.3. Estudio de posibilidades y solución aportada para una empresa particular

Como hemos comentado, realizaremos un nuevo FrameWork de trabajo, para una pequeña empresa familiar llamada Singart. Pero en qué consiste ?

Entendemos como Framework un conjunto estándar de conceptos, prácticas o criterios para enfocar unos tipos de problemática en particular, que sirve como referencia para resolver otros problemas de carácter similar.

Para comprobar que el Framework diseñado funciona correctamente y tomarlo como punto base para la creación de diferentes aplicaciones web en J2EE implementaremos una pequeña aplicación como test.

Signart es una pequeña empresa familiar (actualmente 6 trabajadores) que se dedica al diseño y elaboración de todo tipos de productos gráficos en los ámbitos de la rotulación, escenográfica e imagen corporativa. Os invito a visualizar su web por si queréis ver sus últimos trabajos http://www.signart.es/

Otra de las principales características del Framework que crearemos, SignartFw, es que tanto el código que utilizaremos como las aplicaciones o recursos utilizados son de licencia libre. Pon ello podremos implementar e instalar aplicaciones con este Framework de manera gratuita.

3.3.1. Frameworks J2EE por uso en el mercado

Actualmente existen varios Frameworks Java J2EE en el mercado. Los más utilizados se basan con el modelo MVC (Modelo, Vista y Controlador).

Dentro de los diferentes Frameworks Java existentes, pasaremos a analizar únicamente tres, los cuales son más utilizados actualmente en el entorno de aplicaciones empresariales. Estos serán JSF, Struts y Spring.

Hemos buscado información sobre el Framework Google web Toolkit, pero lo hemos descartado ya que es un Framework dirigido mas para Ajax y Javascript.

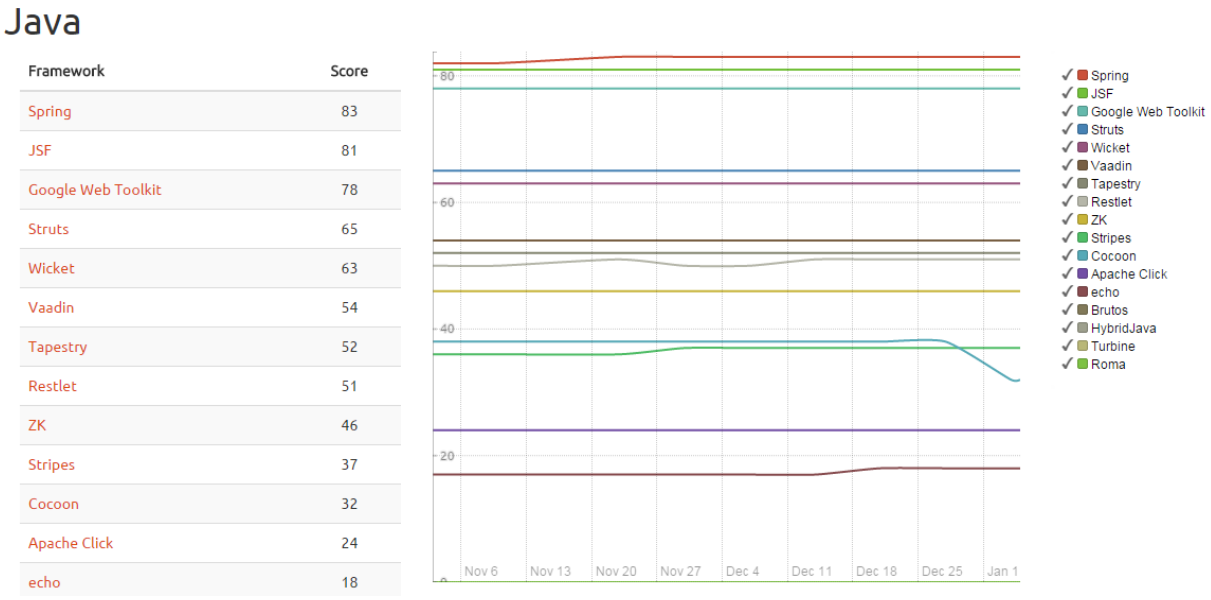


ILUSTRACIÓN 1 USO FRAMEWORKS JAVA

Esta gráfica la he obtenido por Internet desde la siguiente dirección: http://hotframeworks.com/languages/java

3.4. Planificación de proyecto

Teniendo en cuenta las diferentes tareas a realizar, hemos seguido la siguiente planificación para realizar el proyecto:

	Nombre de tarea	Duración	Comienzo	Fin	Predi
1	**1. Plan de trabajo**	20 días?	lun 01/09/14	dom 28/09/14	
2	2.1 Concepto de Patrón y catálogo de Patrones J2EE	6 días?	mié 01/10/14	mié 08/10/14	1
3	2.1 El patrón Model-Vista-Controlador y estudio de los principales FrameWorks	15 días?	jue 09/10/14	mié 29/10/14	2
4	2.3 Análisis y Diseño del FrameWork propio	5 días?	jue 30/10/14	mié 05/11/14	3
5	**2.Anàlisis i Disseny (Pac2) 05/11/2014**	26 días?	mié 01/10/14	mié 05/11/14	
6	3.1 Instalación entorno de trabajo	10 días?	jue 06/11/14	mié 19/11/14	4
7	3.2 Implementación Módulo empleados	8 días	jue 20/11/14	lun 01/12/14	6
8	3.3 Implementación Módulo clientes	8 días	mar 02/12/14	jue 11/12/14	7
9	3.4 Documentación de guia de instalación y manual de uso	5 días	vie 12/12/14	jue 18/12/14	8
10	**3. Implementació**	33 días?	mié 05/11/14	vie 19/12/14	
11	4.1 Instalación aplicación	6 días?	vie 19/12/14	vie 26/12/14	9
12	4.2 Conclusión de memòria	74 días?	mié 24/09/14	lun 05/01/15	
13	4.3 Presentación	6 días?	lun 05/01/15	lun 12/01/15	
14	**4.Entrega Final: Entrega Final - Memoria + Presentación + Prototipo**	96 días?	lun 01/09/14	lun 12/01/15	

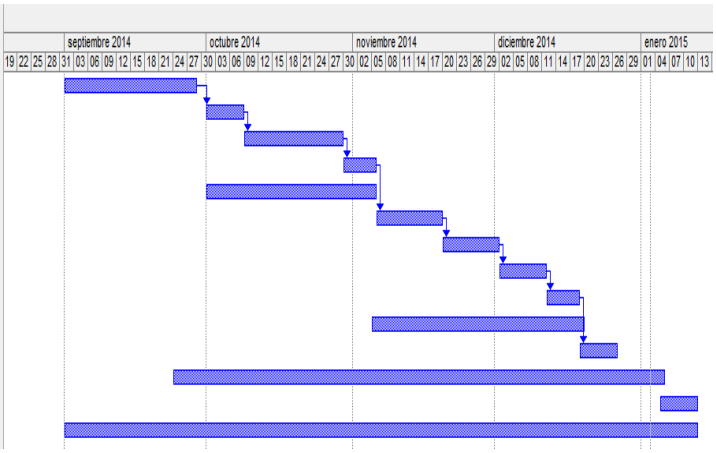


ILUSTRACIÓN 2 PLANIFICACIÓN PROYECTO

1. **Plan de trabajo (Pac1) 01/10/2014**
   - 2.1 Concepto de Patrón y catálogo de Patrones J2EE (01/10/2014 - 08/10/2014)
   - 2.2 El patrón Model-Vista-Controlador y estudio de los principales FrameWorks (16/10/2014 - 29/10/2014)
   - 2.3 Análisis y Diseño del FrameWork propio (23/10/2014 - 05/11/2014)
2. **Análisis y Diseño (Pac2) 05/11/2014**
   - 3.1 Instalación entorno de trabajo (06/11/2014 - 19/11/2014)
   - 3.2 Implementación Módulo empleados (20/11/2014 - 01/12/2014)
   - 3.3 Implementación Módulo clientes (02/12/2014 - 11/12/2014)
   - 3.4 Documentación: Guía de instalación y Manual de Uso (12/12/2014 - 18/12/2014)
3. **Implementación (Pac3) 19/12/2014**
   - 4.1 Instalación aplicación (22/12/2014 - 29/12/2014)
   - 4.2 Conclusión de memoria (24/09/2014 - 05/01/2014)
   - 4.3 Presentación (05/01/2014 - 12/01/2014)
4. **Entrega Final: Entrega Final - Memoria + Presentación + Prototipo 12/01/2015**

3.5. Producto Obtenido

El producto obtenido después de este proyecto final de carrera es un FrameWork de presentación que utilizaremos para desarrollar aplicaciones web en Java J2EE. Para implementarlo, utilizaremos base de datos MySql y el servidor de aplicaciones Tomcat. Ambos son de licencia gratuita, como hemos comentado antes.

Respecto a las tecnologías J2EE empleadas, nos basaremos en Hibernate como capa de persistencia para acceder a la base de datos, en Maven como herramienta para preparar el entorno de trabajo y en el módelo de Struts e Spring como arquitecturas Java.

Pero antes de adelantar acontecimientos, pasaremos a realizar un estudio del J2EE y sus patrones de diseño.

4. Concepto de Patrón y catálogo de Patrones J2EE

4.1. Introducción a los Patrones

Los analistas o programadores desarrollamos diariamente nuestras habilidades para resolver problemas que se presentan en la implementación de software. Para cada problema que se nos presenta, pensamos distintas formas de resolverlo. Algunas veces incluimos soluciones exitosas que ya hemos usado antes en problemas similares.

Conforme vamos adquiriendo experiencia, nuestro abanico de posibilidades para resolver un problema crece, pero al final siempre acabamos coincidiendo en una sola solución que mejor se adapta a nuestro problema. Si documentamos esta solución, podemos reutilizarla y compartir esa información que hemos aprendido para resolver de la mejor manera un problema específico. Esa "solución" reutilizable es lo que podemos definir como patrón.

En definitiva, tenemos que los patrones tratan los problemas que se repiten y que se presentan en situaciones particulares, con el fin de proponer soluciones a ellas. Con todo ello se ha generado una biblioteca de patrones que nos permite tener una base a partir de la cual podemos iniciar un desarrollo.

Hay patrones que describen soluciones para varios ámbitos del software, desde el análisis, hasta el diseño y desde la arquitectura hasta la implementación. Además, los patrones existen en diversas áreas de interés y tecnologías. Por ejemplo, hay un patrón que describe como trabajar con un lenguaje de programación específico o un punto de la industria específico, como la banca, gestión de facturas o sanidad.

4.2. Patrones J2EE

Con la aparición del J2EE como plataforma de programación, se crean todo un nuevo catálogo de patrones de diseño Java por parte Sun Microsystems. Sun tiene una biblioteca grandiosa, donde nos propone una serie de patrones para ayudarnos a la construcción de nuestras aplicaciones implementadas en lenguaje java.

De acuerdo al libro "J2EE PATTERNS Best Practices and Design Strategies", existen 5 capas en la arquitectura J2EE : Cliente, presentación, negocio, integración y Recursos.

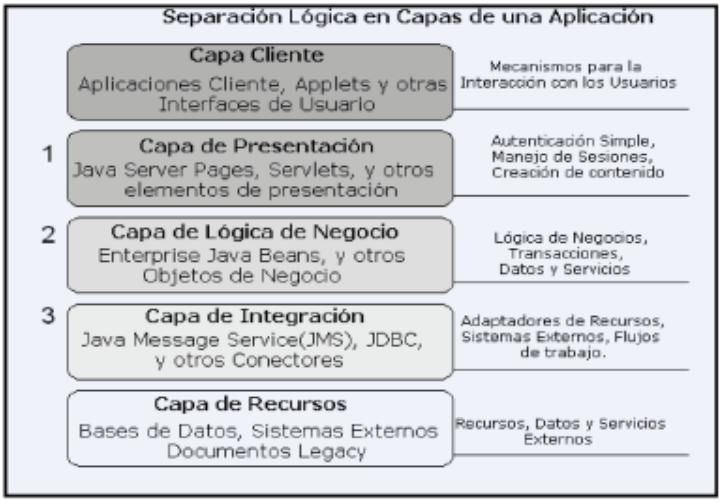


ILUSTRACIÓN 3 CAPAS DE ARQUITECTURA

Capa Cliente: Está formada por la lógica de la aplicación a la que el usuario final accede directamente mediante una interfaz de usuario. La lógica de la capa de cliente podría incluir clientes basados en navegadores, componentes de Java que se ejecutan en un equipo de escritorio o clientes móviles de dispositivos portátiles.

Capa Presentación: Está formada por la lógica de aplicación, que prepara datos para su envío a la capa de cliente y procesa solicitudes desde la capa de cliente para su envío a la lógica de negocios del servidor. Se encuentra en el servidor y recibe los datos del usuario desde la capa cliente y genera una respuesta apropiada a la solicitud.

Capa de Negocio: Consiste en la lógica que realiza las funciones principales de la aplicación: procesamiento de datos, implementación de funciones de negocios, coordinación de varios usuarios y administración de recursos externos como, por ejemplo, bases de datos o sistemas heredados. Se encuentra en el servidor y contiene el núcleo de la lógica del negocio de la aplicación. Provee las interfaces necesarias para utilizar el servicio de componentes del negocio.

Capa de Integración: Está formada por los servicios que proporcionan los datos utilizados por la lógica de negocios. Los datos pueden suelen ser datos de aplicaciones almacenados en una base de datos, pero pueden incluir información de otros recursos.

Capa de Recursos: Esta capa está formada por los datos externos a la aplicación. Comprende básicamente bases de datos o otros modelos de almacenaje de información como puede ser LDAP. Puede contener cierto tipo de lógica o funcionalidades pero lo encontraremos habitualmente en las bases de datos

Dentro de estas 5 capas de arquitectura, analizaremos los patrones más importantes y comunes que encontraremos en las capas de Presentación.

Los patrones de la capa de presentación, contienen toda la información relacionada con los Servlets y la tecnología JSP.

En esta figura se revisan, los principales patrones de diseño J2EE que pasaremos a analizar más al detalle:

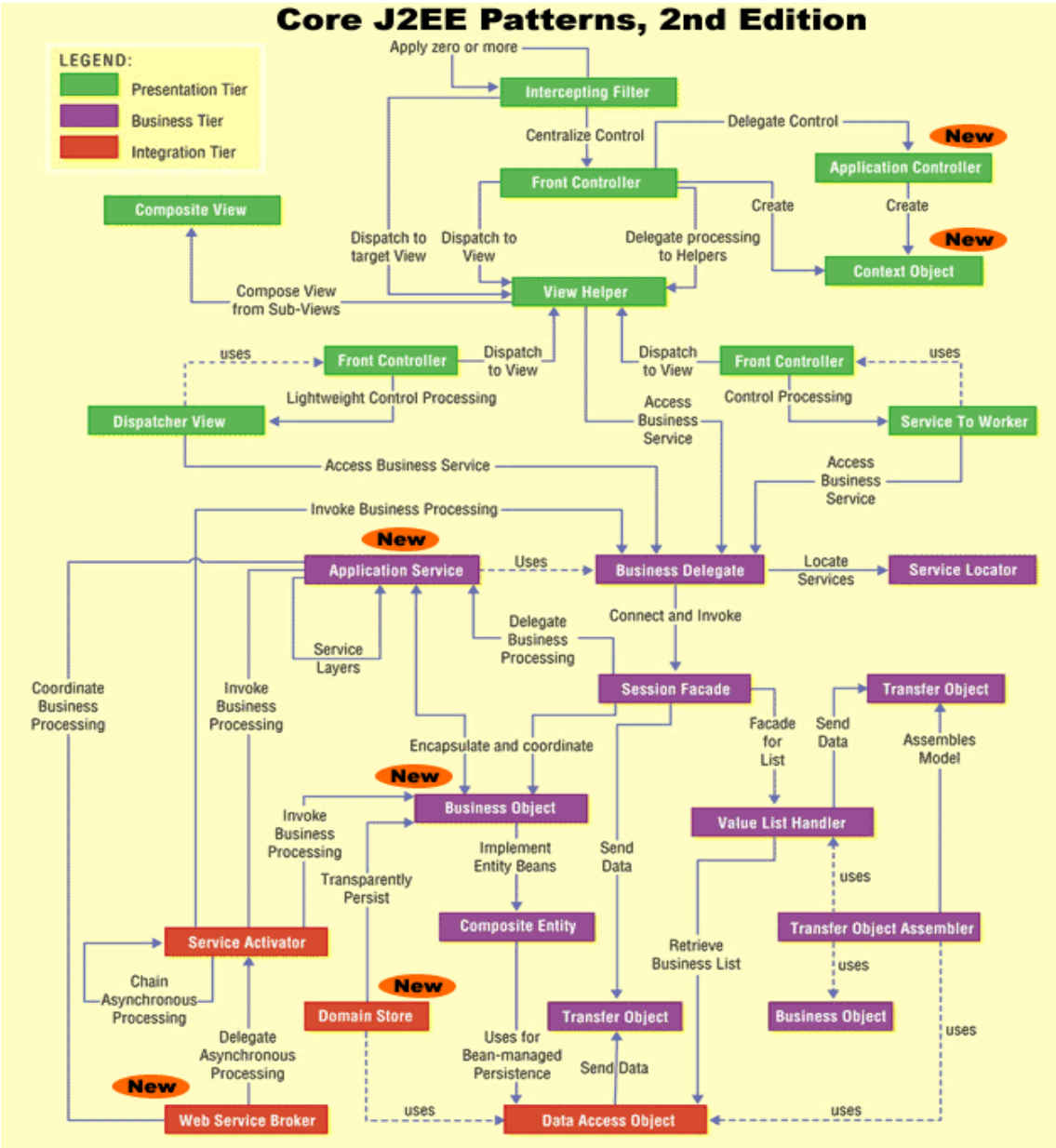


ILUSTRACIÓN 4 PATRONES J2EE

4.3. Plantilla de patrón

Antes de pasar a analizar los patrones J2EE, vamos a realizar una breve introducción de cómo podemos estructurar los patrones para que todos ellos tengan una misma estructura.

Podemos decir que existe una manera común de "redactar" los patrones en J2EE, de tal manera que podemos definir una plantilla en el momento de crear un patrón. Así cada sección de la plantilla del patrón contribuye a entender el patrón particular. Primeramente, observaremos que se ha intentado dar un nombre de patrón descriptivo. Aunque es difícil, se ha intentado que los nombres de patrón proporcionen suficiente información sobre la función del patrón.

La plantilla de cada patrón consta de las siguientes secciones:

- Contexto: Define el conjunto de entornos de trabajo bajo los cuales existe el patrón.
- Problema: Describe los problemas de diseño que se ha encontrado el desarrollador para implementar el patrón.
- Solución: Describe la solución y sus elementos en más detalle. Hace referencia a la estructura y las estrategias utilizadas.
- Implementación y Patrones Relacionados: Lista otros patrones relevantes en el catálogo de patrones J2EE u otros recursos externos relaciones con el patrón que se está implementando. Por cada patrón relacionado, hay una breve descripción de su relación con el patrón que se está describiendo.

4.4. Análisis de patrones de la capa de presentación

4.4.1. Intercepting Filter

La capa de presentación requiere el manejo de diferentes tipos de peticiones. Algunas peticiones simplemente requieren su reenvió al componente apropiado, mientras que otras peticiones deben ser modificadas, auditadas, o descomprimidas antes de continuar con su procesamiento.

Es necesario entonces un conjunto de acciones ante las peticiones o respuestas de un cliente web. Por ejemplo, cuando un usuario solicita una página Web, ésta puede requerir realizar varias validaciones previas antes de realizar el procesamiento principal de la página, ejemplo:

- El cliente ha sido autentificado?
- El cliente tiene una sesión válida dentro del sistema?

- La dirección IP del cliente, pertenece a una red conocida por nuestro sistema?
- El path requerido es correcto dentro de la aplicación?
- El sistema soporta la versión del navegador del usuario?

Mientras que algunos de estos chequeos son test del tipo SI o NO, otros requieren una modificación de la información de entrada a la página.

La solución planteada por este patrón es crear filtros que procesen los servicios comunes en una forma estándar, de tal forma que no requiera efectuar cambios en el código principal del procesamiento de la petición. Los filtros interceptan los datos de entrada y las respuestas de salida, permitiendo un pre-procesamiento y un post-procesamiento. Además permiten agregar o eliminar filtros, sin requerir una modificación del código existente.

El diagrama de clases para el patrón Intercepting Filter sería el siguiente:

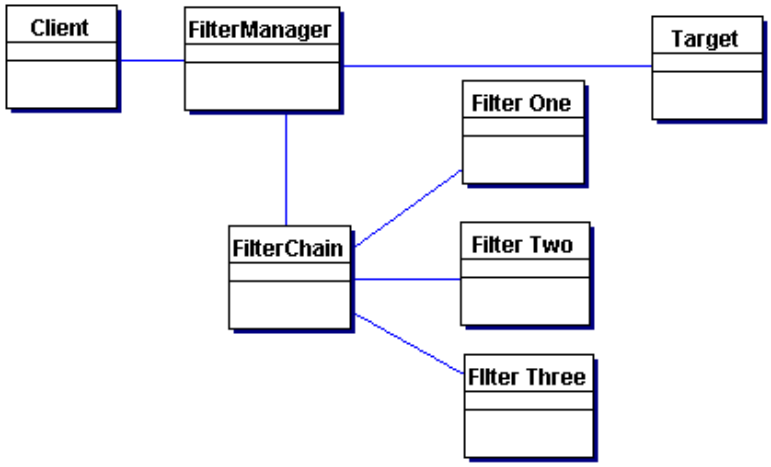


ILUSTRACIÓN 5 DIAG. CLASES INTERCEPTING FILTER

Como podemos ver en el diagrama se añaden diferentes tipos de Filtros que nos servirán para autentificar, verificar la sesión, analizar la IP,…

En el siguiente diagrama de secuencia de patrón Intercepting Filter podemos ver como cada uno de los filtros se va validando consecutivamente hasta obtener una respuesta completa.

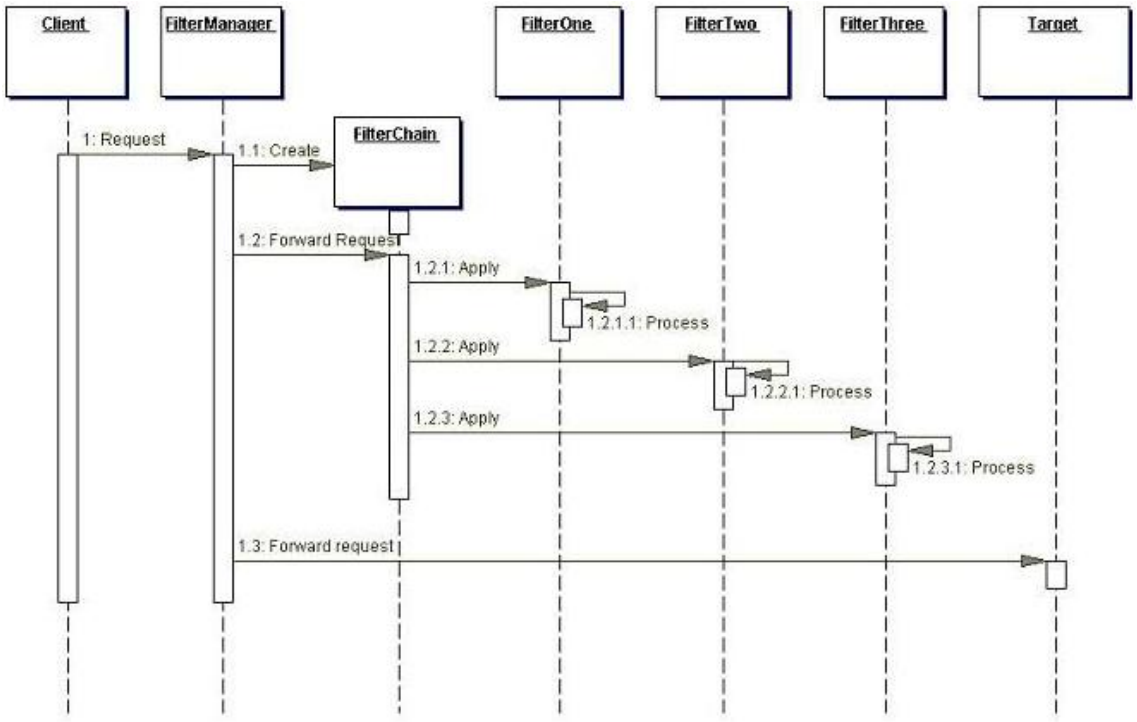


ILUSTRACIÓN 6 DIAG SECUENCIA INTERCEPTING FILTER

Este patrón está relacionado con el Patrón Front Controller y resuelven problemas similares. Mientras Intercepting Filter se centra el Filtrado de las peticiones el Front Controller elige las acciones convenientes a realizar con dicha petición.

4.4.2. Front Controller

En el momento de gestionar las peticiones que nos llegan por parte de los usuarios se debe controlar y coordinar las múltiples solicitudes web que nos hagan, teniendo en cuenta que deberíamos realizar las acciones de gestión y control las menos veces posibles. Este mecanismo debe ser centralizado, ya que si no deberíamos realizar dicha tarea para cada petición que recibamos.

Así aparece el controlador Front Controller, el cual está destinado a centralizar las siguientes tareas dentro de una petición:

- Manejar el control de las peticiones
- Invoca a los servicios de seguridad como el autentificación y autorización
- Delega el procesamiento a otro patrones de negocio
- Controla la elección de la vista adecuada
- Control de errores dentro de una petición
- Selecciona las estrategias de creación de contenido

Unificando dichos puntos en un mismo controlador ayudamos a reducir drásticamente el código Java a utilizar y el número de veces que deberíamos de utilizarlo.

Normalmente el controlador se coordina con un componente de dispatcher. Los dispatcher son responsables de control de las vistas sobre la cuales se va navegando. Así son los encargados de elegir la siguiente vista del usuario y enviar el control al recurso que toca. Estos dispatchers pueden ir dentro del propio controlador o de un componente separado.

En la siguiente figura representamos el diagrama de clases del patrón Front Controller:

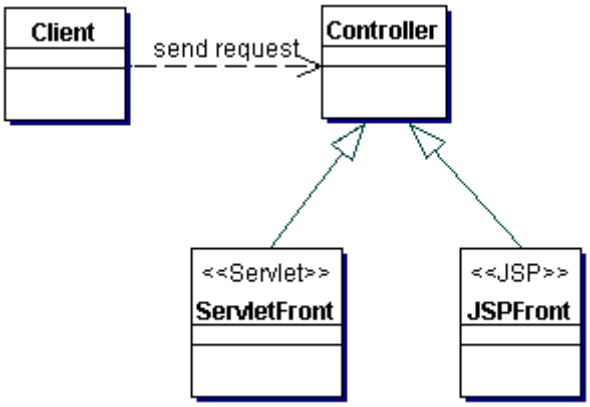


ILUSTRACIÓN 7 DIAG. CLASES FRONT CONTROLLER

En el siguiente diagrama de secuencia podemos observar como el controlador gestiona una petición y la aparición de los Dispatchers para gestionar la siguiente vista de navegación:

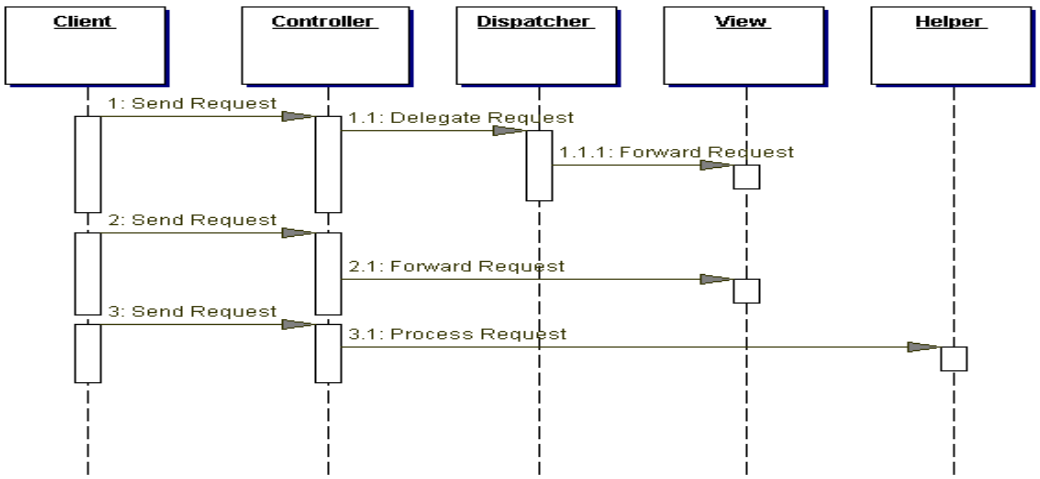


ILUSTRACIÓN 8 DIAD. SECUENCIA FRONT CONTROLLER

4.4.3. View Helper

Las aplicaciones manejan solicitudes web, por tanto los procesos de la capa de presentación requieren la generación de una vista basada en una plantilla y un modelo dinámico. Los cambios en la capa de presentación ocurren muy frecuentemente, son difíciles de desarrollar y mantener cuando la lógica de acceso a los datos del negocio y la lógica del formateo de la presentación de mezclan. Delegar la lógica de negocio en la capa de presentación hace que nuestra aplicación sea más modular y facilita la reutilización de componentes. Varios clientes, como controladores y vistas, podrían utilizar el mismo *Helper* para recuperar y adaptar estados del modelo similares para su presentación en varias vistas. De esta manera eliminamos redundancia en nuestros códigos.

El patrón view Helper encapsula la lógica correspondiente a la presentación y el acceso a datos. Normalmente estos Helper comunes suelen ser JavaBeans o etiqueta personalizadas.

Cuando se utilizan de hojas de estilo para dar formato a la vista también se está haciendo uso de este patrón.

Este sería el diagrama de clases que representa el Patrón View Helper:

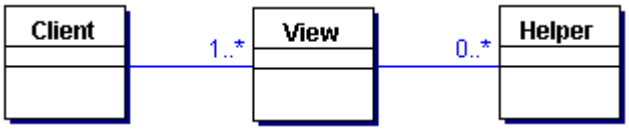


ILUSTRACIÓN 9 DIAG. CLASES VIEW HELPER

En la siguiente figura mostraremos el diagrama de secuencia del patrón View Helper.

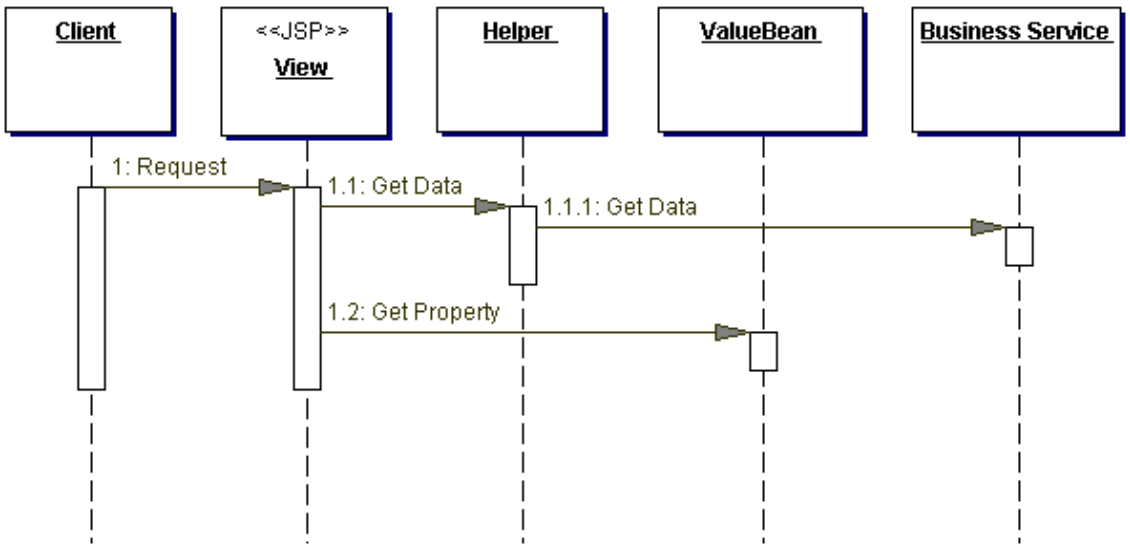


ILUSTRACIÓN 10 DIAG. SECUENCIA VIEW HELPER

Normalmente un controlador media entre el cliente y la vista, aunque también se puede dar el caso en que no se utilice controlador y la vista realice las dos funciones.

4.4.4. Composite view

Las páginas web sofisticadas presentan contenidos comunes que se repiten una y otra vez en la capa de presentación. La modificación de las mismas se hace complicada ya que un cambio que debería ser simple se tiene que realizar varias veces.

Con esta necesidad aparece el patrón Composite view, que es simplemente un objeto vista que está compuesto de otros objetos vista. Por ejemplo, una página JSP que incluye otras páginas JSP y HTML usando la directiva include o el action include es un patrón Composite View. De esta manera cada componente de la plantilla se pude incluir dentro de un total y la modificación y distribución de la página se maneja mas independientemente.

El siguiente diagrama de clases representa el patrón Composite View:

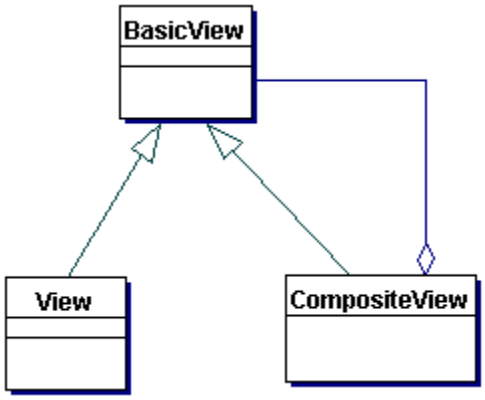


ILUSTRACIÓN 11 DIAG. CLASES COMPOSITE VIEW

Mientras que la siguiente figura representaría el diagrama de secuencia del patrón Composite View.

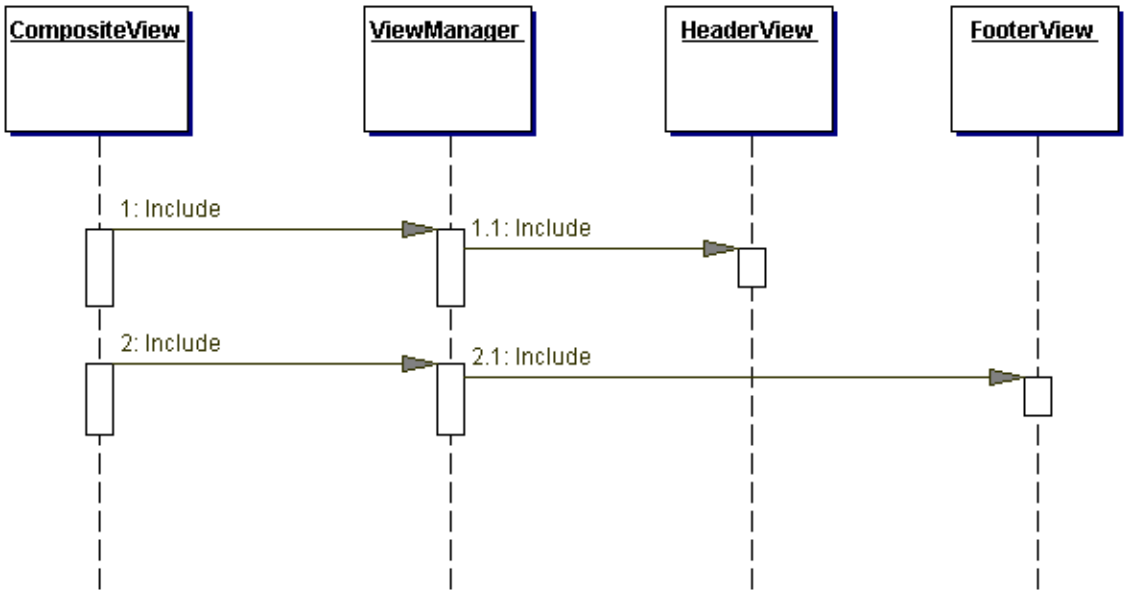


ILUSTRACIÓN 12 DIAG. SECUENCIA COMPOSITE VIEW

En este caso podemos observar como el patrón Composite View contiene el ViewManager que a su vez nos incluye una cabecera y un pie de página.

4.4.5. Service To Worker

Después de haber estudiado los patrones Front Controller y View Helper vemos aquí que combinando dichos patrones obtenemos una respuesta común para utilizar un Controlador, un dispatcher con vistas y un Helper. No obstante, no existe un componente común que centralice la gestión del control de acceso, la recuperación de contenido, la administración de las vistas. Con ello tenemos que existe código de control duplicado y esparcido en múltiples vistas y la lógica de formateo de presentación están mezcladas en estas vistas.

Para resolver estos problemas aparece el patrón Service To Worker. Su finalidad principal es Combinar un controlador, un dispatcher con vistas, y helpers para gestionar las peticiones de los clientes, y preparar una respuesta. Sus principales características son:

- Los Controllers delegan la recuperación de contenido a los helpers.
- Los dispatcher son los responsables de la gestión de vistas y de la navegación.

En la siguiente figura podríamos ver el diagrama de clases que representaría el patrón Service To Worker. Si nos fijamos es una combinación entre Controller, Dispatcher, View y Helper:

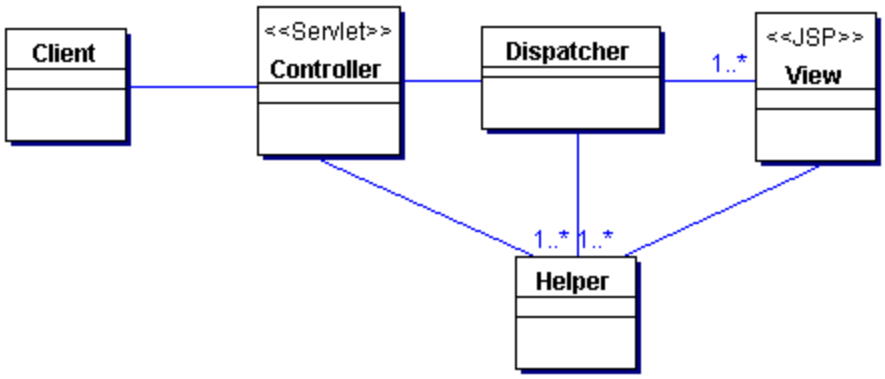


ILUSTRACIÓN 13 DIAG CLASES SERVICE TO WORKER

En el siguiente diagrama de secuencia vemos como se combinar los elementos Controller, Dispatcher, Helper y View dentro del patrón Service to Worker:

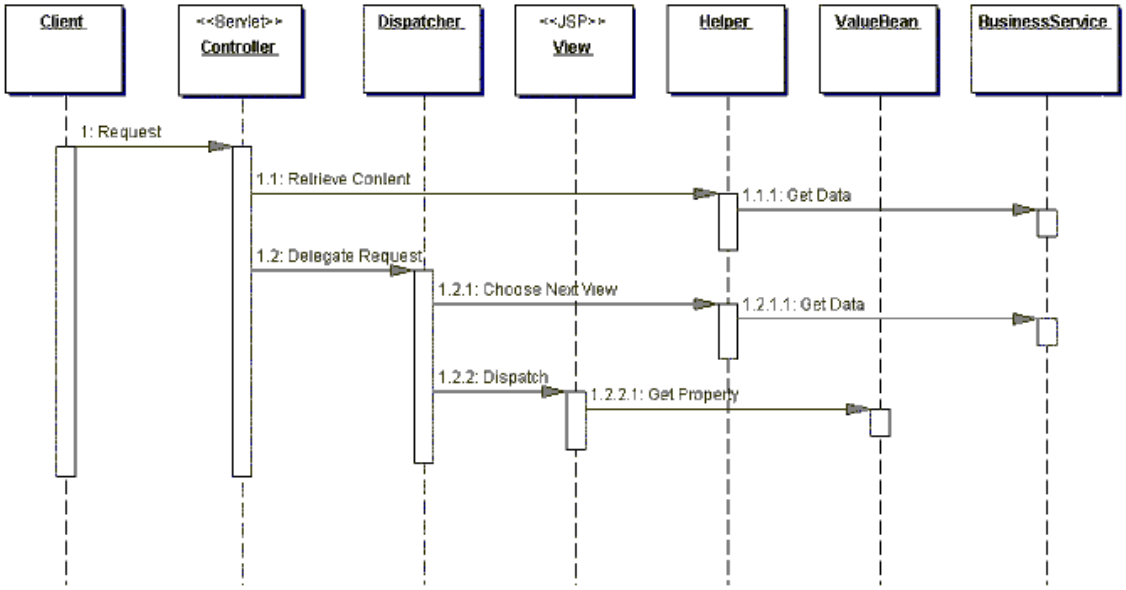


ILUSTRACIÓN 14 DIAG SECUENCIA SERVICE TO WORKER

4.4.6. Dispatcher view

Este patrón es similar al anterior Service to Worker y tienen una funcionalidad similar. El objetivo principal de este patrón es centralizar la gestión del control de acceso, la recuperación de contenidos o la gestión de las vistas. Combinando un controlador y un dipatcher con vistas (View) y centralizándolo en un mismo patrón se centraliza el control y se optimiza el código.

La diferencia básica entre los dos controladores es que el Service to Worker garantiza un comunicación más eficiente entre desarrolladores, mientras que el Dispatcher View recupera la información en el momento de procesar la propia vista.

En la siguiente imagen podemos observar el diagrama de clases que representa el Dispatcher View:

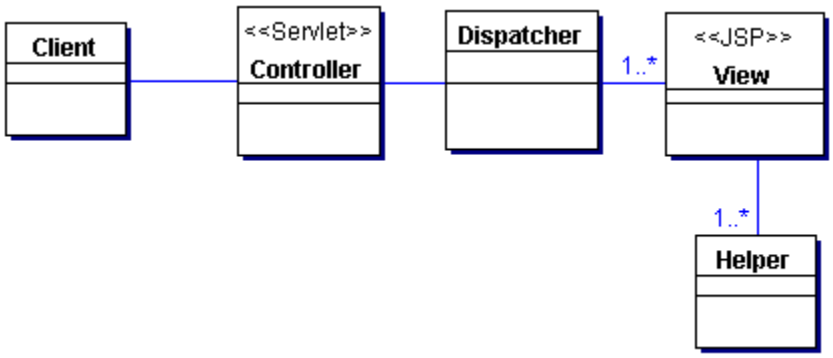


ILUSTRACIÓN 15 DIAG CLASES DISPATCHER VIEW

En la siguiente figura podemos observar el diagrama de secuencia del Dispatcher View:

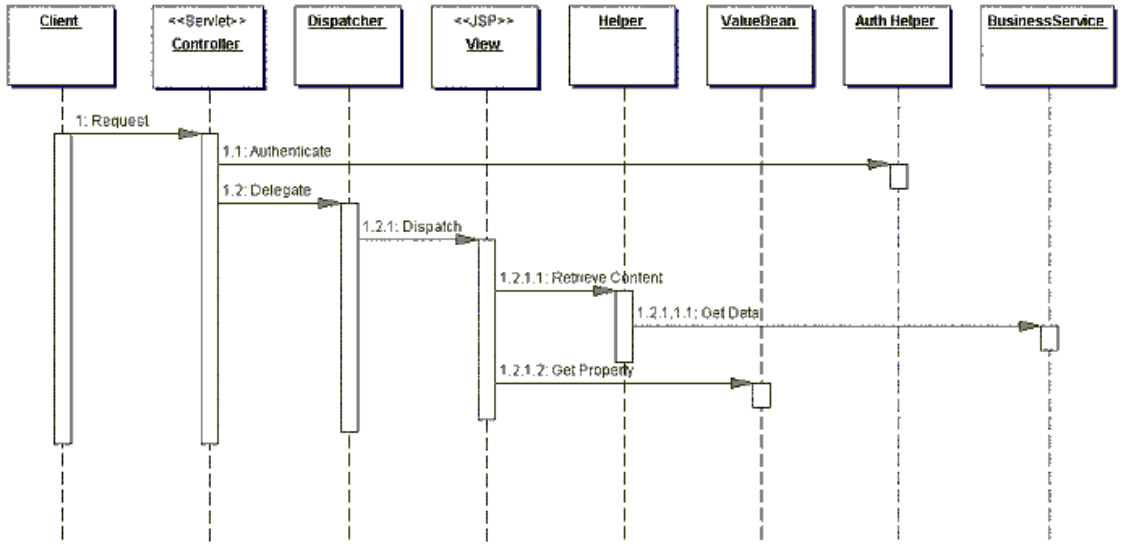


ILUSTRACIÓN 16 DIAG SECUENCIA DISPATCHER VIEW

5. El patrón Model-Vista-Controlador y estudio de los principales FrameWorks

5.1. "Divide et impera": El Modelo Vista Controlador

En la actualidad tenemos muchas tecnologías y lenguajes que los programadores tienen a su disposición para el desarrollo de aplicaciones. Para facilitar las cosas y tener éxito en los diferentes proyectos es necesario establecer un modelo o esquema común que permita estructurar lo máximo posible y poder reutilizar el mayor código posible.

Partiendo de la base de uno de los lemas de la informática (o de la vida) "**Divide y vencerás**", nace uno de los esquemas que actualmente "impera" en el mundo de los desarrolladores J2EE. Presentamos así el Modelo Vista Controlador (MVC), donde se separan claramente las distintas responsabilidades de la aplicación en tres grandes capas.

Así tenemos que cuando hablamos de patrón de arquitectura MVC es un patrón de diseño que especifica cómo se debe dividir o estructurar una aplicación en 3 grandes bloques:

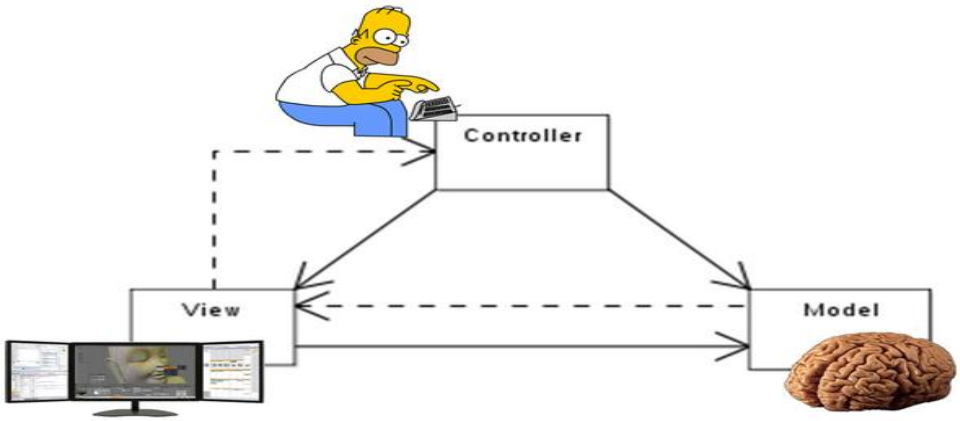


ILUSTRACIÓN 17 ESQUEMA MVC

Modelo: En esta capa encontraremos **el cerebro, el núcleo** de la funcionalidad de una aplicación. Así tendremos la lógica de la aplicación, incluyendo en este punto la manipulación y acceso a los datos de la aplicación. Entendemos como acceso a los datos, la interactuación con la base de datos. Es totalmente independiente a las otras capas de Controlador y Vista.

Vista: Como su propio nombre indica, la vista es la encargada de generar las respuestas que son enviadas a cliente. Podríamos decir que es **la presentación** en Sociedad de los datos que ha obtenido el modelo. Esta capa presenta lo datos pero

no los modifica. En definitiva podríamos concluir con que la Vista es la interfaz que percibe el usuario. No hay que olvidar que pese a que "aparentemente" no parece tan importante como las otras, podríamos decir que es la capa con la que el usuario interactuará, con lo cual es determinante para el éxito de una aplicación.

Controlador: En esta capa nos encontraremos al **director de orquesta** de la aplicación. Todas la peticiones que se realizan des del cliente son dirigidas al controlador. Su misión, será determinar que acciones se van a realizar con las peticiones del cliente y consecuentemente invocar al Modelo y Vista para obtener un resultado satisfactorio.

El comportamiento típico de esta arquitectura sería el siguiente:

- Cliente a través de la vista, solicita una petición/acción sobre los datos.
- El controlador recibe los datos de la vista, los analiza e invoca las acciones que realizará el Modelo.
- El Modelo actúa, busca o modifica, los datos que el Controlador le ha dicho y le devuelve respuesta al Controlador.
- El controlador captura estos datos y se los enviará a la vista que corresponde para que se los presente a cliente.

En este pequeño esquema podemos ver este comportamiento típico:

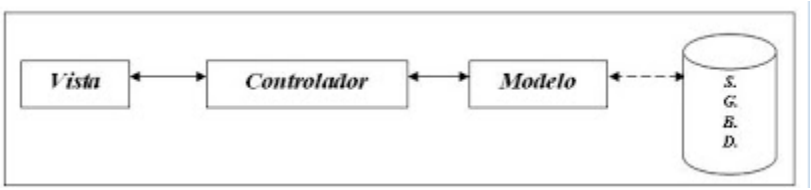


ILUSTRACIÓN 18 ESQUEMA 2 MVC

5.2. Java Server Faces

ILUSTRACIÓN 19 JSF LOGO

5.2.1. Introducción

Los Java Servlet Frames es un framework para crear aplicaciones J2EE basados en el patrón de diseño Modelo Vista Controlador. Este modelo se basa en las JavaServer Pages (JSP) como la tecnología que permite hacer el despliegue de las páginas, pero también permite hacer el despliegue de otras páginas como XML.

Los JSP (Java Server Pages) es una tecnología Java que se utiliza para la creación de páginas web con programación en el servidor. Las páginas JSP están compuestas de código HTML/XML mezclado con etiquetas especiales para programar scripts que tienen sintaxis Java.

Una de las principales características del Framework Java Server Faces, es que en la capa de la vista, ya conoce la acción que se va a invocar en la petición que haga el usuario. Con ello el modelo de vista gana peso al del Controlador, ya que en cierta manera dirige las peticiones que realiza el usuario.

5.2.2. Características principales JSF

- Como hemos comentado, utiliza páginas JSP para generar las vistas, añadiendo una biblioteca de etiquetas propia para crear los elementos de los formularios HTML con más facilidad y de manera estándar.
- Se relaciona a cada vista con datos de formularios, un conjunto de objetos java manejados por el controlador (managed beans) que facilitan la recogida, manipulación y visualización de los valores mostrados en los diferentes elementos de los formularios.
- Introduce una serie de etapas en el procesamiento de la petición, como por ejemplo la de validación, reconstrucción de la vista, recuperación de los valores de los elementos, etc.
- Utiliza un sencillo fichero de configuración para el controlador, en formato xml.
- Es extensible, pudiendo crearse nuevos elementos de la interfaz o modificar los ya existentes.
- Forma parte del estándar J2EE. Hay muchas alternativas para crear la capa de presentación y control de una aplicación web java, como Struts y otros frameworks, pero solo JSP forma parte del estándar.

5.2.3. Ventajas del JSF frente a otros FrameWorks

Gracias a su sencillez, JSF nos permite desarrollar rápidamente aplicaciones dinámicas en las que toda la lógica de negocio se implementa en java, o es llamada desde java, creando páginas para las vistas muy sencillas. Con todo ello tenemos que JSF nos ofrece una serie de ventajas:

- El código JSF con el que creamos las vistas (etiquetas jsp) es muy parecido al HTML estándar. Lo pueden utilizar fácilmente desarrolladores y diseñadores web.
- JSF se integra dentro de la página JSP y se encarga de la recogida y generación de los valores de los elementos de la página.
- JSF resuelve validaciones, conversiones, mensajes de error e internacionalización (i18n).

- JSF permite introducir java script en la página, para acelerar la respuesta de la interfaz en el cliente (navegador del usuario).
- JSF es extensible, por lo que se pueden desarrollar nuevos componentes a medida, También se puede modificar el comportamiento del framework mediante APIs que controlan su funcionamiento.
- El desarrollo de JSF está realmente empezando. Las nuevas versiones del framework recogen la funcionalidad de versiones anteriores siendo su compatibilidad muy alta, de manera que el mantenimiento de aplicaciones no se ve penalizado por el cambio de versiones.

5.2.4. Inconvenientes del JSF frente a otros Frameworks

El principal inconveniente en el momento de desarrollar proyectos en JSF es que su desarrollo puede ser más largo de lo realmente necesario. Los puntos donde se dedica más esfuerzo en el desarrollo de JSF son:

- Debemos adaptarnos a utilizar la herramienta alicate para clavar. JSF. Si nos empeñamos en seguir desarrollando las páginas como siempre, intentando adaptar JSF al modo al que habitualmente desarrollamos en vez de adaptarnos a JSF complicaremos el desarrollo y se puede hacer más engorroso de lo debido.
- Abuso del javascript. JSF permite utilizar javascript para hacer más rápida una página html, evitando peticiones al servidor. Sin embargo la introducción de javascript en la página complica y alarga los desarrollos con JSF, y en general con jsp. La capa javascript añade etapas adicionales a la aplicación, que hace más difícil su depurado.
- La maquetación compleja también complica el desarrollo en JSP ya que obliga a utilizar muchas etiquetas y atributos, especialmente en los datatables. Si la maquetación de nuestras páginas es compleja deberíamos pensar en crear componentes JSF a medida que simplifiquen dicho trabajo.

5.2.5. Ciclo de vida del JSF

E l circulo de vida de una aplicación JSF describe todas las acciones llevadas a cabo des de que el cliente inicializa la sesión. La mayoría de las fases son gestionadas por el Framework JSF, dejando solo una péquela parte de responsabilidad al desarrollador. Podríamos resumir así su ciclo de vida en estas 6 fases:

- Crea o restaura la vista
- Aplica los valores del request
- Ejecuta las validaciones
- Actualiza el modelo

- Invoca a la lógica de la aplicación
- Devuelve respuesta

En la siguiente figura y pintadas en verde podemos verlas representadas:

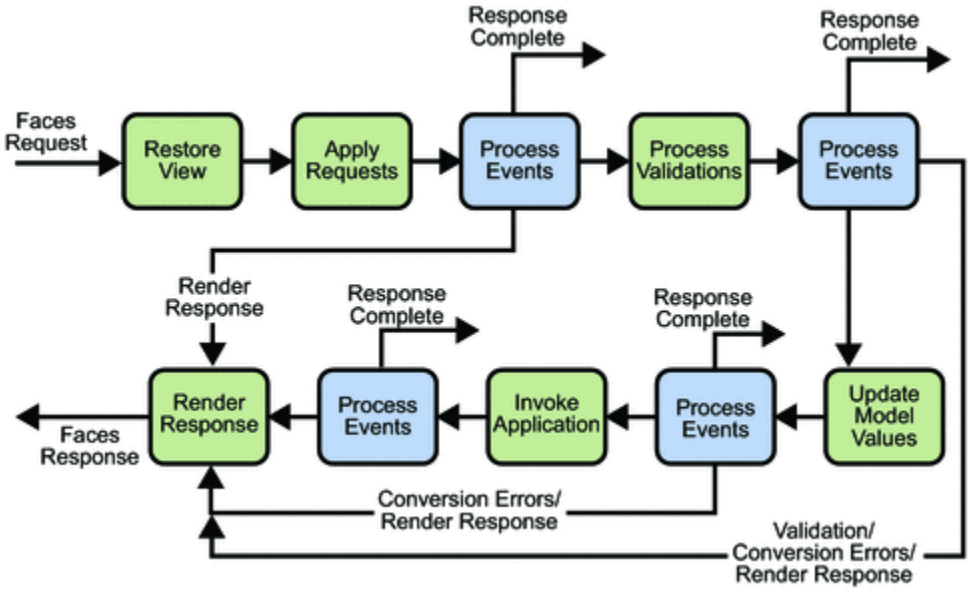


ILUSTRACIÓN 20 JSF CICLO DE VIDA

5.3. Struts

5.3.1. Introducción

En el año 2000 surgió Apache Struts con el objetivo de facilitar el desarrollo de aplicaciones web y se convirtió en un estándar para el desarrollo de aplicaciones web durante varios años coincidiendo y en parte siendo responsable del "boom" de las empresas punto com.

Struts permite reducir el tiempo de desarrollo drásticamente. Su carácter de "software libre" y su compatibilidad con todas las plataformas en las que Java Enterprise está disponible lo convierten en una herramienta altamente potente y disponible.

En 2013, y tras ser durante una década uno de los Frameworks líderes en el desarrollo J2EE, Struts 1 llegó al final de su vida y dejó de estar oficialmente soportado.

Así apareció Struts 2, como un nuevo framework desarrollado entre una mezcla de Struts 1 y otro Framework J2EE llamado WebWork. Así se introdujo algunas mejoras sobre Struts 1, de cara a simplificar las tareas más comunes en el desarrollo de aplicaciones web, así como mejorar su integración con AJAX.

La aportación de WebWork es decisiva, ya que algunas de las librerías más importantes de Struts 2 se basan en webWork. Los ejemplos más claros lo veremos comparando librerías como: struts2-core.jar o xwork-core.jar.

ILUSTRACIÓN 21STRUTS LOGO

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.

5.3.2. Características principales Struts

Las principales característica de Struts son las siguientes:

- Framework basado en acciones (actions)
- Todas las clases del Framework están basadas en interfaces y su núcleo principal es independiente del http.
- Configuración mediante anotaciones o un fichero XML
- Acciones basadas en POJO más simples de probar
- Integración con Spring, SiteMesh y Apache Tiles.
- Integración con OGNL
- Temas de presentación basados en librerías de etiquetas (tags)
- Etiquetas para AJAX, esto hace que la aplicación sea más dinámica.
- Múltiples opciones para la capa de vista (JSP, Freemarker, Velocity y XSLT)
- Framework extensible y modificable a través de plugins.

5.3.3. Ventajas de Struts frente a otros Frameworks

Podríamos decir que las principales características que hacen a Struts un Framework diferente a otros son las siguientes:

- 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.

5.3.4. Ciclo de vida en Struts 2

Vamos a realizar un pequeño análisis de como es el ciclo de vida de una petición Struts 2. Para ello podemos ver la siguiente figura donde está representado dicho ciclo:

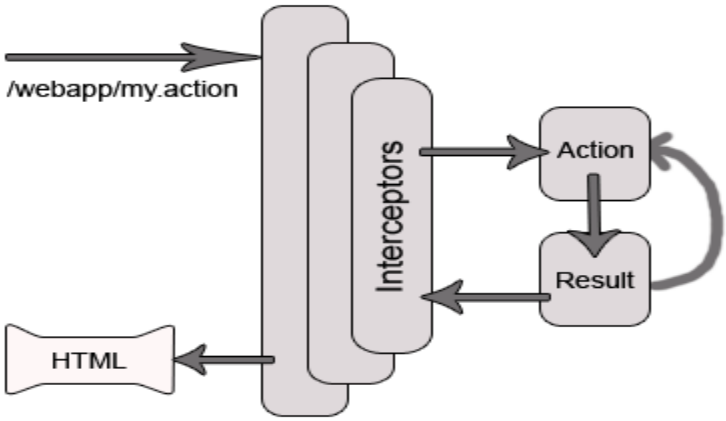


ILUSTRACIÓN 22 STRUTS CICLO DE VIDA

Inicialmente el usuario envía la petición al servidor solicitando algún recurso. En el propio formulario el FilterDispatcher determina la acción apropiada a realizar. En este punto y en el resultado final de la petición es donde encontramos la capa de Vista. Esta comprendida por Jsps con formularios que son los encargados de enviar y recibir la información mediante parámetros.

Los llamados Interceptors realizan la función de controlador dentro de la aplicación y se encargaran de realizar las funcionalidades comunes: control del flujo de trabajo, validación de datos, carga de archivos necesarios, … y dirigirán a dónde queremos realizar la acción.

Posteriormente se ejecuta la acción concreta de la capa de modelo. Normalmente es el bean con los métodos getXx() y setXx() que realiza la acción o consulta contra la bbdd de la cual obtenemos un resultado que bien puede ser un cálculo o simplemente un petición realizada a la base de datos.

Una vez tenemos el resultado, se devuelve a la capa de modelo. En este caso al interceptor que había solicitado dicha acción. Posteriormente el interceptor pasará los datos hacia el Servlet.

Del Servet irán directamente a la capa de Vista y con a la salida de jsp o html correspondiente. El Jsp se encarga de presentar correctamente los datos al usuario mediante el navegador.

5.3.5. Arquitectura de Struts 2

En este punto vamos a profundizar en algunos puntos de cómo es la arquitectura del Struts 2. En la siguiente figura se puede resumir como resulta la arquitectura de Struts 2 y los conjuntos de elementos que soporta.

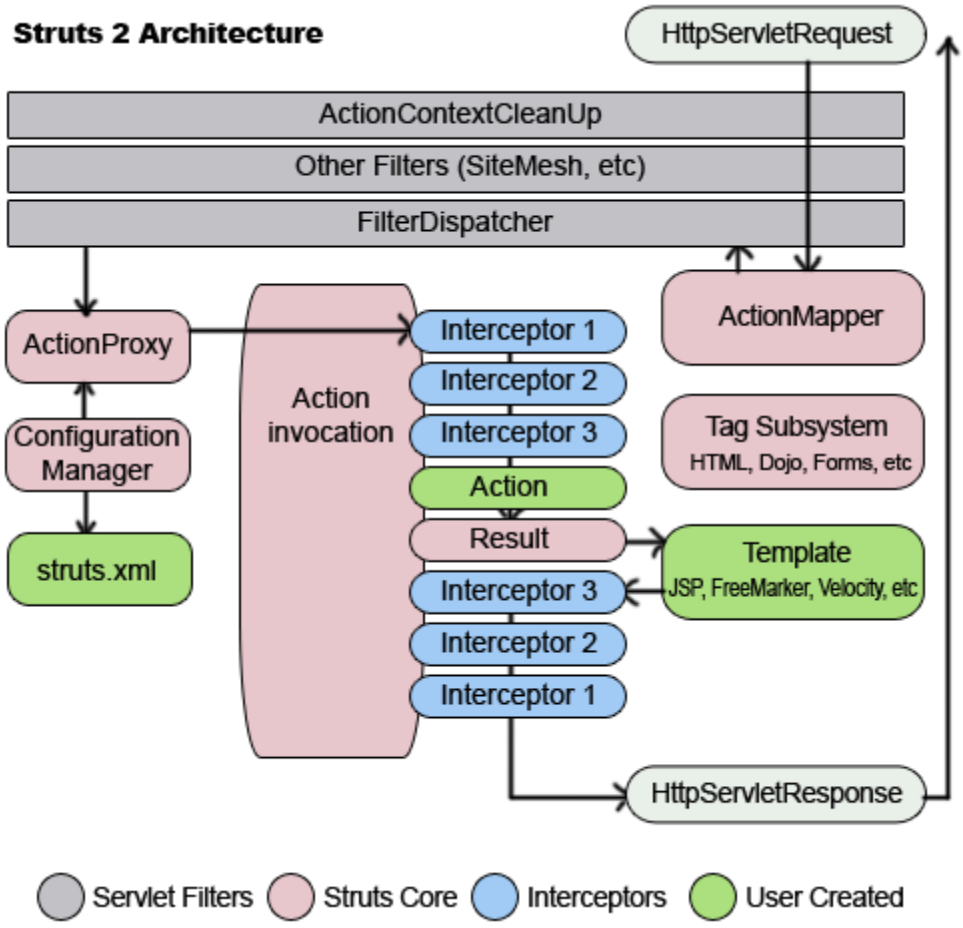


ILUSTRACIÓN 23 STRUTS ARQUITECTURA

Como podemos ver Struts 2 posee un abanico de controladores muy elegante y flexible, sobre la base de muchas otras tecnologías estándar como filtros de Java, Java Beans, ResourceBundles, XML,...

Para acceder a la capa de modelo, como no podía se de otra manera, también tenemos diferentes opciones para acceder a los datos: JDBC, EJB, Hibernate, etc

Finalmente para la vista, Struts 2 soporta JSP, Apache Tiles, plantillas Velocity, Sitemesh, PDF, XSLT, CSS…

Los interceptores, como hemos visto en el ciclo de vida, se utilizan para especificar una acción. Están configurados para aplicar las funcionalidades comunes como el flujo de trabajo, validación de la solicitud, etc.

Referente a los filtros podemos decir que contienen:

- FilterDispatcher: El filtro FilterDispatcher utiliza el ActionMapper para determinar si se debe invocar una acción o no. Si la acción se requiere que se invoque, el FilterDispatcher cede el control al ActionProxy.

- ActionProxy: El ActionProxy revisa la configuración de la aplicación a través del fichero struts.xml. A continuación, el ActionProxy crea un ActionInvocation, que implementa el patrón comando lanzando un proceso en el que invoca a los interceptores y luego llama a la acción. Una vez se obtiene el resultado, se pasa a la parte de presentación de la capa de vista (JSP, plantillas, etc) volviéndose a ejecutar los interceptores en orden inverso.

Otro punto importante que vemos en el cuadro es el fichero de configuración o struts.xml que se utiliza para iniciar sus propios recursos. Estos recursos incluyen:

- Los interceptores que pueden preprocesar y postprocesar una solicitud

- Clases de acción que pueden llamar a la lógica de negocio y el código de acceso a datos

- Los resultados que se pueden preparar en la capa de vista usando JSP, plantillas Velocity y FreeMarker.

- Durante la ejecución, hay una sola configuración de una aplicación. Hay varios elementos que pueden ser configurados, incluyendo los paquetes, espacios de nombres que incluyen acciones, resultados, interceptores y manejadores de excepciones.

Así el archivo struts.xml es el archivo de configuración básica para el marco y que deben estar presente en el *classpath* de la aplicación web. Vamos a ver un pequeño ejemplo de struts.xml y como se llama a una acción "HelloWorld" para que nos hagamos un idea de cómo funcionaria.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
"http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
<constant name="struts.enable.DynamicMethodInvocation" value="false" />
<constant name="struts.devMode" value="true" />
<include file="invoices-config.xml" />
<include file="admin-config.xml" />
```

```
<include file="reports-config.xml" />

<package name="example" namespace="/example" extends="struts-default">

  <action name="HelloWorld"
    class="org.example.Struts2HelloWorld">
    <result>/pages/HelloWorld.jsp</result>
  </action>

  <!-- Add actions here -->
</package>

<bean type="org.example.ObjectFactory" name="factory" class="org.example.MyObjectFactory" />

<!-- Add packages here -->
</struts>
```

La etiqueta de <struts> es la etiqueta raíz de la struts.xml. Puede contener los siguientes tags: paquete, include, bean y constante. Vamos a analizar un poco estas etiquetas:

Los paquetes (package) son una forma de agrupar acciones, resultados, tipos de resultados e interceptores. Conceptualmente, los paquetes son similares a los objetos que pueden ser extendidos y tienen las partes individuales que se pueden reemplazar por subpaquetes. Los atributos que presentan los veremos en la siguiente lista:

Atributo	Obligatorio	Descripción
name	**sí**	clave para otros paquetes para hacer referencia a
extends	no	Hereda el comportamiento del paquete del que extiende
namespace	no	Proporciona una asignación de la dirección URL para el paquete.
abstract	no	Declara un paquete como abstracto (no requiere configuraciones de acción en el paquete)

Sobre la etiqueta de include podemos decir que se utiliza para moldear una aplicación Struts por si necesitara incluir otros archivos de configuración extra.

En cuanto al bean, viene destinado por si nuestra aplicación de ejemplo necesitara extender la configuración java con otro bean. Los atributos de un bean son los siguientes:

Atributo	Necesario	Descripción
class	**sí**	el nombre de la clase bean

Type	no	la interfaz principal de esta clase de Java implementa
name	no	el nombre único de este grano, debe ser único entre los beans de otros que especifican el mismo type
scope	no	el ámbito del bean, puede ser default, singleton, request, session, thread

Por último podemos identificar las constantes que se utilizan como variables que permanecerán en todo el proyecto creadas y siempre serán consultables des cualquier punto de la aplicación.

5.4. Spring

ILUSTRACIÓN 24 SPRING LOGO

5.4.1. Introducción

Spring es un Framework de trabajo OpenSource para plataforma Java, cuyo código fuente está bajo la licencia de Apache. La primera versión fue escrita por Rod Jonhson, quien lo lanzó primero con la publicación de su libro *Expert One-on-One Java EE Design and Development* (Wrox Press, octubre 2002). El framework fue lanzado inicialmente bajo la licencia Apache 2.0 en junio de 2003. El primer gran lanzamiento fue la versión 1.0, que apareció en marzo de 2004 y La versión actual es 4.1.1. Un dato que me ha sorprendido es que también hay una versión para la plataforma .NET, Spring .NET .

Como no, al igual que Struts, Spring también está basado en el patrón Modelo-Vista-Controlador y actualmente se ha convertido en uno de los Frameworks Java más utilizados.

Spring, como veremos más adelante, permite manejar patrones de diseño de una manera sencilla. Es muy modular, por lo que se pueden usar algunos de sus módulos de forma independiente. Una de sus principales características es que se integra fácilmente con otros frameworks, como JavaServer Faces o el propio Struts.

Spring proporciona un potente mecanismo de gestión de la configuración basada en JavaBeans, aplicando los principios de Inversión de Control (IoC), lo que hace que la configuración de aplicaciones sea rápida y sencilla. El contenedor de inversión de control de Spring instancia los Beans del sistema y les asigna sus dependencias. Para

ello, necesita que mediante información de configuración se le indique dónde se encuentran dichos Beans.

5.4.2. Características del Framework Spring

Analizando el Framework de trabajo Spring me ha llamado la atención dos de sus características básicas que creo que lo hacen diferente al resto de FrameWorks. Estos son los conceptos de inyección de dependencias y la programación orientada a aspectos.

- Inyección de dependencias o Inversión de Control:

Cuando diseñas una aplicación en Java dispones de muchos objetos que se relacionan entre sí mediante composición. Para enlazar dos objetos tendrías que inyectarle a uno de ellos una instancia del otro. Esto lo realiza Spring por ti, por eso se llama Inversión de control, porque es spring quien se encarga de estas dependencias, instancia los objetos y los inyecta por reflexión. El objetivo es lograr un bajo acoplamiento entre los objetos de nuestra aplicación.

- Programación orientada a aspectos(AOP pos su sigla en inglés):

Este es un nuevo concepto de programación relativamente reciente cuya intención es permitir que las aplicaciones sean modulares y separar así los diferentes conceptos de una aplicación. Gracias a la AOP se pueden dividir los diferentes conceptos que componen una aplicación en entidades bien definidas, de manera apropiada en cada uno de los casos y eliminando las dependencias entre cada uno de los módulos. De esta forma se consigue razonar mejor sobre los conceptos, se elimina la dispersión del código y las implementaciones resultan más comprensibles, adaptables y reusables.

Con todo ello y aplicando la programación Orientada a aspectos, spring consigue:

- El código es más sencillo, más natural y más reducido.
- Dar facilidad para entender los conceptos, ya que están separados y las dependencias entre ellos son mínimas.
- Código más fácil de depurar y más fácil de mantener.
- Código más reusable y independiente entre sí. De esta manera se puede acoplar y desacoplar cuando sea necesario.

5.4.3. Módulos de Spring

Además de las características ya comentadas, lo siguiente que debemos saber es que Spring está formado por una serie de módulos, ya que es bastante grande. No siempre

se utiliza en un proyecto todo lo que tiene Spring, como ya hemos comentando es bastante moldeable al tipo de aplicación o negocio que debamos utilizar. Por poner un ejemplo, podríamos utilizar Struts para la parte web, en vez de Spring MVC. En la siguiente imagen veremos el conjunto de módulos que forman Spring:

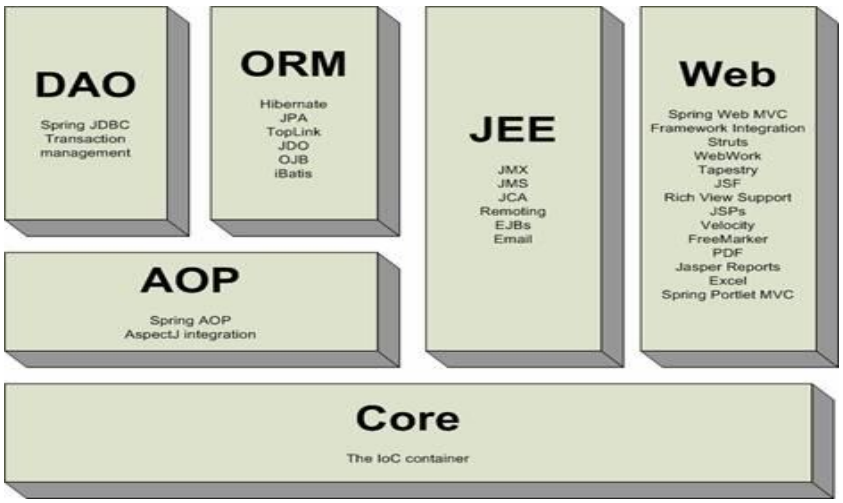


ILUSTRACIÓN 25 SPRING MÓDULOS

CORE: Es la parte fundamental de Spring, el cual se encarga de la Inyección de dependencias. El concepto de este módulo es el BeanFactory, que implementa el patrón de diseño Factory.

DAO: Provee una capa de abstracción sobre JDBC para poder conectar fácilmente con la bbdd, abstrae el código de acceso a datos de una manera simple. Cuenta con una capa de excepciones para manejarlas con sencillez.

AOP: En simples palabras esta capa desacopla el código de una manera limpia, implementando funcionalidad que por lógica y claridad debería estar separada.

ORM: Provee capas de integración para APIs de mapeo objeto - relacional, incluyendo, JDO, Hibernate e iBatis. Usando el paquete ORM es posible usar esos mapeadores en conjunto con otras características que Spring ofrece, como la administración de transacciones mencionada con anterioridad.

WEB: Provee características básicas de integración orientadas a la web, como funcionalidad multipartes (para realizar la carga de archivos), inicialización de contextos mediante servlet listeners y un contexto de aplicación orientado a web. Cuando se usa Spring junto con WebWork o Struts, este es el paquete que te permite una integración sencilla.

J2EE: Provee integración con aplicaciones Java Enterprise Edition así como servicios JMX, JMS, EJB, etc.

5.4.4. Ventajas de Spring frente a otros FrameWorks

- Spring MVC ofrece una división limpia entre Controllers, Models (JavaBeans) y Views.
- Spring MVC es muy flexible, ya que implementa toda su estructura mediante Interfaces, no como Struts que obliga a heredar de clases concretas tanto en sus Actions como en sus Forms.
- Spring MVC provee interceptores y controllers que permiten interpretar y adaptar el comportamiento común en el manejo de múltiples request.
- Los controllers de Spring MVC se configuran mediante IoC como los demás objetos, lo cual los hace fácilmente testeables e integrables con otros objetos que estén en el contexto de Spring, y por tanto sean manejables por éste.
- Las partes de Spring MVC son más fácilmente testeables que las de Struts, debido a que evita la herencia de una clase de manera forzosa y una dependencia directa en el controller del servlet que despacha las peticiones.
- No existen ActionForms, se enlaza directamente con los beans de negocio.
- Struts obliga a extender la clase Action, mientras que Spring MVC no, aunque proporciona una serie de implementaciones de Controllers para que el usuario los utilice. Existen una gran variedad de Controladores.
- Spring tiene una interfaz bien definida para la capa de negocio.

5.4.5. Ciclo de vida en Spring

Uno de los puntos a tener en cuenta en la utilización de Spring es el ciclo de vida de las entidades que lo forman. En este caso vamos a ver cómo funciona el ciclo de vida de los beans, que son utilizados en Spring.

En una aplicación Java normal, los beans son inicializados normalmente (con "new") y una vez finalizado su cometido es el recolector de basura de Java el que los destruye. En una aplicación Spring, los beans vivirán dentro de un contenedor gestionado por Spring. Esto quiere decir que Spring gestiona su ciclo de vida, desde su creación hasta su destrucción.

La siguiente imagen ilustra de forma simple esta característica:

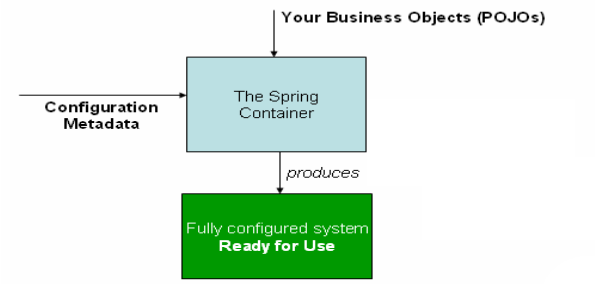


ILUSTRACIÓN 26 SPRING CICLO DE VIDA

Así el ciclo de vida de un bean es algo más complejo que su creación y destrucción. Podemos ver esta imagen las diferentes etapas de un bean:

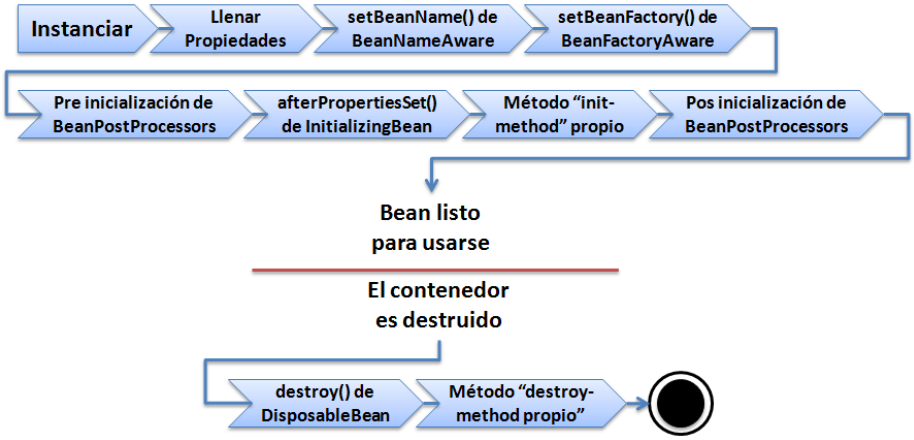


ILUSTRACIÓN 27 SPRING ESQUEMA BEAN

En la creación de bean, en Spring vemos las siguientes etapas:

- Spring instancia el bean (básicamente un "new" por defecto).
- Spring inyecta las propiedades según se hayan definido en los ficheros de configuración.
- Spring establece el nombre del bean (atributo id) y de la factoría.
- Si hay algún BeanPostProccessor, Spring llamará al método postProcessBeforeInitializacion().
- Spring llama al método afterPropertiesSet() provocando que si el bean tiene declarado un método de inicialización personalizado se ejecute este método.
- Si hay algún BeanPostProcessor, Spring llama al método postProcessAfterInitializacion().

En la destrucción, Spring realiza lo siguiente:

- Invoca al método destroy. Si tiene definido un método destroy personalizado, se invoca a este método.

6. Análisis y Diseño del FrameWork para la capa de presentación (SignArtFw)

Después de analizar los diferentes Frameworks de trabajo vamos a diseñar un Framework de presentación particular para la implementación y desarrollo de aplicaciones. En concreto llamaremos a nuestro Framework de trabajo SignArtFw en honor a la empresa para la cual diseñaremos la posterior aplicación de test.

ILUSTRACIÓN 28 LOGO SIGNARTFW

6.1. Características generales del SignArtFw

El framework SignArtFw tiene las siguientes características propias:

- Internacionalización: Las aplicaciones web realizadas con SignartFw tendrán las propiedades en diferentes archivos de propiedades. Lo que hace más sencillo que la aplicación tenga varios idiomas.

- Validación de campos: En el Framework de Signart se validarán los campos necesarios de formulario a través de librerías JQuery. Ello hace que realizar controles en la vista sea sencillo y rápido.

- Estilo propio: Se ha implementado una hoja de estilos propia para las aplicaciones de SingartFw.

- Configuración de la aplicación realizada a través de signartFw-servlet. Toda la configuración de Signart referente a estructura de jsp, internacionalización, conexión con la base de datos y persistencia de la base de datos la encontraremos en este fichero.

- Configuración de las dependencias de clases externas del proyecto a través de Maven y el fichero de configuración pom.xml.

- Acceso a la base de datos a través de Hibernate. En SignartFw hemos realizado la configuración a nuestra propia base de datos de Signart de una manera sencilla. Utilizando tecnología de mapeo Hibernate con notaciones típicas de Spring.

- Hemos creado una cabecera propia para las aplicaciones signartFw. En esta nos aparecerán los logos de UOC y Signart respectivamente y los botones implementados.

6.2. Arquitectura del Framework

SignArtFw tendrá una arquitectura por capas: vista, control y negocio y acceso a datos. Tendrá las siguientes características:

- Accederemos a la base de datos a través de una capa (DAO). En esta capa utilizaremos tecnología Hibernate para que el acceso y su implementación sea lo más sencillo e intuitivo.

- Llamaremos a la capa DAO mediante una capa de servicios. Por ello vamos a tener una interfaz llamada SignArtService.

- La capa de Controlador implementa el patrón Front Crontoller similar a Spring o Struts, la llamaremos desde SignArtController. Des de aquí accederemos a los Servicios correspondientes.

- Finalmente para la capa de presentación utilizaremos JSP basándonos en el patrón Composite View. Para realizar la validación en formularios utilizaremos el patrón Intercepting Filter mediante librerias JQuery. También añadiremos taglibs, que podríamos clasificarlos dentro del patrón ViewHelper.

En la siguiente figura podemos observar cómo sería la arquitectura de SignArtFw. A través de esta arquitectura realizaremos la aplicación de test.

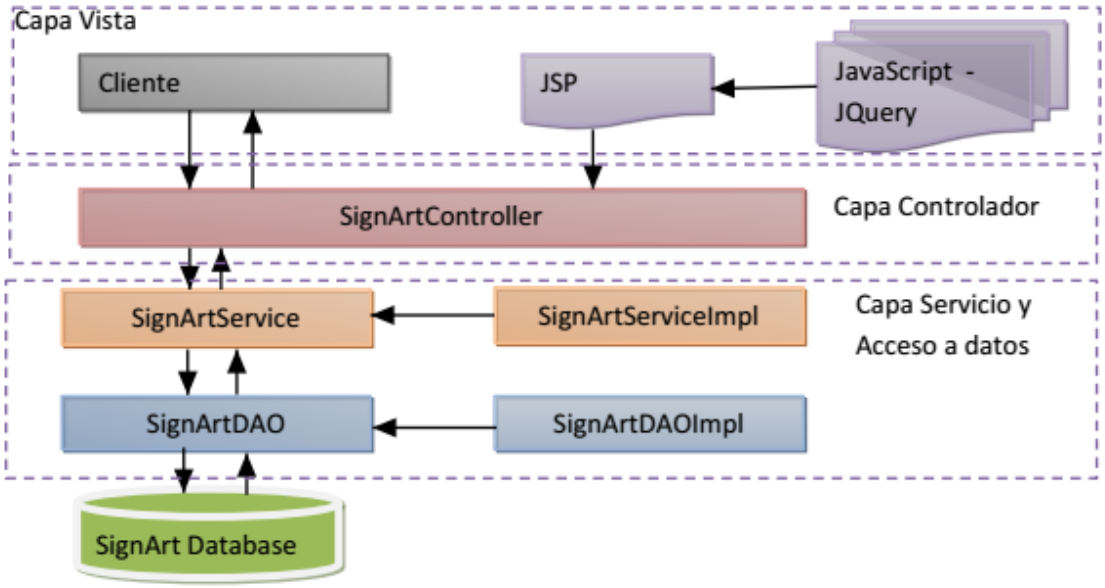


ILUSTRACIÓN 29 SIGNARTFW ARQUITECTURA

6.3. Entorno tecnológico de SigArtFw

En este punto vamos a realizar una pequeña introducción a las tecnologías utilizadas en SignartFw y que no se han comentado anteriormente.

6.3.1. Hibernate

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

En SignartFw utilizamos Hibernate para relacionarnos con la base de datos MySql y anotaciones en los objetos que creemos y que queremos que interactúen con la base de datos.

Mediante el uso de Hibernate no necesitamos tener muchos conocimientos de código sql para interactuar con la base de datos.

6.3.2. Java Virtual Machine

Para desarrollar cualquier aplicación en Java J2EE en necesario que tengamos un Maquina Virtual Java instalada en nuestra máquina. Para SignartFw con tener una máquina Java Versión 1.6 será suficiente.

6.3.3. Apache Tomcat

Es un contenedor de servlets o servidor de aplicaciones que sirve para desarrollar e desplegar aplicaciones web implementadas en Java. Este está escrito en Java y es de licencia gratuita con lo que es perfecto para familiarizarnos con las aplicaciones j2EE.

Tomcat puede funcionar como servidor web por sí mismo. En sus inicios existió la percepción de que el uso de Tomcat era sólo recomendable para entornos de desarrollo y entornos con requisitos mínimos de velocidad. Hoy en día ya no existe esa percepción y Tomcat es usado como servidor web autónomo en entornos con alto nivel de tráfico y alta disponibilidad.

6.3.4. Maven

Apache Maven es una herramienta capaz de gestionar del ciclo de vida de un proyecto de Software (*Project Management Tool*). Es usada tanto por desarrolladores como por ingenieros de pruebas, arquitectos software y jefes de proyecto.

Para nuestro Framework SignartFw utilizamos el archivo de Maven pom.xml para descargar automáticamente todas la librerías externas que necesitamos para implementar la aplicación.

6.3.5. MySql

MySQL es un sistema de gestión de base de datos relacional, multiusuario muy extendido en la actualidad. MySQL AB, desde enero de 2008 una subsidiaria de Sun Microsystems y ésta a su vez de Oracle Corporation desde abril de 2009, desarrolla MySQL como software libre en un esquema de licenciamiento dual.

Hemos escogido para SignartFw este "motor" de base de datos por los siguientes motivos:

- Código abierto, sin ningún tipo de licencia, por lo tanto gratuito.
- Estable y con buen rendimiento.
- Fácil de instalar y conectar con J2EE.

6.3.6. JQuery

Framework java script para implementar los aspectos relacionados con el comportamiento rico y peticiones AJAX des del navegador. Permite simplificar la manera de interactuar con los documentos HTML, desarrollar animaciones y agregar interacción con la técnica AJAX a las páginas web.

JQuery es un software libre y de código abierto y lo hemos utilizado en SignartFw para validar los formularios.

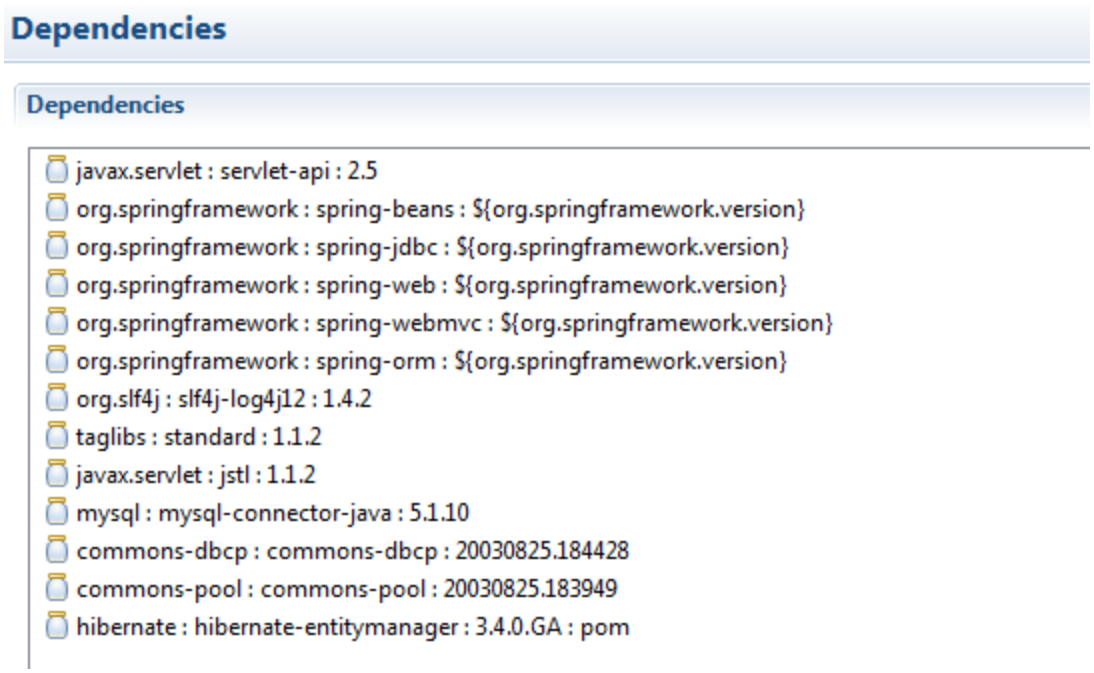
6.3.7. Eclipse o Spring Tool Suite

Eclipse es una aplicación de trabajo que nos facilita la generación de código para construir herramientas y aplicaciones. Eclipse comprende una estructura que puede soportar distintas herramientas de desarrollo y diferentes lenguajes. Tiene una muy buena colección de plugins que hace que sea una herramienta bastante personalizable y que se pueda utilizar para múltiples tecnologías. Desde .Net hasta Android.

Eclipse se ha popularizado en la comunica de desarrolladores J2EE para trabajar como herramienta de aplicaciones Web y han aparecido alguna personalización concreta según el Framework de trabajo que utilicemos. En nuestro caso utilizamos para desarrollar SignartFw Spring Tool Suite, que no es más que una versión de Eclipse "tuneada" o modelada para trabajar con Spring o derivados.

6.4. Dependencias

Para el desarrollo del Framework SignartFw, hemos necesitado librerías externas Java para poder trabajar. Ellas son necesarias para poder conectar con la base de datos, poder trabajar con Hibertate, utilizar los beans, servelts,…



6.5. Diseño externo de la capa de presentación

6.5.1. Diseño de las vistas (Composite View)

Para realizar la capa de presentación de SignartFw hemos tenido en cuenta que debido a su lógica de negocio y con el fin de reducir al máximo el código a escribir, necesitaremos diseñar dentro de la misma página parte dinámicas y partes estáticas. Todas ellas con tecnología JSP (Java server Pages).

Podríamos dividir la página en tres partes Header, Body y Footer. Tanto el Header como el Footer se mantendrán estáticos, mientras que será el body, el que se modifique. Esta implementación fragmentada de la capa de la vista se corresponde con el patrón de diseño Composite view. Con ello tenemos el siguiente diseño:

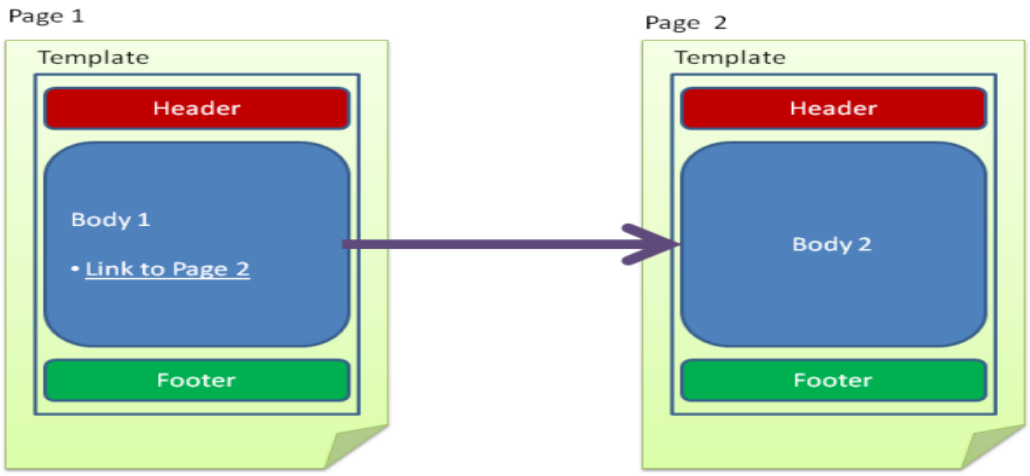


ILUSTRACIÓN 30 SIGNARTFW DISEÑO VISTAS

Para ello crearemos dos partes siempre estáticas que serán: header.jsp, footer.jps mientras la parte de body siempre ira modificándose.

La manera de incluir el header y el footer será utilizando la etiqueta jsp include.

```
<body>
	<!-- Contenedor -->
	<div id="contenedor">
		<!-- Menú de Header -->
		<header>
		    <jsp:include page="header.jsp"/>
		</header>
		<!-- FMenú de Header -->

		<!-- Contenido -->
		<div id="contenido">
			<h3><spring:message code="texto_indice"/></h3>
		</div>
		<!-- F Contenido -->

		<!-- Footer -->
		<footer>
			<jsp:include page="footer.jsp"/>
		</footer>
		<!-- FFooter -->
	</div>
	<!-- F Contenedor -->
</body>
```

En la siguiente imagen podemos ver como quedaría el diseño de la página marcando el header, body y footer:

ILUSTRACIÓN 31 SIGNARTFW COMPOSITE VIEW

6.5.2. Estilo de las páginas

Para Signart Fw hemos definido una hoja de estilo concreta que hemos llamado Signart.css. En ella se define el contenido de las capas flotantes, tablas, estilos de letra, …

Incluiremos la hoja de estilo en la página de la siguiente manera:

```
<link rel="stylesheet" href="<c:url value="/resources/static/css/signart.css" />" type="text/css"
media="screen" />
```

ILUSTRACIÓN 32SIGNARTFW ESTILOS

6.6. Diseño interno

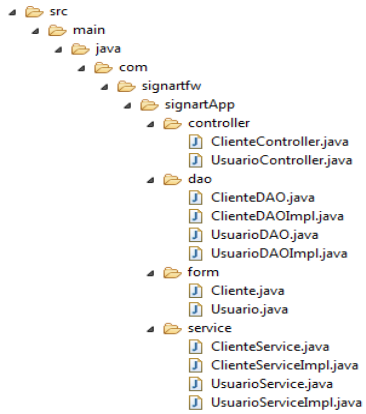
En este punto pasaremos a explicar con más detalle el diseño interno de SignartFw y aquellos patrones J2EE que utiliza.

6.6.1. Estructura de clases de SignartFw

Como hemos comentado antes, SignartFw sigue la estructura del patrón MVC. Así las diferentes clases que compongan las aplicaciones con este Framework se deberán estructurar de la siguiente manera

- Controlador: Son todas aquellas clases Java que dirigen el tráfico de la aplicación. Se encargan de recibir la petición del usuario y redireccionarlo al servicio correspondiente.
- Servicio: En este directorio se situará la lógica de negocio de las aplicaciones. Aquí es donde se deberían realizar las modificaciones a realizar antes de acceder a la base de datos.
- Dao: En este directorio se situarán aquellas clases que accedan o modifiquen directamente la bbdd.

Podemos ver dicha estructura en las carpetas realizadas en la aplicación de test.



Si seguimos analizando el resto de clases necesarios para el Framework podemos ver en el siguiente esquema como se distribuirían las aquellas librerías externas que necesitamos para que funcione correctamente el Framework:

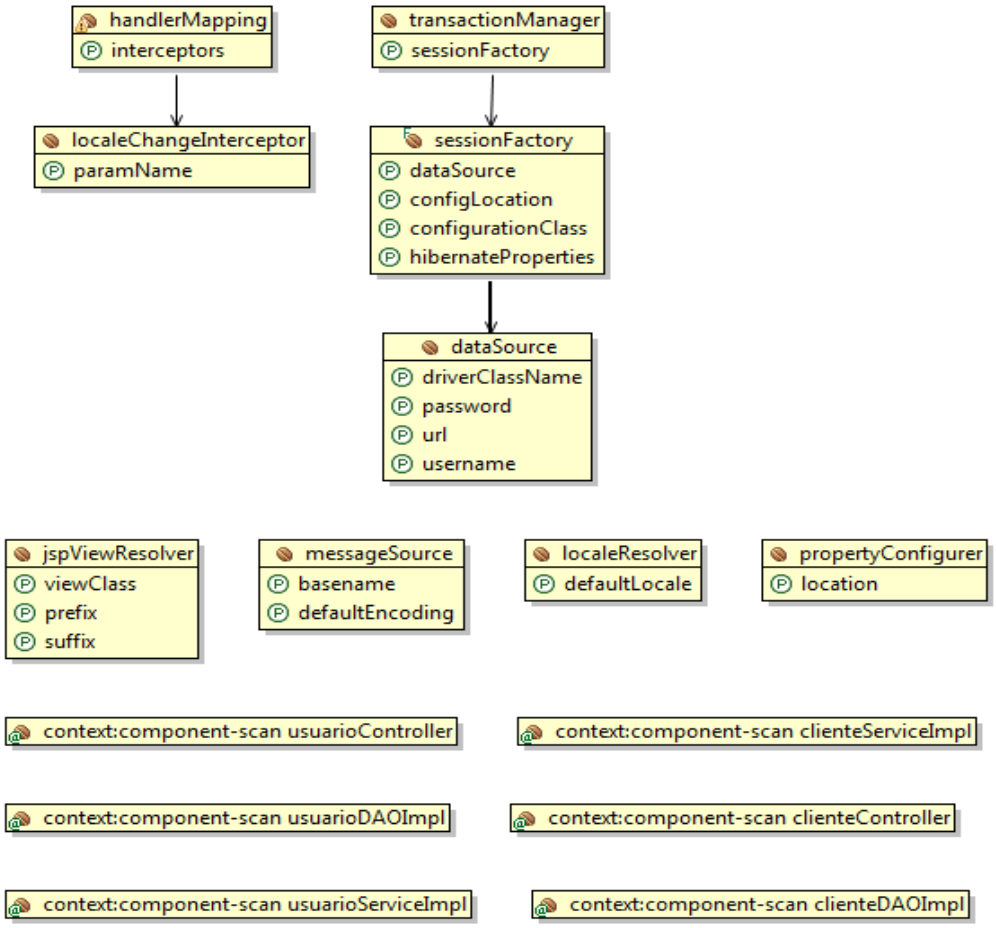


ILUSTRACIÓN 33 SIGNARTFW LIBRERÍAS

6.6.2. Archivos de configuración

Los principales archivos para la correcta configuración de SignartFw son xml. Son los siguientes:

Pom.xml

Maven utiliza este fichero también llamado Project Object Model (POM) para describir el tipo de software a construir, sus dependencias con otras clases y componentes externos, y el orden de construcción de los elementos. Viene con objetivos predefinidos para realizar ciertas tareas claramente definidas, como la compilación del código y su empaquetado. En el caso del Framework SignartFw, el archivo pom.xml nos permitirá descargar todas las clases que necesitamos para trabajar con el Framework sin tener que añadir nuevas librerías.

web.xml

Web.xml es el fichero de configuración que se encarga de recibir todas las peticiones y redirigirlas a los controladores. El apartado <servlet-mapping> indica qué peticiones

del navegador se deben redirigir a qué servlet, en concreto, para signartFw, cualquier petición de cualquier fichero (<url-pattern>) se redirigirá al servlet signartFw.

Por otro lado en este fichero también nos mostrará cual es el servlet central (DispatcherSevlet) que se encarga de realizar la redirección de los request. DispatcherServlet implementa la funcionalidad descrita en el patrón de diseño de FrontController. Este consiste en un único punto de entrada que redirige todas las petiones a los controladores indicados.

También configuraremos en este punto cual será la primera página de la aplicación. En este caso la llamaremos será el index.html

signartFw-servlet.xml

SigartFw nos indicará donde están situados los beans de la aplicación. Si nos centramos en la aplicación de test SignartApp nos encontraremos que:

- jspViewResolver nos indicará el directorio de las vistas o jsp
- messaResource nos indicará el directorio de las properties
- propertyconfigurer: Nos indicara propiedades para pode acceder a la base de datos.
- dataSoruce: Indicaremos datos para conectarnos a la base de datos.
- sessionFactory nos indicara que vamos a utilizar Hibernate como capa de persistencia y donde está dicho archivo de Mapeo a la bbdd.

Hibernate.cgt.xml

Este fichero de configuración de Hibernate nos servirá para marcar el mapeo de entidades que tenemos en la aplicación. En nuestro caso para SigartApp únicamente tenemos dos: Cliente y Usuario. Si hubiera alguna más deberíamos indicarlo aquí.

6.6.3. Diseño de filtros (Intercepting Filter winth JQuery)

Como hemos comentado anteriormente, para validar los formularios existentes en SignartFw utilizaremos librerías JQuery. Añadiremos así la librería jquery.js al framework y de esta manera será mucho más sencillo marcar que campos son obligatorios en un formulario y que mensaje queremos que aparezca en el caso que estos campos no se completen. Además gracias a JQuery también podemos también marcar otras características a los campo:, la máxima longitud de campo, que se un correo electrónico valido, etc.

De esta manera podemos decir que estamos utilizando el patrón Intercepting Filter, siempre teniendo en cuenta que Jquery no es Java. Así las clases Jquery incluidas en el Framework representan un filtro que está entre el cliente y el resto de la aplicación web.

Como ejemplo, mostraremos el código de cómo podemos marcar que campos son obligatorios y sus respectivos mensajes que queramos que aparezcan por pantalla.

```
<script type="text/javascript">
                $(document).ready(function() {
                        $("#cliente").validate({
                rules: {
                        nombre: { required: true, minlength: 2},
                        alias: { required: true, minlength: 2},
                        descripcion: { required:true, minlength: 2},
                        email: { required:false, email: true},
                        telefono1: { required: true, minlength: 2}
                },
                messages: {
                        nombre: "<spring:message code="cli.validate.nombre"/>",
                        alias: "<spring:message code="cli.validate.alias"/>",
                        descripcion: "<spring:message code="cli.validate.descripcion"/>",
                        email: "<spring:message code="cli.validate.email"/>",
                        telefono1:"<spring:message code="cli.validate.telefono1"/>"
                }
                    });
                });
</script>
```

ILUSTRACIÓN 34 SIGNARTFW FILTROS

6.6.4. Diseño del controlador (Front Controller)

Los controladores en SignartFw son los directores de orquesta del patrón de presentación. Así en nuestro caso nos indicarán que acción esperan recibir a través de la notación @requestMapping, para después realizara las acciones oportunas con la capa de negocio y finalmente realizar el return hacia la vista que le corresponda. En el

siguiente esquema podemos ver como el controlador recibe la petición del Front Controller, crea el modelo necesario y marca que modelo o vista debe retornar.

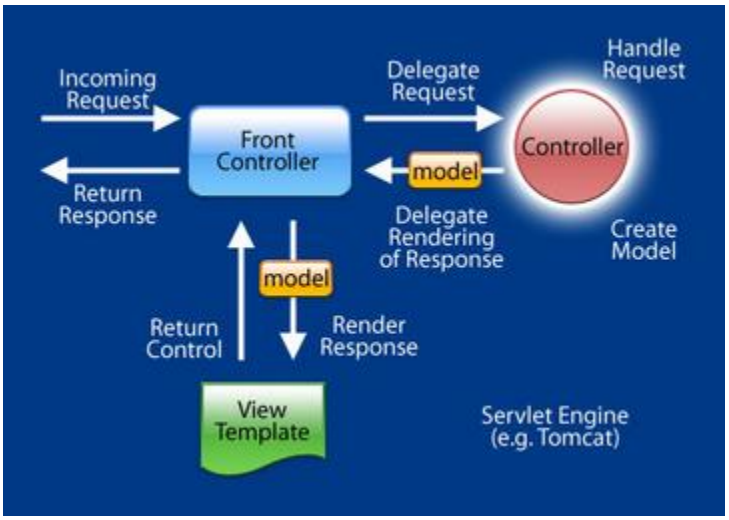


ILUSTRACIÓN 35 SIGNARTFW CONTROLADOR

6.6.5. Diseño de taglibs a través de View helper

Me gustaría remarcar la manera como se presenta la información solicitada a través del jsp. Esta no se hace de otra manera que ayudados por el patrón ViewHelper. Para la creación de las vista existe una clase que mediante diferentes métodos ayuda a las JSP a crear la vista final que el cliente visualiza.

En nuestro caso hemos elegido tags lib propios de sun y de spring para realizar lógica sobre las vista en el momento de la visualización.

```
<%@taglib uri="http://www.springframework.org/tags" prefix="spring"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
```

Por ejemplo, en el momento de saber si un listado de usuarios está lleno o vacio realizamos un función escita por sun isEmpty y para recuperar el mensaje explicativo utilizamos el tag message:

```
<c:if  test="${empty listaUsuarios}">
<td colspan="7"><spring:message code="usu.sin_usuarios"/></td>
</c:if>
```

6.7. Diccionario de clases y métodos

Para ver el todas las clases, funciones y métodos implementados en SignartFw y su aplicación de test, adjuntamos conjuntamente con el proyecto el javadoc correspondiente: singartApp-javadoc.rar

7. Aplicación test SignartApp

Una vez comentado SignartFw vamos a crear una aplicación mediante la cual demostremos que siguiendo la estructura comentada anteriormente podemos crear una aplicación que se adapte a las necesidades de nuestra pequeña empresa de rotulación Signart. Por ello hemos bautizado a dicha aplicación web SignartApp.

7.1. Funcionalidades implementadas en SignartApp

Dado que la primera carencia que hemos encontrado en dicha empresa en la falta de ninguna aplicación donde se gestionen los usuarios o los clientes, desarrollaremos un primer embrión de la aplicación de gestión dónde los dos primeros módulos serán precisamente estos:

- Módulo de gestión usuarios.
- Módulo de gestión de clientes.

7.1.1. Pantalla inicial de bienvenida

Al acceder a la página inicial de SignartApp nos encontraremos con una pantalla de presentación o índice de la aplicación.

ILUSTRACIÓN 36 SIGNARTAPP PANTALLA BIENVENIDA

En el header de la aplicación nos encontraremos con 2 links destacados con sus respectivos enlaces. Estos son los correspondientes con la web de UOC y de Signart.

Seguidamente podremos ver la botonera inicial, que nos llevará a la pantalla de Inicio, a la gestión de usuarios y a la gestión de clientes.

En el footer encontraremos la posibilidad de cambiar el idioma de la aplicación.

Cabe destacar como hemos comentado en el punto Diseño de vistas, que la división de la página inicial en tres secciones: header, body y footer se corresponde con el patrón de diseño Composite view. Recordamos el diagrama de secuencias de dicho patrón:

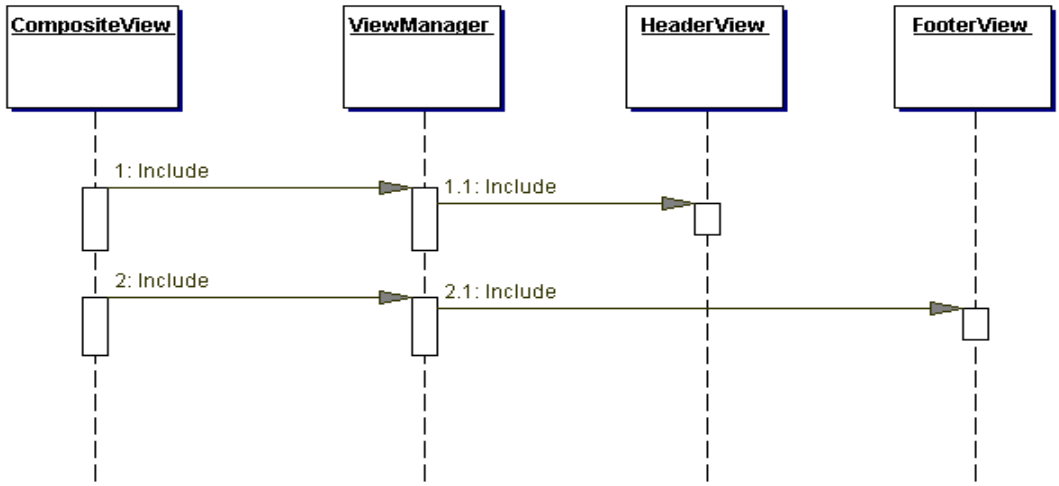


ILUSTRACIÓN 37 DIAG. SECUENCIA COMPOSITE VIEW

Gracias a la utilización de dicho patrón obtenemos una mejoría notable en el rendimiento de la aplicación y reutilizamos código.

Recordar también, que como hemos comentado el Framework SignartFw soporta la internacionalización en varios idiomas. Así la aplicación creada tiene dos ficheros de properties. Uno para el castellano y otro para el catalán (messages_es.properties, messages_ca.properties). Podemos considerar que esta característica es parte del patrón de diseño ViewHelper, ya que accedemos des de la capa de vista a una propiedad según el idioma marcado por el usuario.

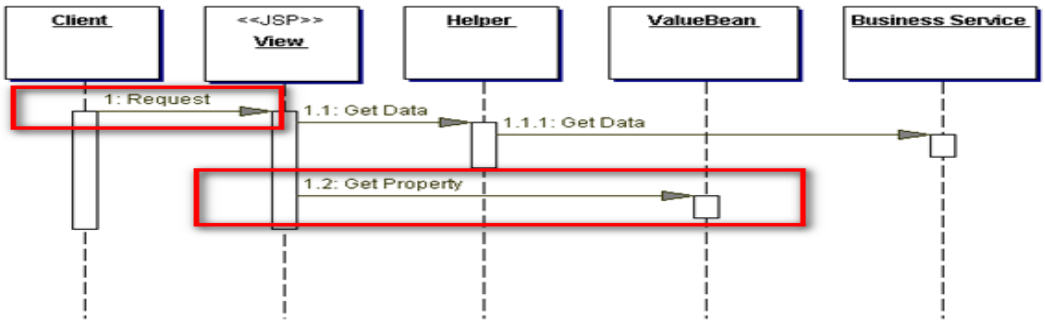


ILUSTRACIÓN 38 DIAG. SECUENCIA VIEW HELPER

En la siguiente imagen mostramos los mensajes en los dos idiomas como hemos comentado. Si fuera necesario para la empresa Signart, realizar la versión en inglés, solo deberíamos añadir un nuevo messages_en.properties.

```
# Properties en castellano

texto_indice=Esta aplicación quiere ser la presentación del FrameWork SignArtfw y un punto de partida para desarrollar aplicaciones en J2EE para SignArt
usu.titulo=Modificar usuario
usu.nombre=Nombre
usu.apellido1=Primer Apellido
usu.apellido2=Segundo Apellido
usu.email=Email
usu.telefono=Telefono
usu.perfil=Perfil
usu.insertar=Añadir usuario
usu.listadoUsuarios=Listado de Usuarios
usu.modificar=Modificar usuario
usu.titulo.insertar.usuarios=Crear nuevo usuario
usu.sin_usuarios=Por el momento, no hay usuarios creados
usu.validate.nombre= El nombre es un campo obligatorio
usu.validate.apellido= El apellido es un campo obligatorio
usu.validate.email= El email es un campo obligatorio
usu.validate.perfil= El perfil es un número entre 1 y 3

inicio=Inicio
usuarios=Usuarios
clientes=Clientes
eliminar=Eliminar

cli.listadoClientes=Listado de clientes
cli.nombre=Nombre
cli.alias=Alias
cli.descripcion=Descripción
cli.nif=Nif
cli.email=Email
cli.telefono1=Teléfono 1
cli.telefono2=Teléfono 2
cli.direccion=Dirección
cli.poblacion=Población
cli.cp=Código Postal
cli.masDatos=Mas datos
cli.sin_clientes=Por el momento, no hay ningún cliente creado
cli.titulo.insertar.clientes=Añadir nuevo cliente
cli.titulo.insertar.cliente=Añadir un nuevo cliente
cli.fechaCreacion=Fecha de creación
cli.validate.nombre= El nombre es un campo obligatorio
cli.validate.alias= El alias o nombre es un campo obligatorio
cli.validate.descripcion= La descripción del cliente es un campo obligatorio
cli.validate.email= El email es un campo obligatorio
cli.validate.telefono1= El campo teléfono es un campo obligatorio
cli.titulo.insertar.cliente=Insertar nuevo cliente en Signart
cli.nif=Nif
cli.cif=Cif
cli.titulo.modificar.cliente=Modificar cliente de Signart
cli.fechaCreacion=Fecha creación
```

ILUSTRACIÓN 39 SIGNARTAPP MESSAGES

```
# Properties en catalan

texto_indice=Aquesta aplicació vol ser una presentació del FrameWork SignArt i el punt de partida per desenvolupar aplicacions Web J2EE per a SignArt
usu.titulo=Modifica usuari
usu.nombre=Nom
usu.apellido1=Primer Cognom
usu.apellido2=Segon Cognom
usu.email=Email
usu.telefono=Telèfon
usu.perfil=Perfil
usu.insertar=Afegir un usuari
usu.listadoUsuarios=Llistat d'usuaris
usu.modificar=Modificar usuari
usu.titulo.insertar.usuarios=Crear un nou usuari
usu.sin_usuarios=Per el moment, no hi ha usuaris creats.
usu.validate.nombre= El nóm és un camp obligatori
usu.validate.apellido=  El cognom és un camp obligatori
usu.validate.email=  El email és un camp obligatori
usu.validate.perfil= El perfil és un número entre 1 i 3

inicio=Inici
usuarios=Usuaris
clientes=Clients
eliminar=Eliminar

cli.listadoClientes=Llistat de clients
cli.nombre=Nom
cli.alias=Alias
cli.descripcion=Descripció
cli.nif=Nif
cli.email=Email
cli.telefono1=Telèfon 1
cli.telefono2=Telèfon 2
cli.direccion=Direcció
cli.poblacion=Població
cli.cp=Còdi Postal
cli.masDatos=Mes dades
cli.sin_clientes=Per el moment no hi ha cap client creat
cli.titulo.insertar.clientes=Afegir client
cli.titulo.insertar.cliente=Afegir un nou client
cli.fechaCreacion=Data de creació
cli.validate.nombre= El nóm és un camp obligatori
cli.validate.alias= Alias o nóm és un camp obligatori
cli.validate.descripcion= La descripció és un camp obligatori
cli.validate.email= El camp email és un camp obligatori
cli.validate.telefono1= El telèfon és un camp obligatori
cli.titulo.insertar.cliente=Insertar nou client a Signart
cli.nif=Nif
cli.cif=Cif
cli.titulo.modificar.cliente=Modificar nuevo cliente de Signart
cli.fechaCreacion=Data creació
```

7.1.2. Módulo de gestión de usuarios

A través de la botonera y clicando sobre Usuarios accederemos a la gestión de usuarios de Signart. Desde esta pantalla podremos observar los usuarios dados de alta en Signart, modificarlos, darlos de baja o crear nuevos.

Definiremos por lo tanto una entintad usuario con las siguientes características:

*__Usuario__ [identificador interno, nombre y apellidos, perfil (1,2,3), correo electrónico, teléfono]

En este mantenimiento aparecen en la pantalla inicial un listado con todos los usuarios creados. Al acceder por primera vez a la página no encontraremos ningún usuario dado de alta, con lo que aparecerá el mensaje de que no existen dados de alta.

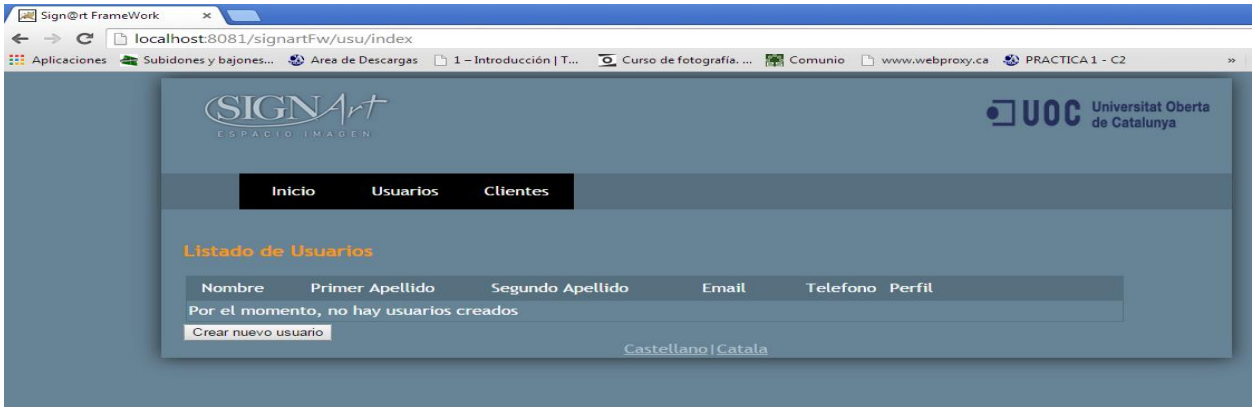


ILUSTRACIÓN 40 SIGARTAPP LISTADO USUARIOS

Como hemos comentado en el diseño interno la gestión de clases se realiza siguiendo el patrón FrontController y es desde los dos Controladores de la aplicación (UsuarioController y ClientreController) des de donde se gestionan las acciones a realizar.

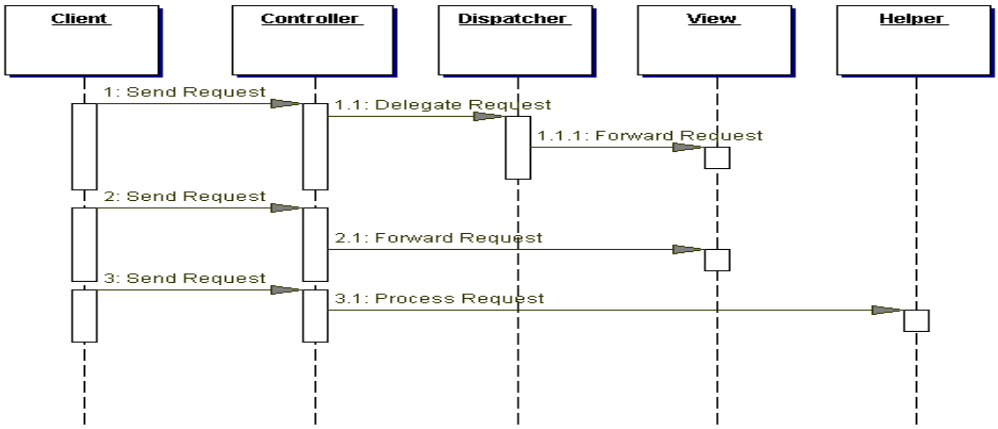


ILUSTRACIÓN 41SIGNARTAPP FRONT CONTROLLER

Ahora nos disponemos a dar de alta un usuario en SignartApp a través del botón de crear usuario. Aparecerá el formulario de creación de usuarios. Este formulario está controlado utilizando Jquery y los campos obligatorios son (Nombre, Apellido, Apellido, Email, Perfil)

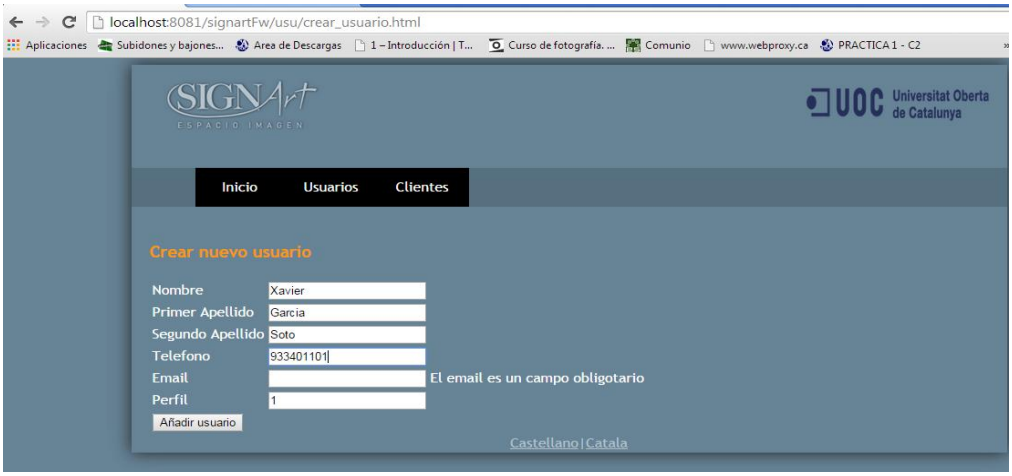


ILUSTRACIÓN 42 SIGNARTAPP MODIFICAR USUARIO

Si cualquiera de estos campos no está informado, las clases Jquery agregadas al Framework Signart nos mostraran un mensaje por pantalla comunicándonos que informemos dicho campo obligatorio. De alguna manera podemos decir que estamos utilizando el patrón Intercepting Filter, siempre teniendo en cuenta que Jquery no es Java. Recordamos el diagrama de secuencia dl Intercepting Filter donde vemos que hasta que todos los filtros obligatorios no están correctamente informados, no permitimos ejecutar la acción. En este caso el alta.

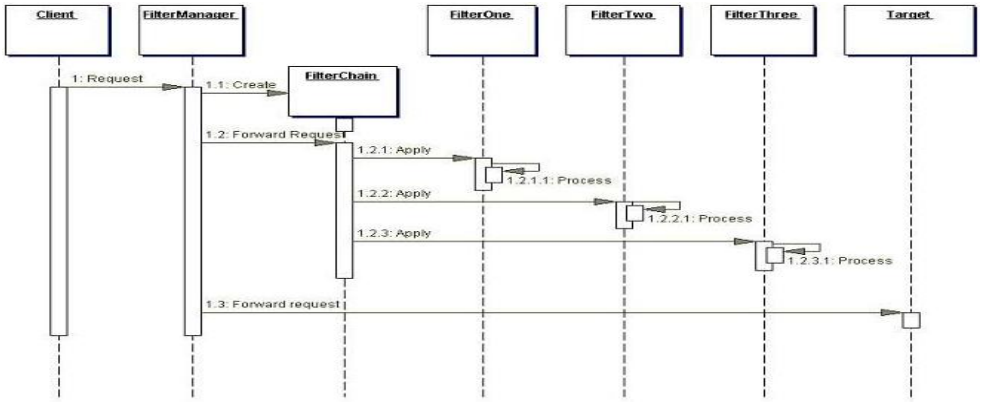


ILUSTRACIÓN 43 SIGANRTAPP INTERCEPTING FILTER

Podemos crear tantos usuarios como queramos, estos se irán añadiendo a la base de datos creada en MySql y serán mostrados en la lista de la aplicación.

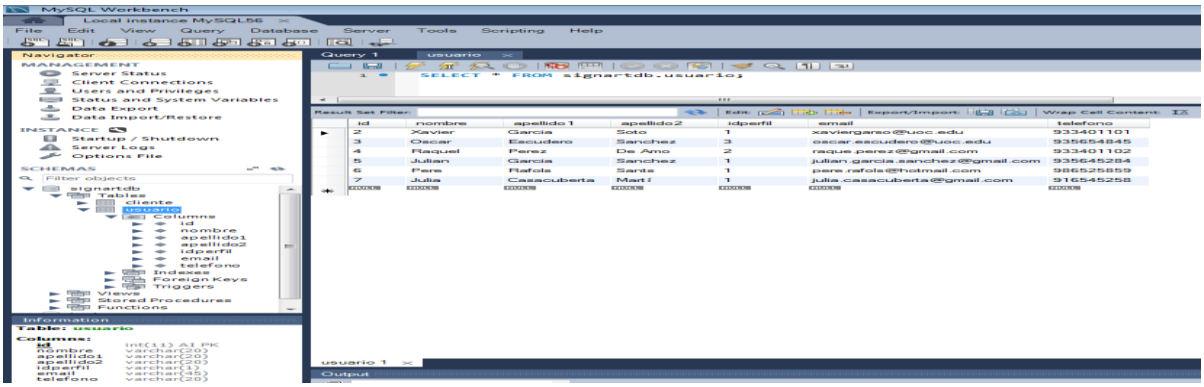


ILUSTRACIÓN 44 SIGNARTAPP TABLA USUARIOS MYSQL

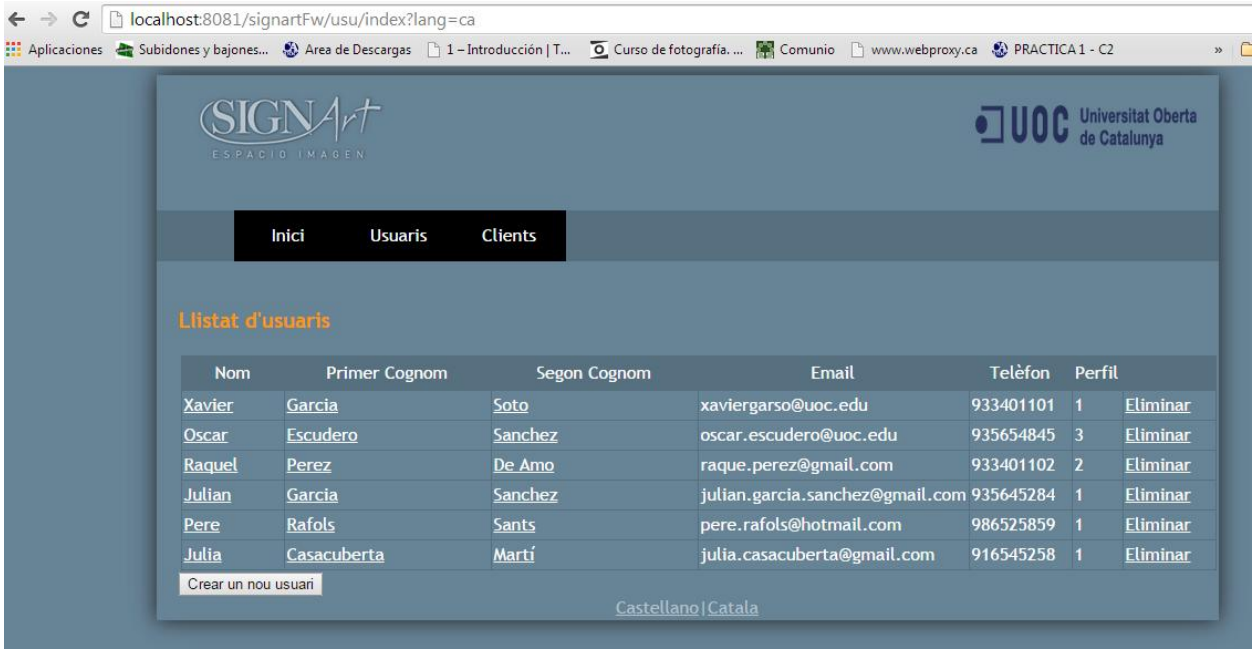


ILUSTRACIÓN 45 SIGARTAPP LISTA USUARIOS

Si clicamos encima del nombre de los usuarios nos aparecerá el formulario desde donde podremos modificar los datos del mismo. Este también tiene las mismas validaciones JQuery que el formulario de alta.

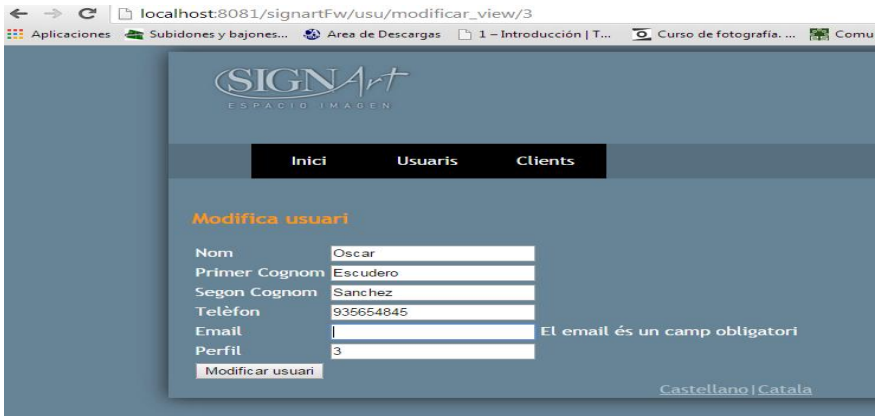


ILUSTRACIÓN 46 SIGNARTAPP MODIFICACIÓN USUARIOS

Si cumplimos con dichas validaciones podremos modificar el usuario y volver al listado donde observaremos que las modificaciones se han realizado correctamente.

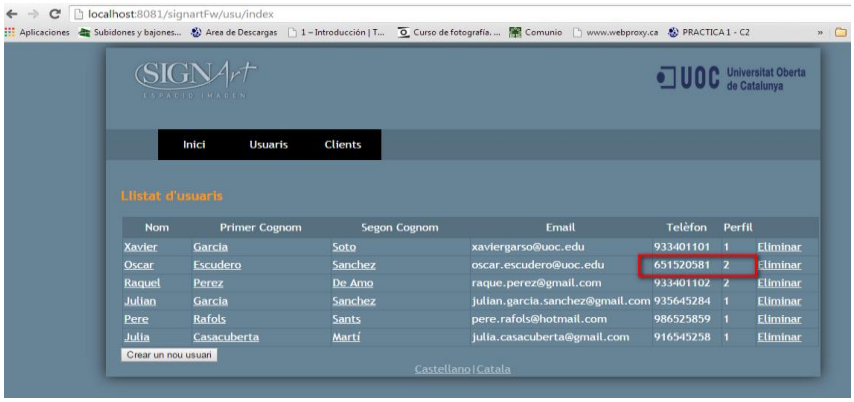


ILUSTRACIÓN 47 SIGARTAPP LISTADO USUARIOS MODIFICADOS

Por último a través del botón eliminar podremos borrar los usuarios de la aplicación Signart. En la siguiente pantalla vemos el resultado de eliminar a Pere y a Julia.

ILUSTRACIÓN 48 SIGNARTAPP LISTADO USUARIOS ELIMINADOS

7.1.3. Módulo de gestión de clientes

Siguiendo el mismo patrón de los usuarios, crearemos un mantenimiento de clientes para Signart.

La definición del cliente en la base de datos será similar a la siguiente:

*__Cliente__ [Identificador, Nombre Completo, Nombre Corto o alias, Descripción Actividad, CIF, NIF, Email, Teléfono 1, Teléfono 2, Dirección, Población, CP, Fecha Creación]

A través del botón de clientes accederemos a la gestión de los clientes. La primera página que nos encontramos es un listado de los mismos. En este caso aparece vacío, ya que aún no tenemos ninguno dado de alta.

ILUSTRACIÓN 49 LISTADO CLIENTES VACIO

A través del botón añadimos un nuevo cliente a la aplicación. El formulario que nos encontramos tiene los siguientes campos obligatorios: nombre, alias y teléfono:

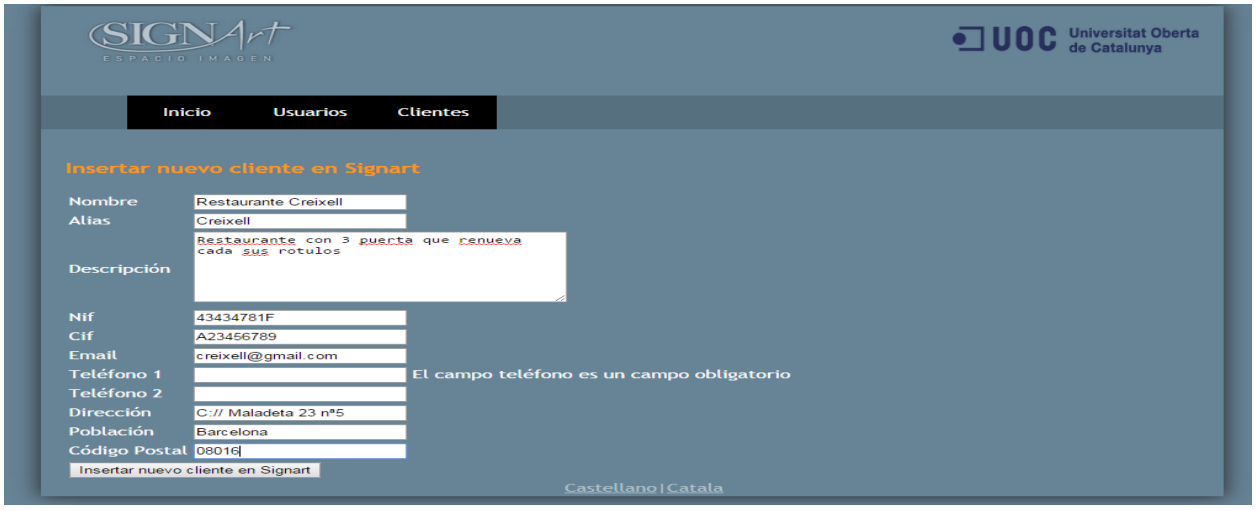


ILUSTRACIÓN 50 SIGNARTAPP CREAR CLIENTE

Respetado los campos obligatorios podemos insertar todos los clientes que necesitemos.

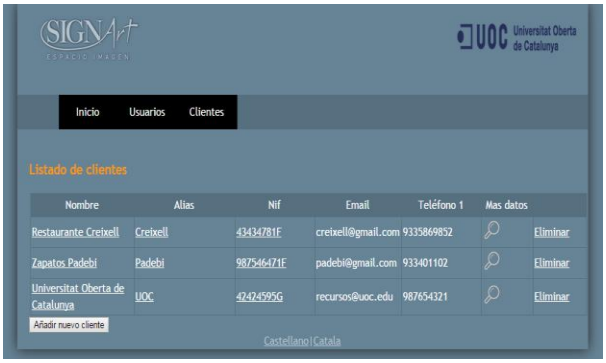


ILUSTRACIÓN 51 SIGNARTAPP LISTADO CLIENTES

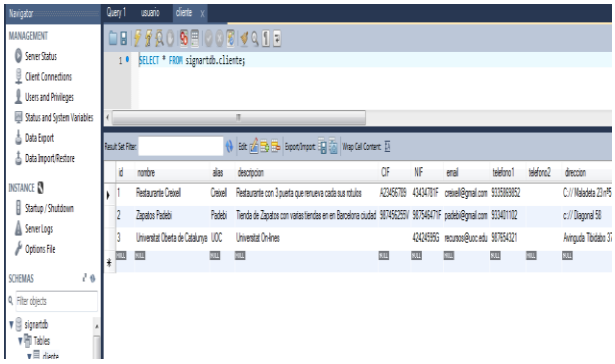


ILUSTRACIÓN 52 SIGARTAPP LISTADO CLIENTES MYSQL

Si clicamos en la lupa o en el nombre, alias o Nif, accederemos al detalle creado, e incluso vemos la fecha en la cual se creó dicho cliente. Des de aquí podemos modificar los datos del cliente.

ILUSTRACIÓN 53 SIGNARTAPP MODIFICAR CLIENTE

Finalmente, también podemos eliminar cliente, a través de la acción eliminar. En la siguiente pantalla podemos ver como se ha eliminado Padebi del listado anterior.

ILUSTRACIÓN 54 SIGNARTAPP LISTADO CLIENTES ELIMINADOS

7.2. Diagrama de clases o UML

La mejor manera de mostrar las clases implementadas y sus relaciones entre ellas es realizando su respectivo diagrama de clases. Tener definido correctamente el diagrama de clases es la base para elaborar la arquitectura MVC. Presentaremos dos diagramas de clases.

7.2.1. Diagrama de clases SignartApp

Podríamos decir que el siguiente es diagrama de clases únicamente de la aplicación SignartApp. En él, solo se muestran las clases Java propias de la aplicación. Como podemos observar las entidades Cliente y usuarios en si no dependen de otras clases. Mientras que el resto: Controller, Services, DAO tienen relación entre ellas.

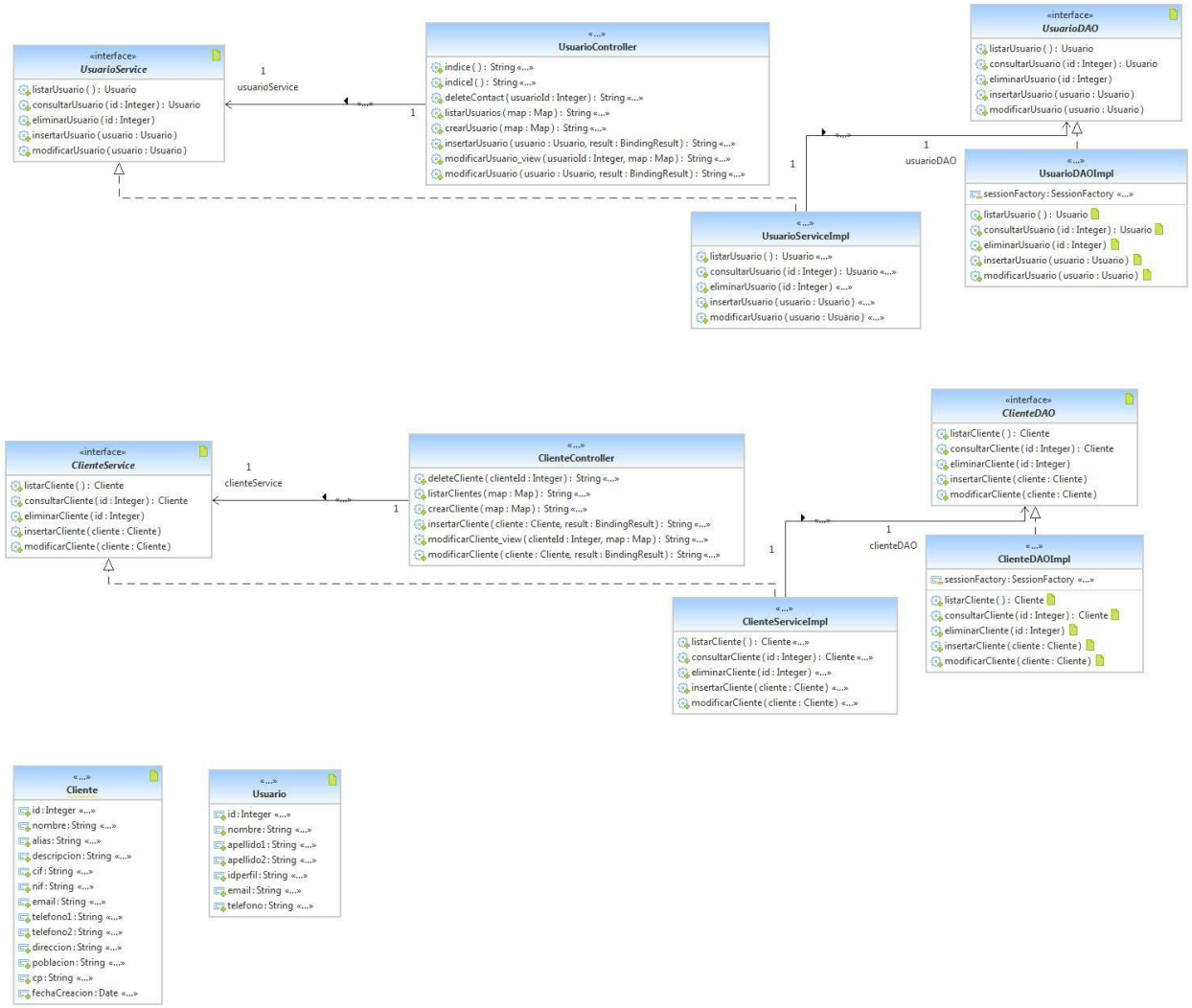


ILUSTRACIÓN 55 SINGARTAPP UML CLASES

7.2.2. Diagrama de clases UML Completo

En el siguiente diagrama de clases queremos mostrar cómo sería el diagrama más completo, incluyendo el framework Signart y sus clases de dependencias.

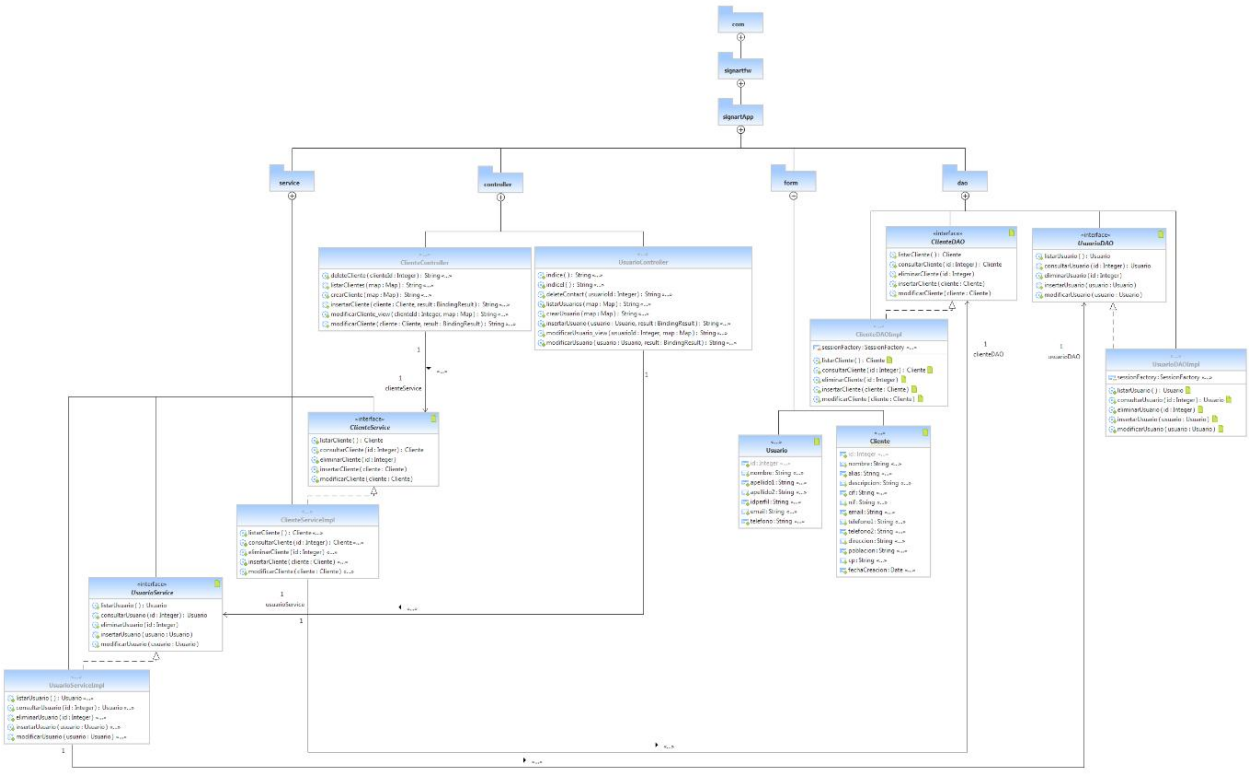


ILUSTRACIÓN 56 SIGANRTFW UML CLASES CON DEPENDENCIAS

7.3. Diagrama caso de usos de SignartApp

El diagrama de casos de uso es una simplificación de la operativa que encontramos en una aplicación. Representa como un cliente opera con todo el sistema. En el caso de SignartApp el diagrama sería el siguiente.

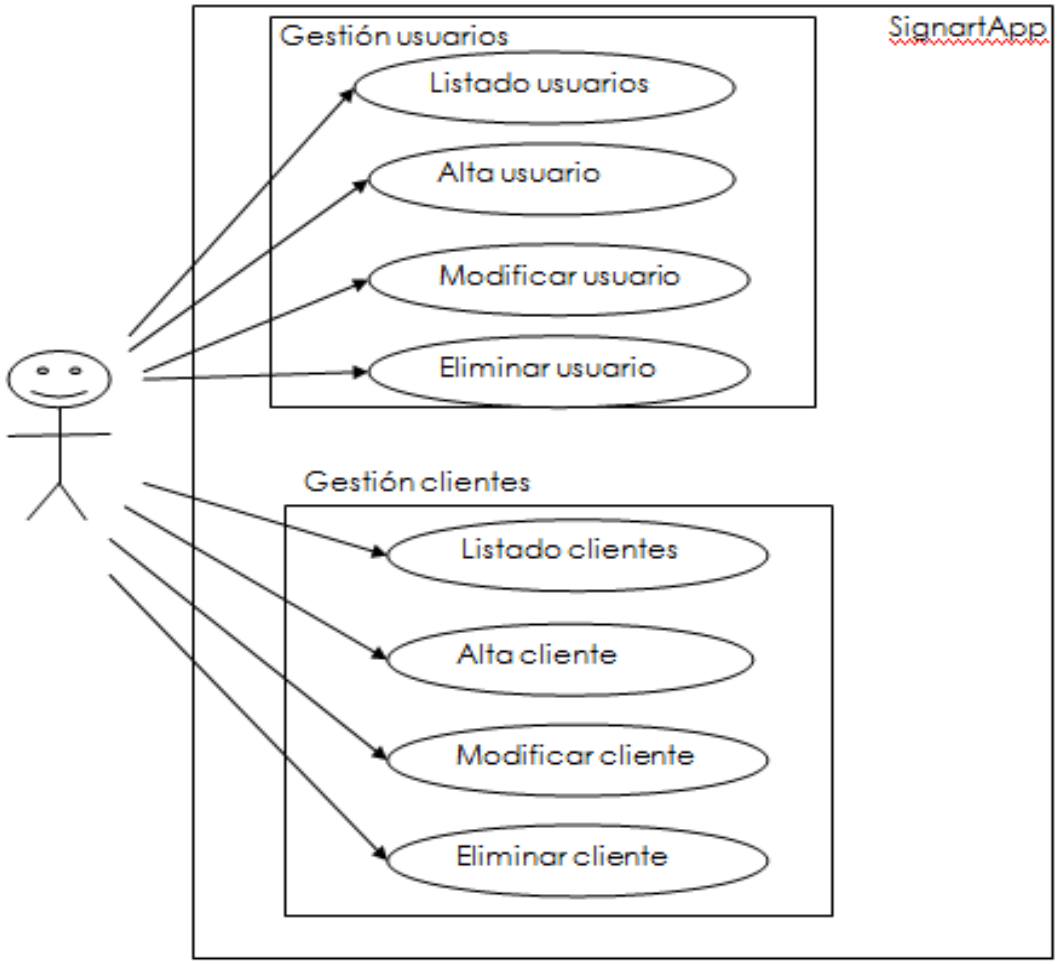


ILUSTRACIÓN 57 SIGNARTAPP CASO DE USO

8. Guía de instalación

Esta es una guía para instalar el SingartFw y la aplicación de test SignartApp.

8.1. Creación de la base de datos para SignartApp

Lo primero de todo será crear una base de datos para la aplicación de test del FrameWork de SignartApp. Para ello hemos utilizado MySQL. En mi caso he utilizado el siguiente software gratuito.

- **MySQL Worbech 6.0**: Framework de base de datos para desarrollo Local
- **Mysql-connector-java-5.1.12**: Conector base de datos MySql

Una vez tenemos el entorno de la base de datos instalada lanzamos el script con las sql que crean la tablas que utilizaremos en SignartApp. Encontrareis el fichero en el directorio: signartApp\src\main\resources

```
-- Crear Esquema:
CREATE SCHEMA `signartdb`;
-- Creación Tabla Usuario:
CREATE TABLE `signartdb`.`usuario` (
  `id` INT NOT NULL,
  `nombre` VARCHAR(20) NULL,
  `apellido1` VARCHAR(20) NULL,
  `apellido2` VARCHAR(20) NULL,
  `idperfil` VARCHAR(1) NULL,
  `email` VARCHAR(45) NULL,
  `telefono` VARCHAR(20) NULL,
  PRIMARY KEY (`id`),
  INDEX `ind_id` (`id` ASC));
-- Creación Tabla Clientes:
CREATE TABLE `signartdb`.`cliente` (
  `id` INT NOT NULL,
  `nombre` VARCHAR(45) NULL,
  `alias` VARCHAR(15) NULL,
  `descripcion` VARCHAR(120) NULL,
  `CIF` VARCHAR(15) NULL,
```

```
  `NIF` VARCHAR(15) NULL,
  `email` VARCHAR(45) NULL,
  `telefono1` VARCHAR(20) NULL,
  `telefono2` VARCHAR(20) NULL,
  `direccion` VARCHAR(45) NULL,
  `poblacion` VARCHAR(15) NULL,
  `cp` VARCHAR(10) NULL,
  `fechacreacion` DATETIME NULL,
  PRIMARY KEY (`id`),
  INDEX `ind_cliente` (`id` ASC),
  INDEX `ind_cliente_nom` (`nombre` ASC),
  INDEX `ind_cliente_alias` (`alias` ASC),
  INDEX `ind_cliente_poblacion` (`poblacion` ASC));
-- Id de usuario:
ALTER TABLE `signartdb`.`usuario`
CHANGE COLUMN `id` `id` INT(11) NOT NULL AUTO_INCREMENT ;
-- Id de cliente:
ALTER TABLE `signartdb`.`cliente`
CHANGE COLUMN `id` `id` INT(11) NOT NULL AUTO_INCREMENT ;
```

8.2. Creación de carpetas de trabajo

Crearemos en la raíz de nuestro equipo la carpeta SignartApp y dentro de la misma crearemos las siguientes 4 carpetas: workSpace, Maven, Tomcat7 y Eclipse. En la imagen aparece también una carpeta de doc, donde he ido dejando otros documentos, pero esta carpeta no es necesaria para la correcta instalación del entorno de trabajo.

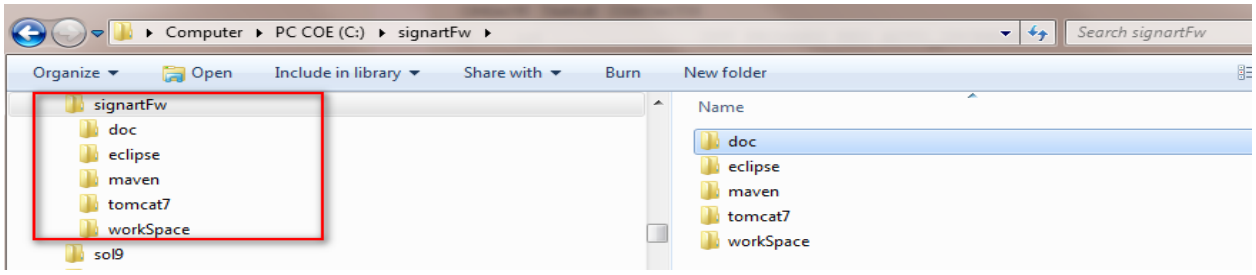


ILUSTRACIÓN 58 INSTALACIÓN CARPETAS

8.3. Descargar entorno desarrollo

En este paso descargaremos el entorno de trabajo con el cual seremos capaces de desarrollar en nuestro FrameWork de trabajo J2EE para SignartApp. Necesitaremos Spring FrameWork, Tomcat, Maven y como no el proyecto SigartApp.

Eclipse o Spring Framework: Es un software especialmente preparado para desarrollar aplicaciones J2EE. En concreto yo he utilizado la última versión que había publicada en Noviembre 2014 (spring-tool-suite-3.6.2.RELEASE-e4.4.1-win32-x86_64), la cual si no recuerdo mal, ya viene con el plugin de Maven. El software se puede descargar de la página: http://spring.io/tools/sts

Una vez descargado el zip, lo descomprimimos en la carpeta /signartApp/Eclipse y creamos un acceso directo en nuestro escritorio, para ternelo a mano:

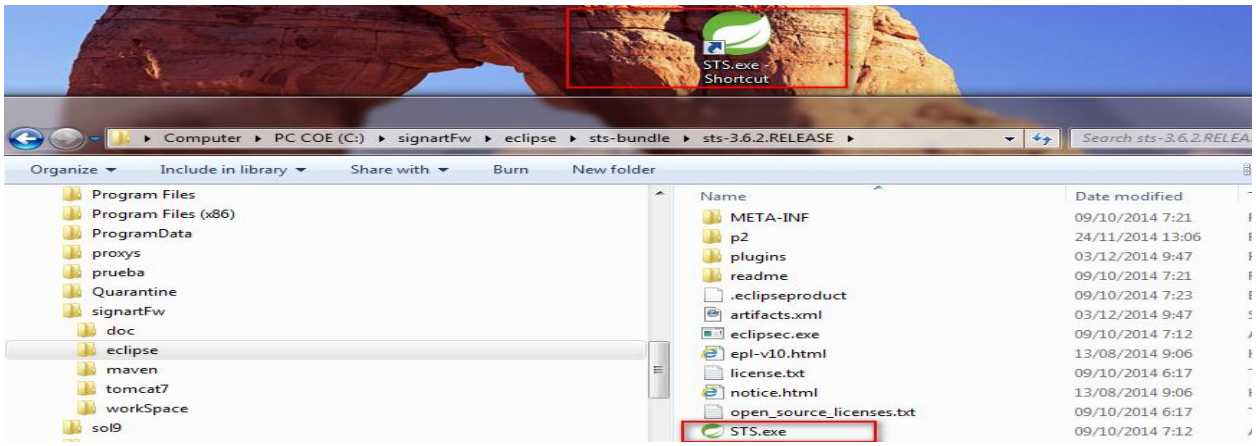


ILUSTRACIÓN 59 INSTALACIÓN STS

Tomcat 7: Ahora vamos a por el servidor de aplicaciones. Nos servirá con un Apache Tomcat 7. Es gratuito y lo podréis encontrar en http://tomcat.apache.org/download-70.cgi. En concreto he utilizado la versión 7. La cual he descargado y descomprimido en la carpeta /signartApp/tomcat7.

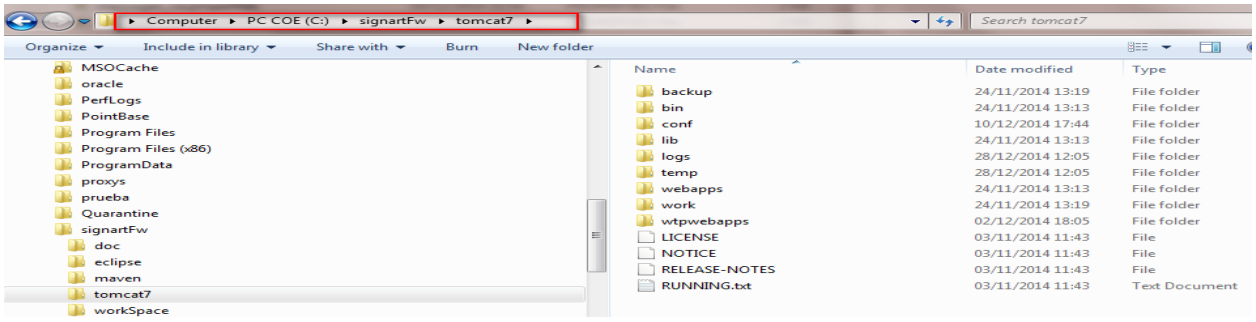


ILUSTRACIÓN 60 INSTALACIÓN TOMCAT7

Maven: Esta es una herramienta de Apache que nos facilita la construcción de Aplicaciones en Java. La descargaremos de http://maven.apache.org/download.cgi.

La versión que he utilizado es la apache-maven-3.2.3-bin y como no podía ser de otra manera la he descomprimido es su correspondiente carpeta.

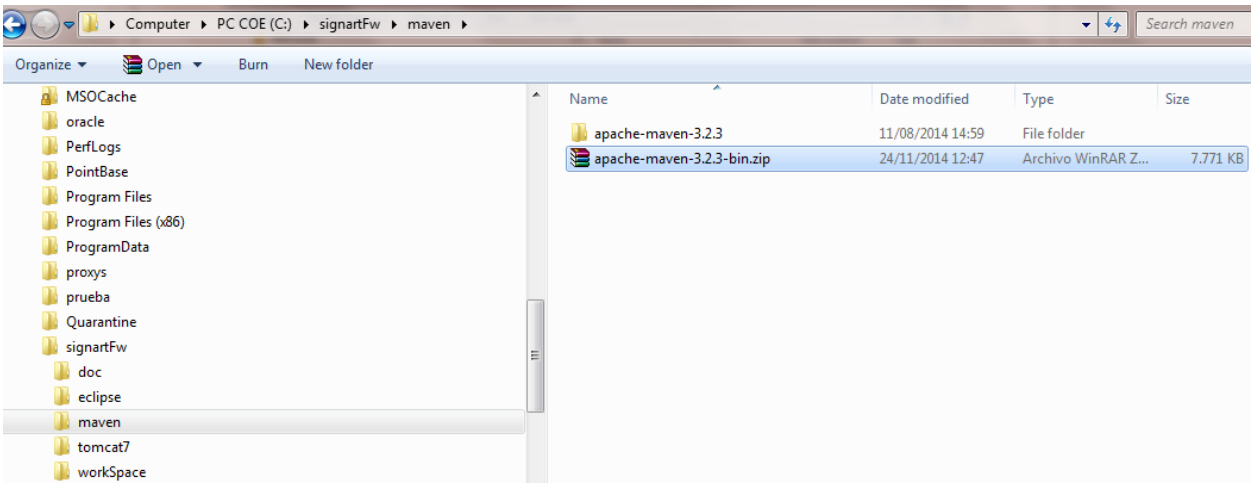


ILUSTRACIÓN 61 INSTALACIÓN MAVEN

SignartApp: Lo último que necesitamos es descargar la propia aplicación de test. Ella viene comprimida y la deberíamos descomprimir en el directorio C:\signartFw\workSpace

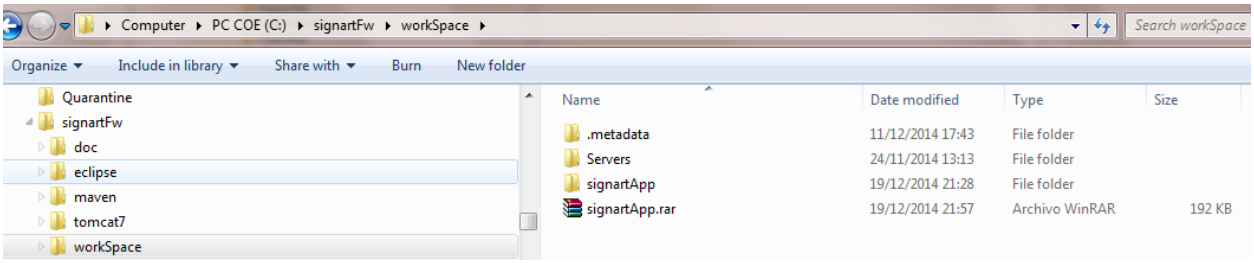


ILUSTRACIÓN 62 INSTALACIÓN SIGNARTAPP

8.4. Algunas configuraciones adicionales

MAVEN:

- En la variable de sistema PATH hemos de incluir referencia al Maven. Para ello copiamos la ruta donde la hemos instalado en el PATH. Para nosotros, C:\signartFw\maven\apache-maven-3.2.3\bin. Podemos comprobar que Maven funciona correctamente a traves de línea de comandes: Ejemplo línea de comandos Windows 7: cmd -> mvn –version y nos aparece que versión de maven tenemos instalada. Sino nos aperece es que tenmos algún problema con el path

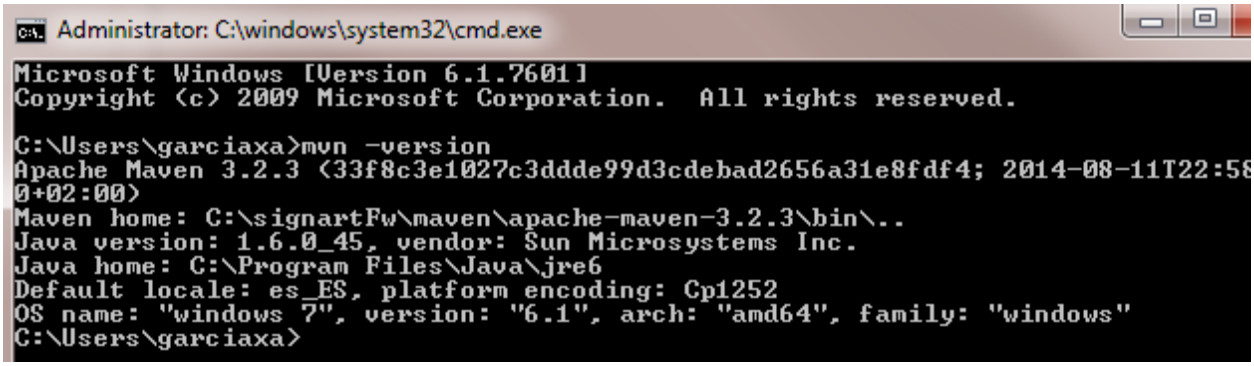


ILUSTRACIÓN 63 INSTALACIÓN CONFIGURACIÓN MAVEN

Eclipse o Spring FrameWork:

- Marcamos como WorkSpace el directorio que hemos creado C:\signartFw\workSpace.

- Marcamos el directorio de la instalación de Maven en el SpringFramework. Para ello desde la opción
  Windows -> Preferences -> Maven -> Installations. Seleccionamos la versión de maven C:\signartFw\maven\apache-maven-3.2.3

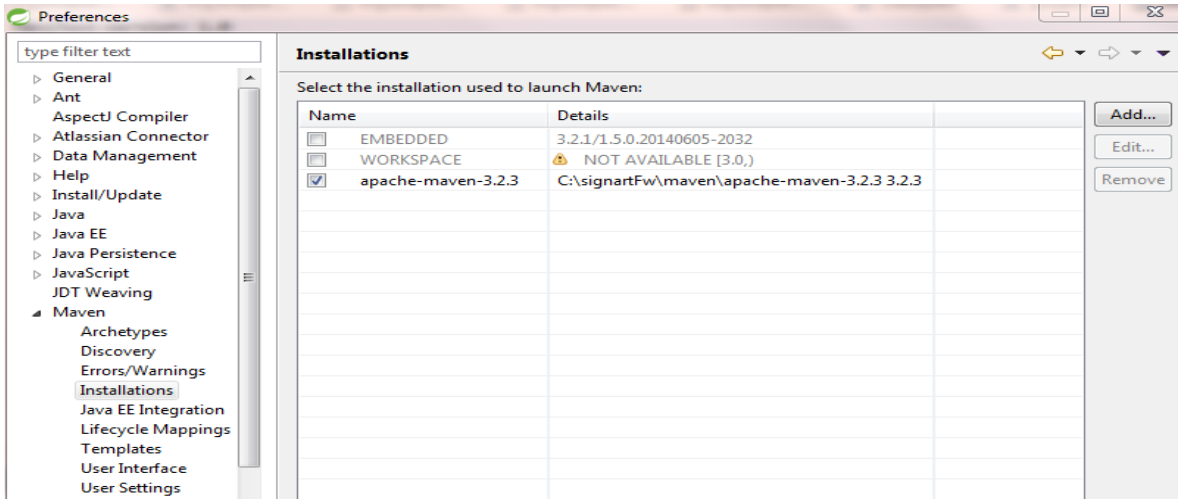


ILUSTRACIÓN 64 INSTALACIÓN STS MAVEN

- Creamos el servidor **Tomcat 7** en el SpringFramework a través del botón derecho de la pestaña de servers (sino aparece agregarla a través de Windows –Show Views -> Other -> Server), new server. En el apartado de configuración del mismo podemos indicarle el directorio donde hemos instalado Tomcat. En nuestro caso:

C:\signartFw\tomcat7

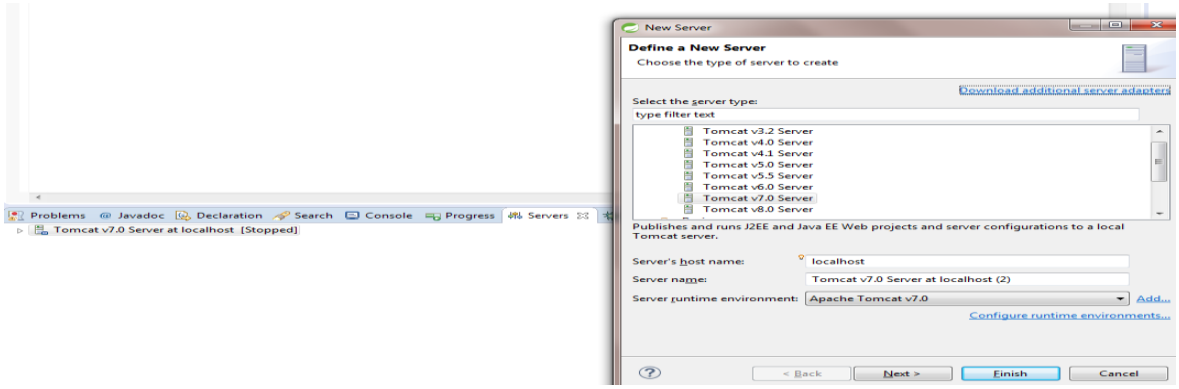


ILUSTRACIÓN 65 INSTALACIÓN TOMCAT7 ARRANQUE

Tomcat7: Una vez instalado Tomcat lo podemos arrancar. A través de la consola del propio SpringFramework podemos observar si hubiera algún tipo de error. Por otra parte si queremos acceder a la consola del Tomcat deberíamos primero modificar el fichero de configuración usuarios de tomcat C:\signartFw\tomcat7\conf\ tomcat-users.xml y escribir un usuario y password de la siguiente manera:

```
<tomcat-users>
<role rolename="manager"/>
<role rolename="admin"/>
<role rolename="manager-gui"/>
<user username="admin" password="admin" roles="admin,manager,manager-gui"/>
</tomcat-users>
```

Una vez modificado, arrancamos servidor y podemos acceder a la consola de tomcat a través de la siguiente dirección http://localhost:8081/. A la aplicaciones instaladas en nuestro local a través del enlace de Manager App o bien: http://localhost:8081/manager/html con el usuario y password que hemos indicado.

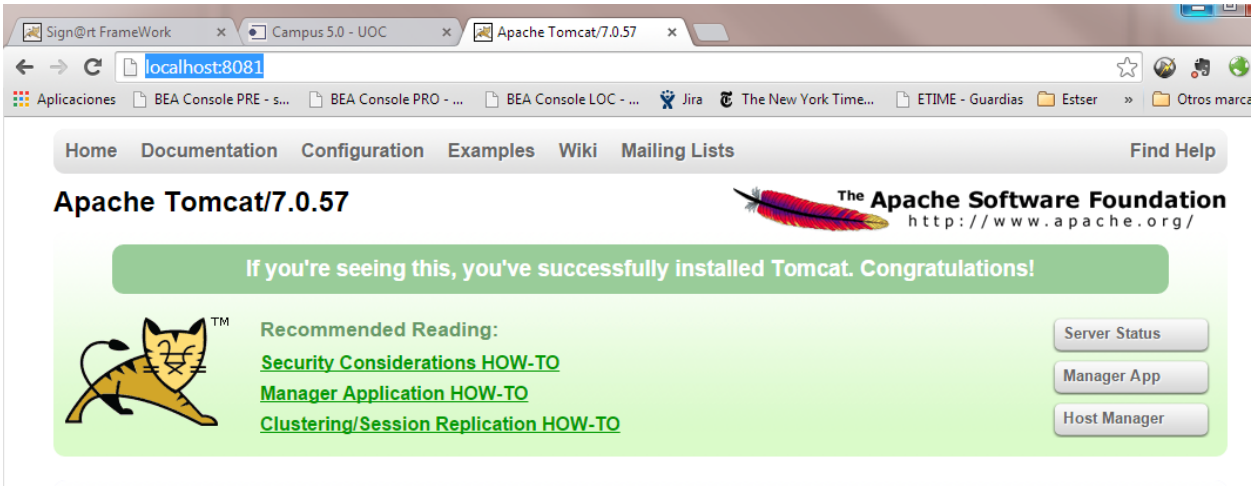


ILUSTRACIÓN 66 INSTALACIÓN TOMCAT7 WELCOME

En el menú Manager App deberíamos ver las aplicaciones desplegadas. En esta imagen ya nos aparece la imagen de SignArtApp. En el punto siguiente la importaremos desde Spring Framework pero para desplegar una aplicación J2EE en Tomcat 7 solo es necesario indicarlo el directorio base de la misma.

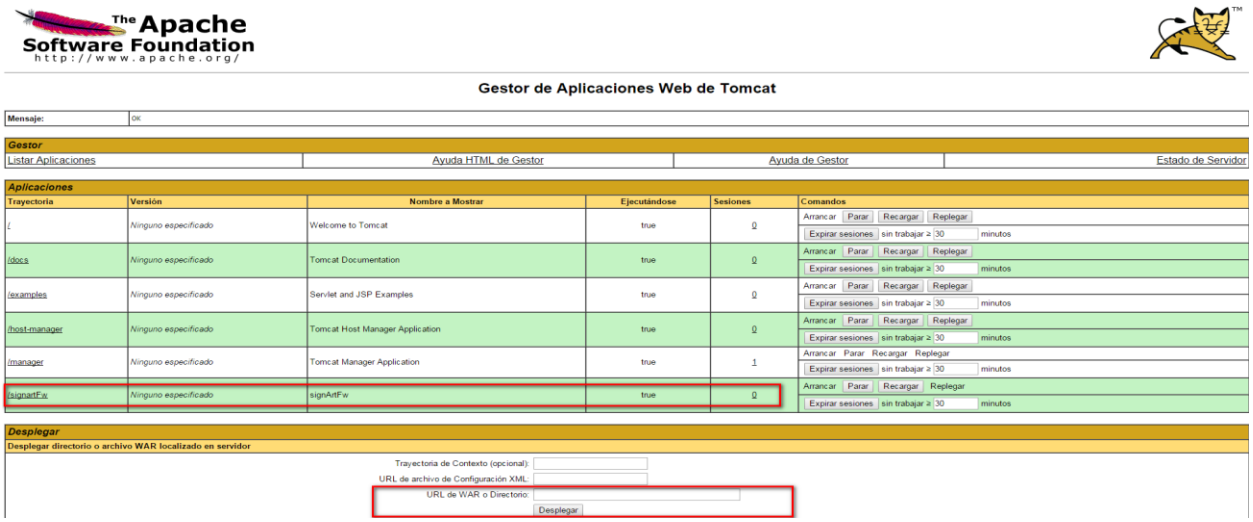


ILUSTRACIÓN 67 INSTALACIÓN TOMCAT7 SIGNARTAPP

8.5. Importamos el proyecto SignartApp

Una vez tenemos el entorno de instalación listo, solo nos faltará abrir la aplicación. Para ello a través del SpringFrameWork debemos importar la misma como un proyecto Maven. Esto se realiza mediante la siguiente opción File -> Import -> Maven -> Existing Maven projects.

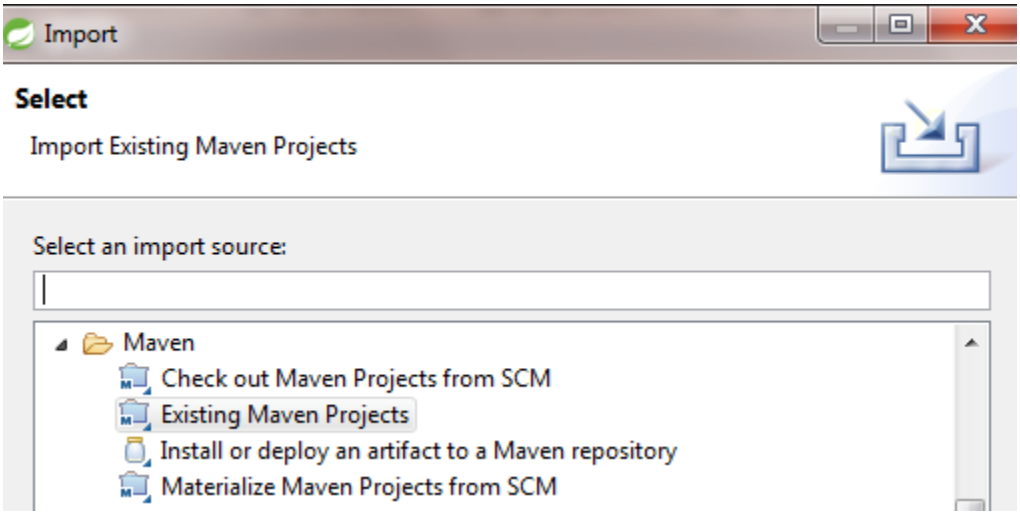


ILUSTRACIÓN 68 INSTALACIÓN MAVEN IMPORTAR

Y seleccionando la carpeta donde hemos descomprimido SignartApp. Para nuestro caso: C:\signartFw\workSpace\signartApp

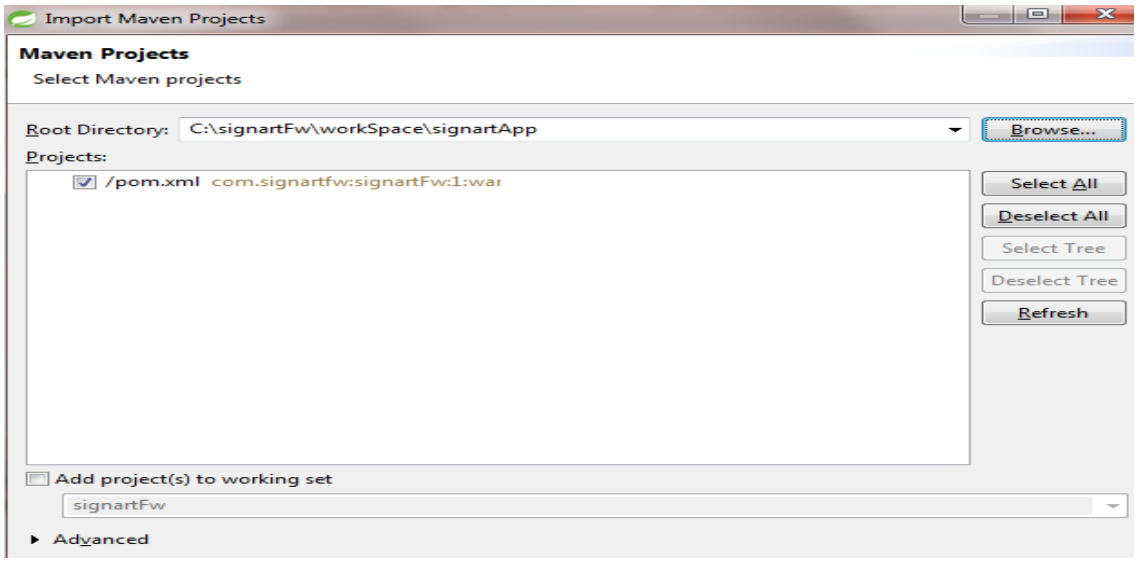


ILUSTRACIÓN 69 INSTALACIÓN MAVEN POM

Una vez desplegado el proyecto, y aunque ya debería haberse realizado automáticamente, deberíamos Updatear el proyecto con Maven para asegurarnos que tenga todas las herramientas instaladas. Esto se realiza a través de botón derecho encima del proyecto y clickando: SignartApp -> Maven -> Update Project

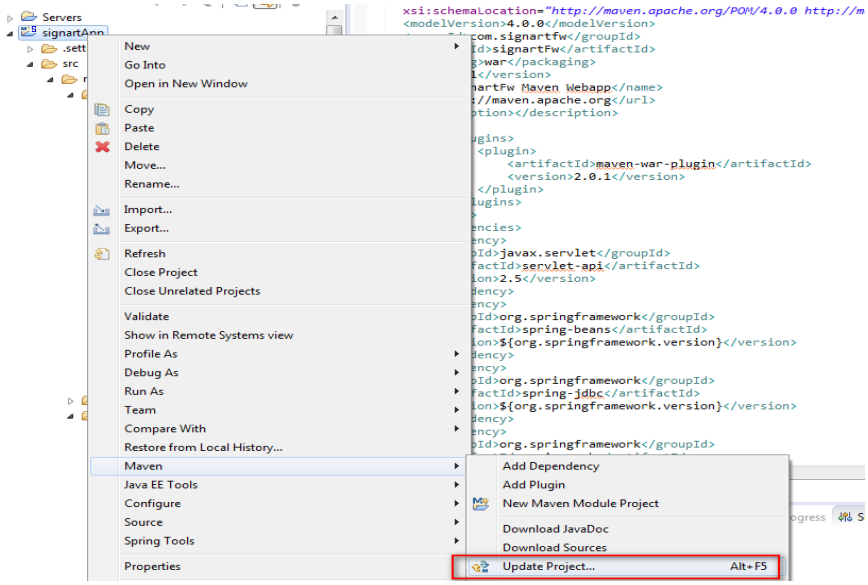


ILUSTRACIÓN 70 INSTALACIÓN MAVEN UPDATE

Una vez finalizado este paso ya tenemos la aplicación Signart preparada para ejecutarse. Podemos ver y desplegar todo el código de la aplicación.

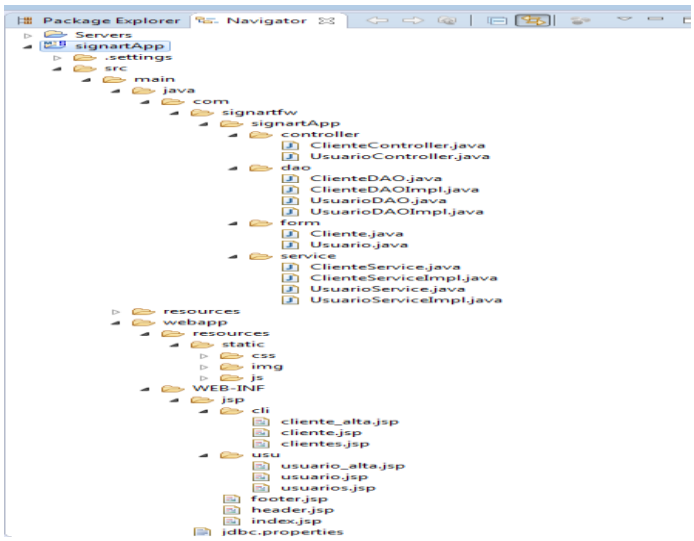


ILUSTRACIÓN 71 INSTALACIÓN VERIFICAR ESTRUCTURA

8.6. Arrancando SignartApp

Ahora solo nos falta arrancar la aplicación de test. Para ello deberemos arrancar tomcat desde el propio SpringFramework clicando encima del servidor y Restart server.

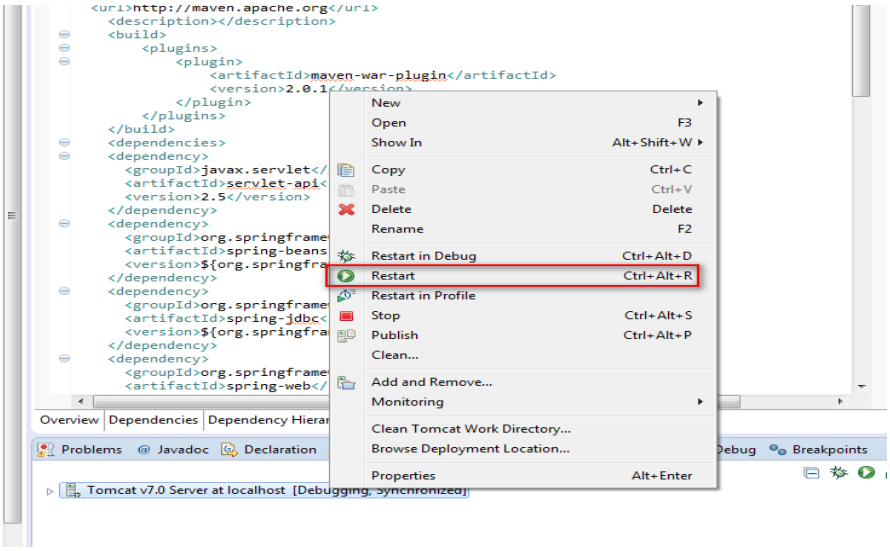


ILUSTRACIÓN 72 INSTALACIÓN ARRANQUE TOMCAT7

Una vez arrancado deberíamos acceder a la aplicación de pruebas mediante el siguiente enlace: http://localhost:8081/signartFw/

ILUSTRACIÓN 73 INSTALACIÓN WELCOME SIGNARTAPP

9. Conclusiones y mejoras

En el presente proyecto presentado he buscado varios objetivos, por un parte realizar una tarea de investigación inicial sobre los patrones Java de la capa de presentación y por otra revisar los principales Frameworks Java existentes en la actualidad.

Posteriormente y gracias a la investigación previa con los patrones de la capa de presentación J2EE, he creado un nuevo Framework de presentación para poder desarrollar aplicaciones web con carácter empresarial. Este Framework de capa de presentación es la suma de varias tecnologías encontradas, que pasan desde el Jquery para validar formularios, los estilos css para seguir una línea de diseño, taglibs para formularios y botoneras , o hibernate para presentar fácilmente los datos .

Finalmente, he implementado una aplicación de pruebas con el fin de que sea el punto de partida de una aplicación de gestión de una pequeña empresa de rotulación que actualmente tengo en mi entorno familiar.

La verdad es que ha supuesto para mí todo un reto realizar el conjunto de tareas en un breve periodo de tiempo. Pero la idea de estudiar y crear un nuevo Framework de presentación partiendo des de cero y realizar una aplicación para demostrar su funcionamiento ha supuesto para mí un atractivo reto.

El reto en mi caso continúa ya que me gustaría mejorar algunos aspectos de dicho framework y añadir más funcionalidades que sean útiles para la pequeña empresa. Gestión de presupuestos y facturas será el siguiente reto, intentando también digitalizar y clasificar estos documentos actualmente en papel en la base de datos.

10. Glosario

Apache Software Foundation: Fundación sin ánimo de lucro creada para apoyar los proyectos Apache, incluyendo el popular servidor HTTP Apache.

Core J2EE Patterns: Un patrón describe, con algún nivel de abstracción, una solución experta a un problema. Normalmente, un patrón está documentado en forma de una plantilla.

Framework: Conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar.

JavaBean: modelo de componentes creado por Sun Microsystems para la construcción de aplicaciones en Java.

Se usan para encapsular varios objetos en un único objeto (la vaina o Bean en inglés), para hacer uso de un solo objeto en lugar de varios más simples.

JDBC: Java Database Connectivity, más conocida por sus siglas JDBC, es una API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java, independientemente del sistema operativo donde se ejecute o de la base de datos a la cual se accede, utilizando el dialecto SQL del modelo de base de datos que se utilice.

J2EE: Java Platform, Enterprise Edition o Java EE, es una plataforma de programación—parte de la Plataforma Java—para desarrollar y ejecutar software de aplicaciones en el lenguaje de programación Java.

JSP: Tecnología que ayuda a los desarrolladores de software a crear páginas web dinámicas basadas en HTML, XML, entre otros tipos de documentos. JSP usa el lenguaje de programación Java.

MVC: Acrónimo para Modelo-Vista-Controlador, patrón de arquitectura en ingeniería de software. En una aplicación compleja que presenta una gran cantidad de datos al usuario, es deseable separar los datos (Modelo) y los problemas de la interfaz de usuario (Vista), de tal manera que los cambios en la interfaz no afecten el manejo de los datos, y que los datos puedan ser organizados sin cambiar la interfaz de usuario.

SQL: El lenguaje de consulta estructurado o SQL es un lenguaje declarativo de acceso a bases de datos relacionales que permite especificar diversos tipos de operaciones en ellas.

Taglib: Librería de clases java que realizan una ampliación de las etiquetas posibles de html. Podríamos llamar con unas etiquetas -tags- especiales a las clases java que hemos hecho en nuestra librería.

Tomcat: Funciona como un contenedor de servlets desarrollado bajo el proyecto Jakarta en la Apache Software Foundation. Tomcat implementa las especificaciones de los servlets y de JavaServer Pages (JSP) de Oracle Corporation (aunque creado por Sun Microsystems).

UML: Lenguaje Unificado de Modelado (UML, por sus siglas en inglés, Unified Modeling Language) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; está respaldado por el OMG (Object Management Group).

XML: Lenguaje de marcas extensible, es un metalenguaje extensible, de etiquetas, desarrollado por el World Wide Web Consortium (W3C). Permite la compatibilidad entre sistemas, permitiendo compartir información de una manera segura, fiable y fácil.

11. Bibliografía y referencias

Anoto algunos de los enlaces que me han servido como documentación para realizar el proyecto y el conjunto de la memoria:

- **Adictos al trabajo**
http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=patronesj2ee

- **Wikipedia. Comparación de Frameworks**
http://en.wikipedia.org/wiki/Comparison_of_web_application_frameworks

- **Spring Source Community. Spring Framework.**
http://static.springsource.org/

- **Uso de Java y estadísticas**
http://hotframeworks.com/languages/java

- **Patrones J2EE**
http://www.corej2eepatterns.com/AboutTheBook.htm

- **Java server Faces**
http://www.oracle.com/technetwork/java/javaee/documentation/index-137726.html

- **Java Struts**
https://struts.apache.org/birdseye.html

- **Maven documentación**
http://maven.apache.org/what-is-maven.html

- **JQuery documentación**
http://api.jquery.com/

- **Front Controller documentación**
http://java.dzone.com/articles/understanding-front-controller