Distribuido bajo licencia GPL

Puede copiar y distribuir el Programa (o un trabajo basado en él), según se especifica en el apartado 2, como código objeto o en formato ejecutable según los términos de los apartados 1 y 2, suponiendo que además cumpla una de las siguientes condiciones:

1. Acompañarlo con el código fuente completo correspondiente, en formato electrónico, que debe ser distribuido según se especifica en los apartados 1 y 2 de esta Licencia en un medio habitualmente utilizado para el intercambio de programas, o

2. Acompañarlo con una oferta por escrito, válida durante al menos tres años, de proporcionar a cualquier tercera parte una copia completa en formato electrónico del código fuente correspondiente, a un coste no mayor que el de realizar físicamente la distribución del fuente, que será distribuido bajo las condiciones descritas en los apartados 1 y 2 anteriores, en un medio habitualmente utilizado para el intercambio de programas, o

3. Acompañarlo con la información que recibió ofreciendo distribuir el código fuente correspondiente. (Esta opción se permite sólo para distribución no comercial y sólo si usted recibió el programa como código objeto o en formato ejecutable con tal oferta, de acuerdo con el apartado 2 anterior).

Universitat Oberta de Catalunya

www.uoc.edu

Ingeniería en Informática

2013/2014

Proyecto de fin de carrera:

Diseño e implementación de un framework de presentación.

El framework Myst

Autor: Miguel Souto Bartolomé

Consultor: Óscar Escudero Sánchez

13 de enero de 2014

Agradecimientos

Dedico el presente Proyecto de Fin de Carrera a mis padres, a su apoyo incondicional y a su buena voluntad para intentar encauzar con buen destino tanto mi educación como mi propio futuro. Espero sinceramente que este paso justifique, aunque sea de pequeña manera, parte de todo ese esfuerzo que han invertido en mí.

Agradezco muy especialmente a Judith el haberme facilitado la realización de este Proyecto. Muchas gracias por su comprensión y apoyo durante estos breves pero muy intensos meses.

Agradezco también a todo el mundo que a lo largo de mis estudios me ha apoyado y animado para llegar a su consecución, tanto amigos como familia. A todos ellos mi mayor agradecimiento por darme su confianza.

Resumen

En el mundo de Internet, al igual que en los sistemas operativos de escritorio, el desarrollo de aplicaciones es un enorme y muy importante sector de mercado. Desde sus orígenes, como sencillas aplicaciones cliente-servidor, hasta su situación actual, décadas de desarrollo e innovación han acercado al gran público estas aplicaciones, usadas diariamente por millones de personas desde todo tipo de dispositivos. La página web de un banco, una página para consultar el correo electrónico, las redes sociales, una tienda online; prácticamente todas las páginas web existentes utilizan, de una u otra forma, aplicaciones web.

Sin embargo, aunque dichas aplicaciones muchas veces sean transparentes para el usuario, o no le parezcan más complejas que cualquier aplicación de escritorio, lo cierto es que tienen una complejidad intrínseca subyacente, hay varios matices que las diferencian del resto de aplicaciones. Durante muchos años este hecho ha provocado que los programadores de aplicaciones web necesitaran adquirir conocimientos especiales de la arquitectura de Internet para fabricar sus productos.

Para tratar de atajar esta complejidad, el mundo de la programación web adoptó un concepto venido de la arquitectura de edificios, los patrones. Estos patrones consisten en soluciones contrastadas y de calidad a problemas frecuentes, y tienen como objetivo alcanzar la máxima divulgación para facilitar la labor de los programadores en todo el mundo.

Aunque los patrones han sido, sin duda, una ayuda efectiva a la problemática de las aplicaciones web, también ha sido superada por una combinación de soluciones que incluye bibliotecas de software, a los propios patrones y, sobre todo, una capa de abstracción que evita que el programador deba adquirir conocimientos especiales. Este cóctel de soluciones recibe el nombre de framework, o marco de trabajo.

Como es de esperar, no existe un framework único, sino que existe una amplia y rica variedad que responde a diferentes enfoques para enfrentar una misma problemática. Además, los frameworks en ocasiones se especializan en una de las diferentes capas problemáticas de una aplicación web, por lo que se pueden encontrar frameworks de acceso a datos, frameworks de presentación, etc., aunque no se sale de lo habitual encontrar soluciones que engloban todas las capas.

Dentro de los marcos de trabajo, Myst es un sencillo pero versátil framework de presentación, que convierte la tarea de programar la interfaz de una aplicación web en una tarea mucho más rápida y sencilla, y que no requiere conocimientos profundos sobre la arquitectura de internet.

Índice

1. Introducción 11
   1.1. Descripción del trabajo 11
   1.2. Objetivos del trabajo 11
   1.3. Plan de trabajo 12
   1.4. Tareas y subtareas 15
   1.5. Hitos del proyecto 15
   1.6. Productos a entregar 16
2. Patrones 17
   2.1. Introducción 17
   2.2. Singleton 18
   2.3. Fachada 18
   2.4. Command 18
   2.5. Inyección de dependencias 19
   2.6. Observador 20
   2.7. MVC 20
   2.8. Intercepting Filter 22
   2.9. Context Object 23
   2.10. Front Controller 23
   2.11. Application Controller 24
   2.12. View Helper 24
   2.13. Composite View 25
   2.14. Dispatcher View 25
   2.15. Service to Worker 26
3. Estudio de Frameworks 27
   3.1. Introducción 27
   3.2. Struts 29
      3.2.1. Arquitectura y características 29
      3.2.2. Ciclo de vida de una petición 30
      3.2.3. Configuración 31

3.3. Struts[2] 32
3.3.1. Arquitectura y características 33
3.3.2. Ciclo de vida de una petición 35
3.3.3. Configuración 36
3.4. Spring 37
3.4.1. Características 37
3.4.2. Arquitectura 38
3.4.3. Spring Web MVC 41
3.4.4. Ciclo de vida de una petición 42
3.4.5. Configuración 43
3.5. Tapestry 44
3.5.1. Características 45
3.5.2. Arquitectura 46
3.5.3. Ciclo de vida de una petición 47
3.5.4. Configuración 49
3.6. Comparativa de los frameworks 50
4. El Framework Myst 54
4.1. Análisis 55
4.1.1. Juego de funcionalidades 55
4.1.2. Patrones implementados 57
4.2. Diseño 59
4.2.1. Diagrama de clases 60
4.2.2. Descripción de las clases 61
4.2.3. Diagramas de secuencias 66
4.3. Implementación 70
4.3.1. Decisiones de diseño 70
4.3.2. Framework Myst 71
4.3.3. Manual de usuario 86
4.3.4. Aplicación para pruebas: MystTest 90
5. Aplicación de muestra: HelpDesk 94
5.1. Descripción del problema 94

5.2. Análisis 95
5.2.1. Diagrama de casos de uso 95
5.2.2. Diagrama de estados de una incidencia 96
5.2.3. Diagramas de actividades 97
5.2.4. Diagrama de despliegue 102
5.3. Diseño 103
5.4. Implementación 104
5.4.1. Capa de datos 104
5.4.2. Capa de lógica del negocio 105
5.4.3. Capa de presentación 107
5.5. Manual 113
5.5.1. Instalación de HelpDesk 113
5.5.2. Configuración de JBoss 114
5.5.3. Manual de usuario 116
6. Conclusiones 123
6.1. Sobre los frameworks 123
6.2. Trabajo personal 124
6.3. Mejoras futuras 125
7. Tecnologías 126
8. Glosario 126
9. Bibliografía 129

Tabla de ilustraciones

Ilustración 1: Diagrama de Gantt – Fases 1 y 2 13
Ilustración 2: Diagrama de Gantt – Fases 3 y 4 14
Ilustración 3: Patrón Singleton 18
Ilustración 4: Patrón Fachada 18
Ilustración 5: Patrón Command 19
Ilustración 6: Patrón Inyección de Dependencias 20
Ilustración 7: Patrón Observador 20
Ilustración 8: Patrón Modelo-Vista-Controlador 22
Ilustración 9: Patrón Intercepting Filter 22
Ilustración 10: Patrón Context Object 23
Ilustración 11: Patrón Front Controller 23
Ilustración 12: Patrón Application Controller 24
Ilustración 13: Patrón View Helper 24
Ilustración 14: Patrón Composite View 25
Ilustración 15: Patrón Dispatcher View 25
Ilustración 16: Patrón Service to Worker 26
Ilustración 17: Arquitectura de Struts 29
Ilustración 18: Diagrama de secuencia de una petición Struts 31
Ilustración 19: Ejemplo de archivo de configuración de Struts 32
Ilustración 20: Arquitectura de Struts^2 33
Ilustración 21: Resumen de una petición Struts2 35
Ilustración 22: Diagrama de secuencia de una petición Struts2 36
Ilustración 23: Ejemplo de archivo de configuración de Struts^2 37
Ilustración 24: Módulos de Spring 39
Ilustración 25: Arquitectura Spring MVC 41
Ilustración 26: Petición Spring simplificada 43
Ilustración 27: Controlador en Spring 43
Ilustración 28: Configuración de un nuevo interceptor en Spring 44
Ilustración 29: Arquitectura de Tapestry 46
Ilustración 30: Ciclo de vida de una petición Tapestry 48
Ilustración 31: Context Object - Diagrama de secuencias 57
Ilustración 32: Service to Worker- Diagrama de clases 58
Ilustración 33: Service to Worker - Diagrama de secuencias 59
Ilustración 34: Diagrama de clases de Myst 60
Ilustración 35: Diagrama de secuencias - Carga del framework 66
Ilustración 36: Diagrama de secuencias - Gestión de una petición 68
Ilustración 37: Paquetes del framework Myst 71
Ilustración 38: Clases del paquete Core 72

Ilustración 39: Clases del paquete Configuration 74
Ilustración 40: Clases del paquete Command 75
Ilustración 41: Clases del paquete Form 76
Ilustración 42: Clases del paquete View 77
Ilustración 43: Clases del paquete Exception 79
Ilustración 44: Clases del paquete Filter 81
Ilustración 45: Clases del paquete Auxiliary 82
Ilustración 46: Clases del paquete International 83
Ilustración 47: Clases del paquete Taglib 84
Ilustración 48: : Ejemplo de archivo de configuración de Myst 87
Ilustración 49: Menú de la aplicación de prueba 91
Ilustración 50: Tests con resultados exitosos 91
Ilustración 51: Tests que provocan errores 92
Ilustración 52: Ejemplo de error internacionalizado 93
Ilustración 53: Diagrama de casos de uso 95
Ilustración 54: Diagrama de estados de una incidencia 96
Ilustración 55: Diagrama de actividad - login 98
Ilustración 56: Diagrama de actividad - mostrar incidencia 99
Ilustración 57: Diagrama de actividad - marcar incidencia como en trámite 100
Ilustración 58: Diagrama de actividad - cerrar incidencia 101
Ilustración 59: Diagrama de actividad - buscar todos los trabajadores 102
Ilustración 60: Diagrama de despliegue 102
Ilustración 61: Diagrama Entidad-Relación de la base de datos 105
Ilustración 62: Paquetes de la aplicación HelpDesk-ejb 105
Ilustración 63: Clases del paquete Entities 106
Ilustración 64: Clases del paquete Bussinesslogic 106
Ilustración 65: Paquetes de la aplicación HelpDesk-web 107
Ilustración 66: Clases del paquete Command 108
Ilustración 67: Clases del paquete Form 109
Ilustración 68: Clases del paquete Filter 109
Ilustración 69: Clases del paquete Exception 110
Ilustración 70: Clases del paquete ExceptionHandler 111
Ilustración 71: Clase del paquete Utility 111
Ilustración 72: Páginas JSP de la aplicación 112
Ilustración 73: Configuración de module.xml 115
Ilustración 74: Configuración de driver en standalone.xml 115
Ilustración 75: Configuración de origen de datos en standalone.xml 115
Ilustración 76: HelpDesk - pantalla de login 116
Ilustración 77: HelpDesk - fallo de login 116
Ilustración 78: HelpDesk - pantalla de menú ingeniero 117

Ilustración 79: HelpDesk - pantalla de mostrar incidencias 117
Ilustración 80: HelpDesk - pantalla de incidencia para una ingeniera 118
Ilustración 81: HelpDesk - pantalla de incidencia marcada como como en proceso... 119
Ilustración 82: HelpDesk - pantalla inicial en inglés para administradores 120
Ilustración 83: HelpDesk - pantalla con todos los trabajadores..... 120
Ilustración 84: HelpDesk - pantalla con las incidencias de una trabajadora..... 121
Ilustración 85: HelpDesk - pantalla de incidencia para un administrador 121

1. Introducción

1.1. Descripción del trabajo

Para la realización del Proyecto final de carrera, y ante mi ausencia de contacto con el mercado de trabajo informático, he decidido aceptar una de las propuestas del departamento, en concreto la de realizar un framework de presentación.

Como dice la propuesta, un framework de presentación es un marco que ayuda a la programación de aplicaciones web, abstrayendo al programador, en la medida de lo posible, de la dificultad subyacente. Es por ello que un framework provee de herramientas para facilitar la configuración del servidor web, la presentación de la información como páginas web, el control del flujo, etc.

Como paso previo a la programación del framework, se realizará un estudio de varios frameworks de presentación populares basados en el patrón Modelo-Vista-Controlador; se estudiará su origen, su evolución, las diferentes vertientes existentes, los patrones de diseño más utilizados, y se extraerá un catálogo de características comunes que resulten de especial interés. Este catálogo servirá de base para especificar los requisitos de nuestro framework.

Una vez definidos los requisitos, se harán las tareas de análisis y diseño del framework, y también se proyectará una aplicación que utilizará el framework y servirá para testarlo y comprobar su utilidad. La aplicación estará orientada, pues, a probar la capa de presentación, resultando circunstanciales las capas de lógica de negocio y de datos. Finalizada esta etapa, se completará el trabajo con la implementación del framework y los juegos de pruebas pertinentes, estando todo ellos debidamente documentados.

El proyecto finalizará con la entrega del framework, la aplicación de prueba, la memoria y la presentación, pudiendo ampliarse el número de entregables a demanda del consultor.

1.2. Objetivos del trabajo

- Conocer las diferentes aproximaciones para ayudar a desarrollar la capa de presentación en aplicaciones JEE. Estudiar un conjunto relevante de los frameworks y extraer sus funcionalidades básicas.
- Diseñar e implementar un framework de presentación que facilite las funcionalidades básicas extraídas.

- Construir una aplicación JEE que utilice el framework implementado, y que muestre claramente su capacidad de abstracción sobre diversos conceptos de aplicaciones empresariales.

1.3. Plan de trabajo

La planificación del proyecto se dividirá en cuatro grandes fases o bloques, los cuales coincidirán con las fechas de entregas señaladas en el aula del proyecto.

1º Fase – Propuesta de proyecto

Estudio previo de patrones y frameworks de presentación. Propuesta del proyecto de fin de carrera, especificando los objetivos a alcanzar y la planificación de su realización.

2ª Fase – Análisis y diseño

Estudio en profundidad de patrones y frameworks de presentación. Definición de los requisitos del framework, análisis y diseño.

3ª Fase – Implementación

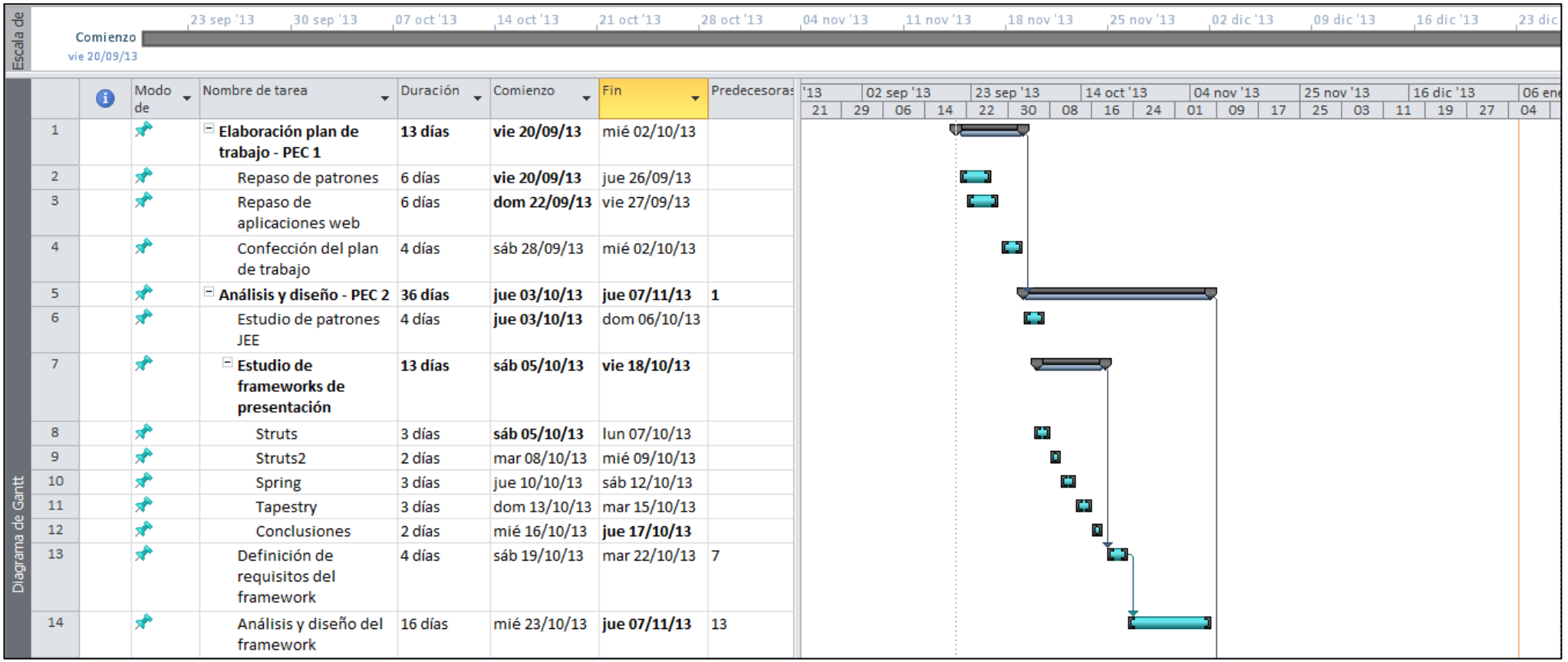
Implementación del framework de presentación. Batería de pruebas a través de la aplicación de pruebas.

4ª Fase – Entrega del proyecto

Redacción de la memoria, elaboración de la presentación, programación de la aplicación de muestra, depuración y mejora del framework.

Para mayor claridad, se proporciona también el cronograma del proyecto, con las diferentes tareas a realizar en forma de diagrama de Gantt. Dicho diagrama se muestra en las siguientes imágenes.

Diagrama de Gantt:



	Modo de	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1		**Elaboración plan de trabajo - PEC 1**	**13 días**	**vie 20/09/13**	mié 02/10/13	
2		Repaso de patrones	6 días	**vie 20/09/13**	jue 26/09/13	
3		Repaso de aplicaciones web	6 días	**dom 22/09/13**	vie 27/09/13	
4		Confección del plan de trabajo	4 días	sáb 28/09/13	mié 02/10/13	
5		**Análisis y diseño - PEC 2**	**36 días**	**jue 03/10/13**	**jue 07/11/13**	**1**
6		Estudio de patrones JEE	4 días	**jue 03/10/13**	dom 06/10/13	
7		**Estudio de frameworks de presentación**	**13 días**	**sáb 05/10/13**	**vie 18/10/13**	
8		Struts	3 días	**sáb 05/10/13**	lun 07/10/13	
9		Struts2	2 días	mar 08/10/13	mié 09/10/13	
10		Spring	3 días	jue 10/10/13	sáb 12/10/13	
11		Tapestry	3 días	dom 13/10/13	mar 15/10/13	
12		Conclusiones	2 días	mié 16/10/13	**jue 17/10/13**	
13		Definición de requisitos del framework	4 días	sáb 19/10/13	mar 22/10/13	7
14		Análisis y diseño del framework	16 días	mié 23/10/13	**jue 07/11/13**	13

Ilustración 1: Diagrama de Gantt – Fases 1 y 2

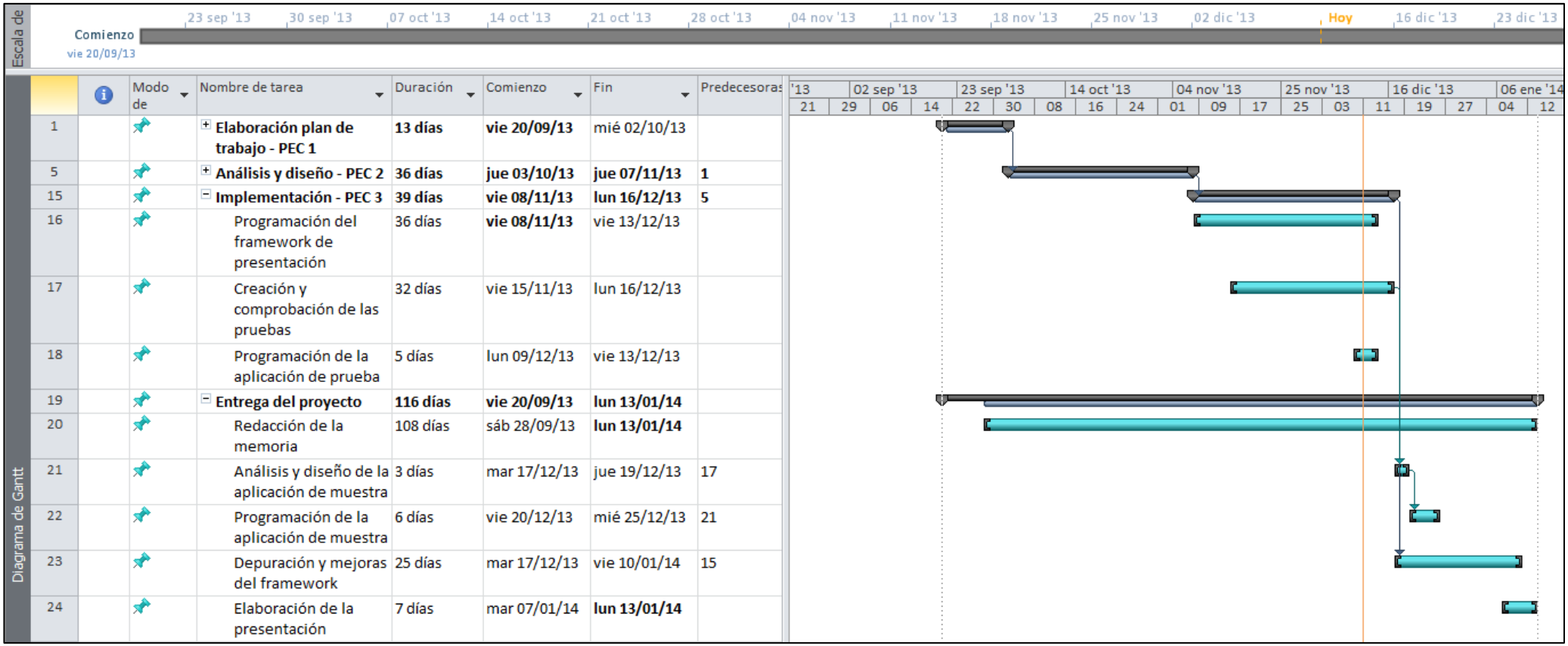


Ilustración 2: Diagrama de Gantt – Fases 3 y 4

14

1.4. Tareas y subtareas

- Estudio de los patrones de diseño más utilizados en la capa de presentación de una aplicación web.
- Estudio de cuatro frameworks JEE de presentación de gran relevancia: Struts, Struts^2, Spring y Tapestry.
    - Extracción de las funcionalidades más reseñables.
    - Análisis comparativo.
- Creación de un framework de presentación para aplicaciones JEE.
    - Implementa un juego de funcionalidades extraído del estudio de los frameworks existentes.
    - Etapas de análisis y diseño.
    - Implementación y juegos de pruebas.
- Programación de una aplicación JEE que utilice el framework creado.
    - Análisis y diseño.
    - Implementación y demostración.
- Documentación del proyecto.
    - Código auto documentado.
    - Memoria del proyecto.
- Presentación
    - Presentación basada en diapositivas donde se explique el objetivo del proyecto, la funcionalidad del framework y las conclusiones.

1.5. Hitos del proyecto

2 de Octubre de 2013 - PEC 1

- Propuesta del proyecto.
- Entrega del plan de trabajo.

7 de Noviembre de 2013 - PEC 2

- Análisis y diseño del framework.

- Memoria sobre el estado actual del proyecto, incluyendo el estudio sobre los patrones y frameworks de presentación.

16 de Diciembre de 2013 - PEC 3

- Entrega del framework JEE de presentación.
- Entrega de la aplicación de prueba.
- Presentación del estado de las baterías de pruebas.
- Memoria sobre el estado actual del proyecto.

13 de Enero de 2014 - Entrega del proyecto

- Entrega del framework de presentación y su aplicación de prueba completamente operativos y documentados.
- Memoria de la batería de pruebas y sus resultados.
- Entrega de documentación finalizada.
- Entrega de la memoria del Proyecto de Fin de Carrera.
- Entrega de la presentación.

1.6. Productos a entregar

- Código auto documentado del framework y aplicación de prueba.
- Código y resultado de las pruebas.
- Memoria del proyecto.
- Presentación.

2. Patrones

2.1. Introducción

- patrón, na. (Del lat. patrōnus). m. Modelo que sirve de muestra para sacar otra cosa igual.

El vocablo patrón es utilizado frecuentemente en el habla cotidiana, tal vez en la mayoría de ocasiones para referirse a un modelo de conducta o de comportamiento; también se identifica popularmente con las plantillas a repetir una y otra vez en la confección de tejidos. En el mundo de la ingeniería del software, se ha adoptado este término para referirse a soluciones aplicables a diversos problemas que se repiten asiduamente en el diseño de un producto software.

Históricamente, los patrones de diseño cobraron popularidad con la publicación en 1994 del libro "Design Patterns: Elements of Reusable Object-Oriented Software", de los autores conocidos como *Gang of Four*. En este libro apareció una primera lista de 23 patrones de diseño, clasificados en tres grandes grupos: creacionales, estructurales y de comportamiento.

Los patrones de diseño aportan varias ventajas. Por una parte, se dispone de un repositorio de conocimiento, de soluciones reutilizables, que puede ser usada como material de consulta, pero también como material didáctico. Por otro lado, su uso en el diseño de aplicaciones ahorra tiempo y esfuerzo, al aprovechar la experiencia de otros ingenieros, que identificaron un problema similar y crearon una solución correcta y adaptable.

Con el tiempo la lista de patrones ha ido creciendo, hasta llegar a un número y variedad de patrones prácticamente inabarcable. Dichos patrones no han escapado a las críticas, siendo las más feroces (y en cierta manera certeras) las dirigidas a que su necesidad viene motivada por las carencias de los lenguajes orientados al objeto. Otras críticas se dirigen al uso excesivo de patrones en circunstancias innecesarias, lo que perjudica el diseño del software, añadiendo una complejidad innecesaria.

Sin embargo, la experiencia ha venido a demostrar que la aplicación de patrones de diseño en programación orientada al objeto es muy aconsejable, y que su utilidad en el diseño de aplicaciones web resulta prácticamente incontestable. A continuación se describirán varios patrones que han alcanzado gran relevancia en su aplicación a aplicaciones web.

2.2. Singleton

Patrón de diseño que se utiliza para restringir a una el número de instancias de una clase. Solamente es útil cuando realmente es necesaria la existencia de una única instancia, aunque, desafortunadamente, muchas veces se abusa del patrón y se utiliza erróneamente en situaciones en las que no conviene aplicarlo.

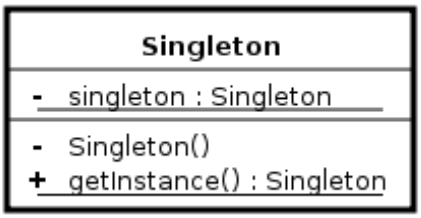


Ilustración 3: Patrón Singleton

2.3. Fachada

Patrón de diseño que proporciona una interfaz simple y única para acceder a un subsistema. Las razones de utilizar un único punto de acceso pueden ser varias, como ocultar la complejidad del subsistema subyacente, o desacoplar el subsistema de sus diferentes clientes para mejorar su portabilidad.

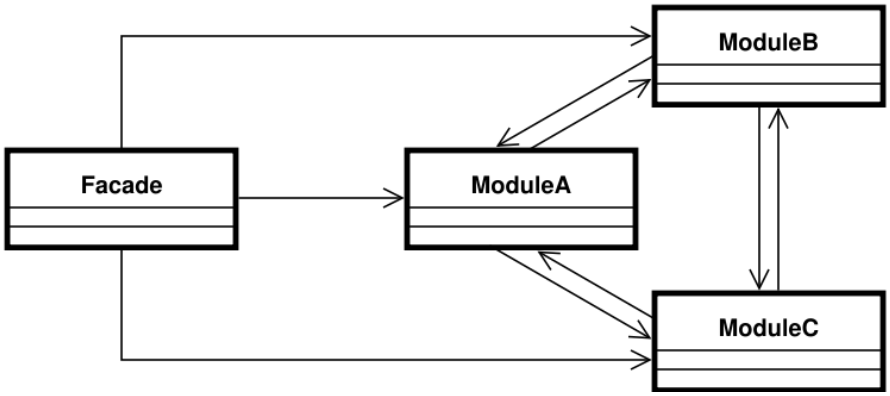


Ilustración 4: Patrón Fachada

2.4. Command

Patrón de comportamiento por el que se crea un objeto que encapsula y representa la información necesaria para invocar a un método en un futuro. De esta forma, se puede solicitar una operación a un objeto sin conocer realmente el método ni el receptor real de la petición.

Este patrón facilita la construcción de componentes generales, facilitando la parametrización. Permite definir la lógica delegando el comando concreto al momento de la ejecución.

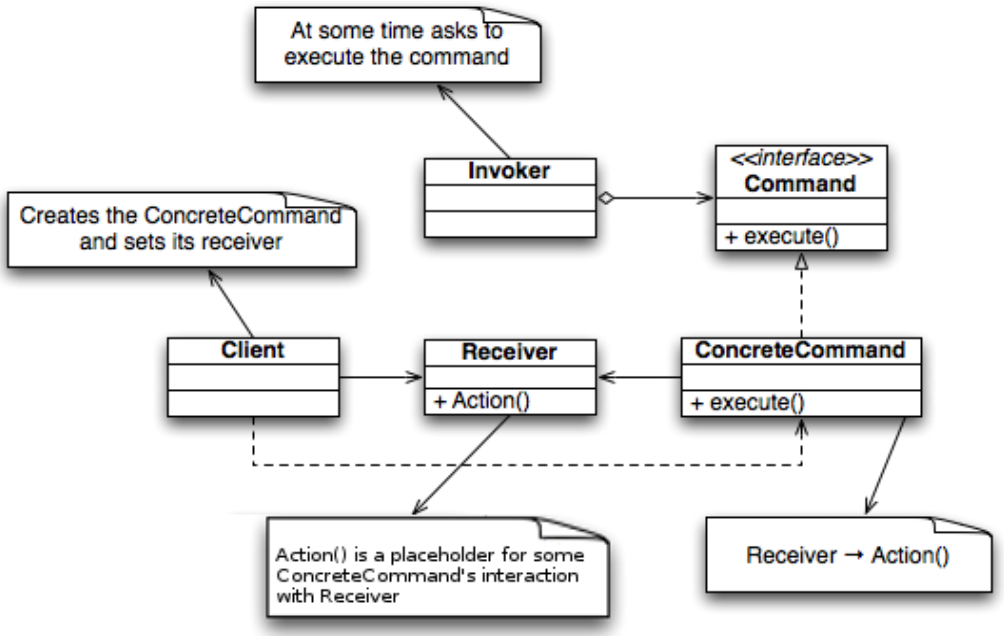


Ilustración 5: Patrón Command

2.5. Inyección de dependencias

Patrón de diseño por el cual se suministran objetos a una clase en lugar de ser ella misma la que instancie objetos de las clases que le interesen.

Su objetivo es permitir seleccionar entre diferentes implementaciones de una interfaz de dependencias en tiempo de ejecución, en lugar de en tiempo de compilación. Esta selección se puede realizar, entre otras maneras, a través de un archivo de configuración.

La inyección de dependencias es una gran herramienta para ayudar a la realización de pruebas unitarias en grandes componentes de software, facilitando la utilización de objetos simulados o mocks. También permite la carga dinámica de plugins.

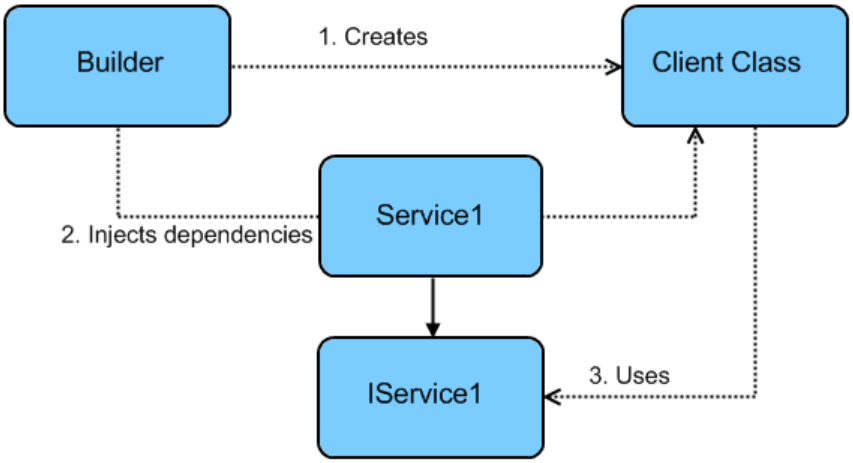


Ilustración 6: Patrón Inyección de Dependencias

2.6. Observador

Patrón de diseño creado para solventar la problemática del acoplamiento entre un objeto susceptible a cambios (Sujeto), y los objetos dependientes que deben ser notificados de dichos cambios (Observadores); en otras palabras, soluciona la problemática de la dependencia uno-a-muchos.

Se crea un objeto abstracto que observará los cambios y los notificará a los objetos dependientes, minimizando su acoplamiento y aumentando la modularidad.

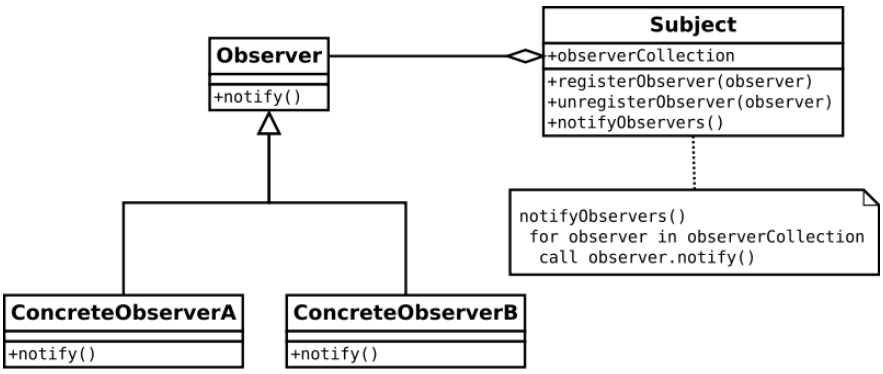


Ilustración 7: Patrón Observador

El patrón Observador es la base del patrón MVC, definiendo el Controlador.

2.7. MVC

Modelo-Vista-Controlador es un patrón de tipo arquitectónico muy usado en la capa de presentación, que ha ganado gran popularidad por su excelente adecuación al diseño de aplicaciones web. El objetivo de este patrón es desacoplar la interfaz gráfica,

una parte diferenciada y compleja de toda aplicación, de los datos y la lógica del negocio, de forma que puedan tratarse como problemas distintos.

La solución que este patrón propone es separar la problemática en tres módulos distintos:

- **El Modelo**: representa los datos de la aplicación y la lógica del negocio. Su función es gestionar el acceso a la información, y enviar a la Vista la información a mostrar. La manipulación del Modelo se realiza a través del Controlador.
- **La Vista**: presenta la información al usuario a través de una interfaz.
- **El Controlador**: recoge las peticiones que el usuario emite a través de la interfaz, y las convierte en órdenes que envía al Modelo. Además modifica la Vista en caso de que varíe la presentación del modelo.

Los primeros frameworks MVC definían un cliente ligero, por lo que la mayoría de la funcionalidad residía en el servidor. Con el avance de la microinformática, los clientes han aumentado en gran medida su potencia, lo que unido a tecnologías como JavaScript permite una mayor riqueza y capacidad de procesamiento en los clientes. De esta manera, gran parte de las tareas de presentación, así como ciertas tareas del modelo (comprobación de datos, etc.) residen en el cliente.

Aunque puede variar en mayor o menor medida, la interacción clásica con un sistema basado en el patrón Modelo-Vista-Controlador sería el siguiente:

1. El usuario interacciona con el sistema a través de la vista, enviando peticiones al controlador.
2. El controlador recibe las peticiones de usuario. En caso de ser necesario, realiza peticiones o modificaciones en el modelo.
3. Una vez terminadas las operaciones en el modelo, el controlador selecciona la vista a mostrar. La vista obtendrá la información que debe mostrar del modelo.
4. Se presenta la nueva vista al usuario.

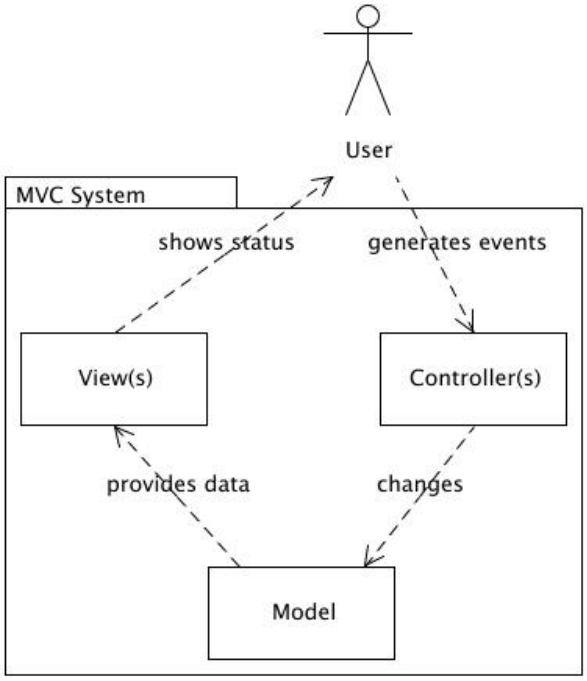


Ilustración 8: Patrón Modelo-Vista-Controlador

El patrón Modelo-Vista-Controlador es la base de la mayoría de las aplicaciones web empresariales, y también de gran parte de los frameworks de presentación. Por todo ello que cobra especial relevancia en nuestro estudio.

A continuación se verán los patrones JEE de presentación más usuales, los conocidos como centrales, todos ellos situados en un entorno Modelo-Vista-Controlador.

2.8. Intercepting Filter

Ayuda a realizar tareas de pre y post procesamiento sobre una petición. Se intercepta la petición, y se cede el control de la misma a un gestor de filtros, que decidirá el orden en el que se realizan dichass tareas, así como cuándo se gestiona la propia petición.

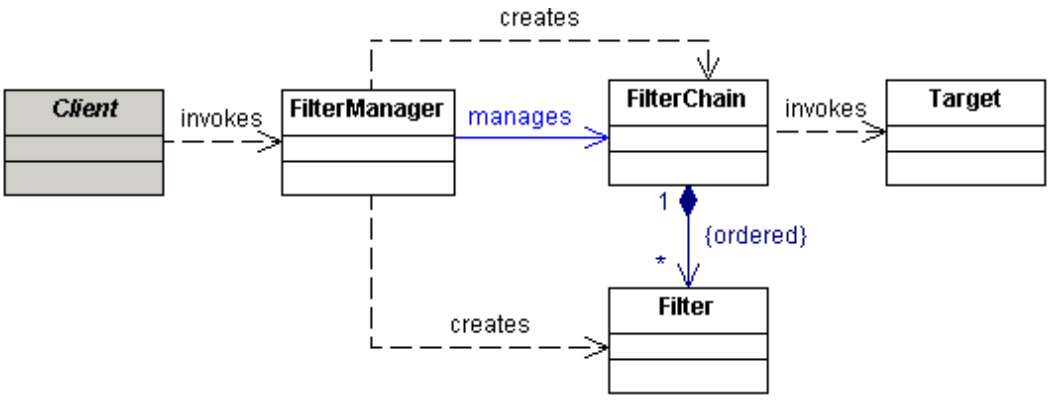


Ilustración 9: Patrón Intercepting Filter

2.9. Context Object

Propone la creación de una clase para almacenar el estado del sistema, de forma que los componentes y servicios de la aplicación puedan acceder a esa información de forma independiente al protocolo del sistema.

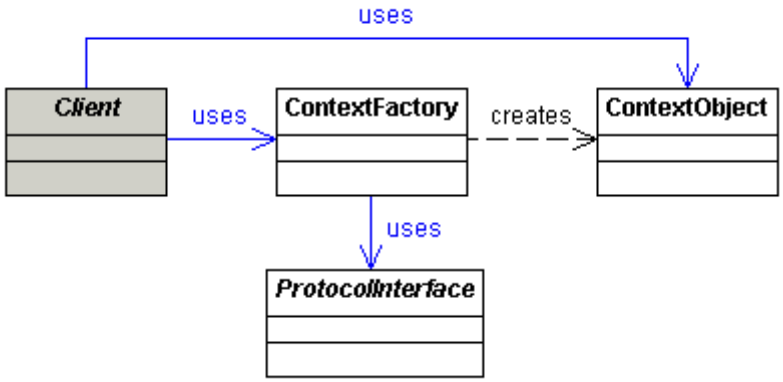


Ilustración 10: Patrón Context Object

2.10. Front Controller

Patrón de diseño que provee un único punto de acceso para procesar las peticiones del cliente. De esta forma, resulta mucho más sencillo controlar el flujo de la aplicación. También evita la duplicación de muchas tareas comunes a peticiones, como las relacionadas con seguridad, internacionalización, o la personalización de la presentación para los usuarios.

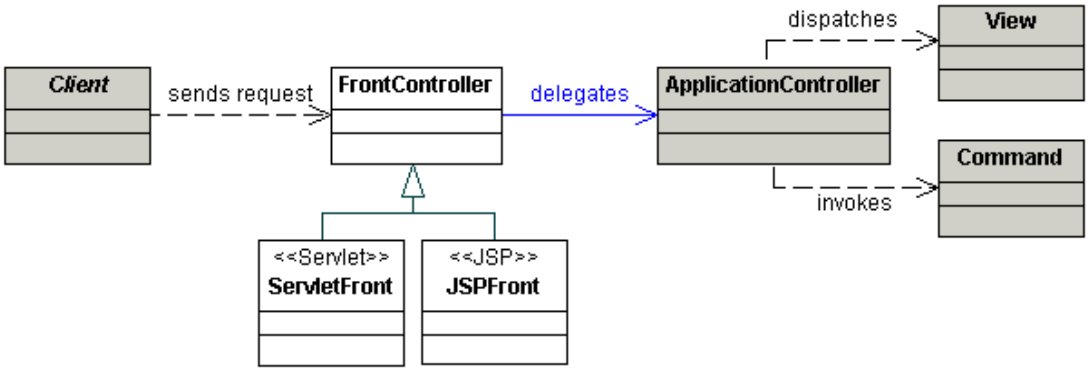


Ilustración 11: Patrón Front Controller

2.11. Application Controller

De forma similar a Front Controller, proporciona un único punto para gestionar el flujo de la aplicación y la navegación. Dicho de otra manera, controla de forma centralizada, según las peticiones recibidas, las acciones a realizar y las vistas a mostrar como respuesta.

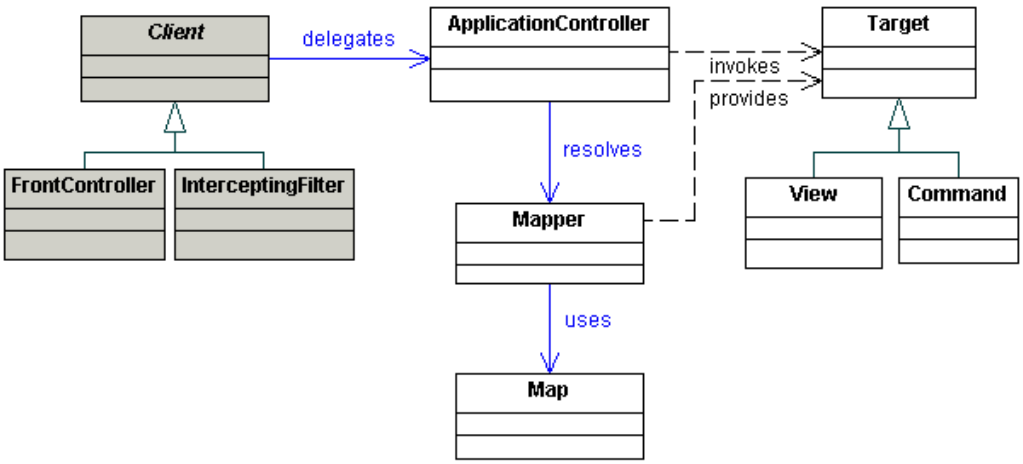


Ilustración 12: Patrón Application Controller

2.12. View Helper

Como su nombre sugiere, ayuda a separar de una vista su lógica de proceso. La vista delegará el procesamiento en una clase de ayuda. Esta clase servirá de adaptador entre el modelo y la vista, y realizará tareas de formateo, comúnmente a través de tags.

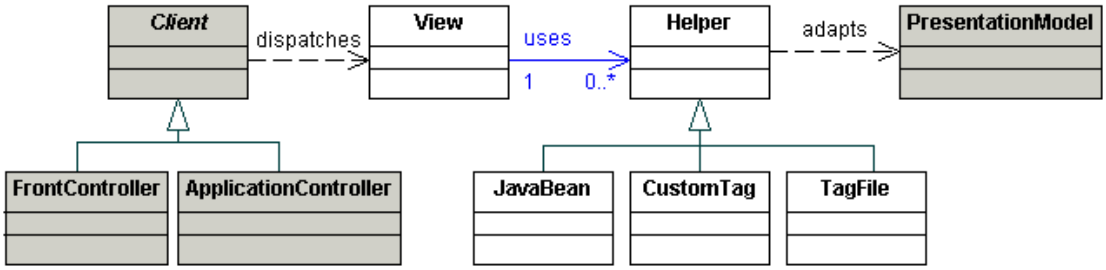


Ilustración 13: Patrón View Helper

2.13. Composite View

En realidad más que un patrón en sí mismo, es la especialización del patrón Composite para gestionar vistas complejas en una aplicación web.

Composite View permite crear vistas complejas a partir de componentes, o vistas, más sencillas. Además, permite que el diseño de la vista se gestione de forma independiente al contenido.

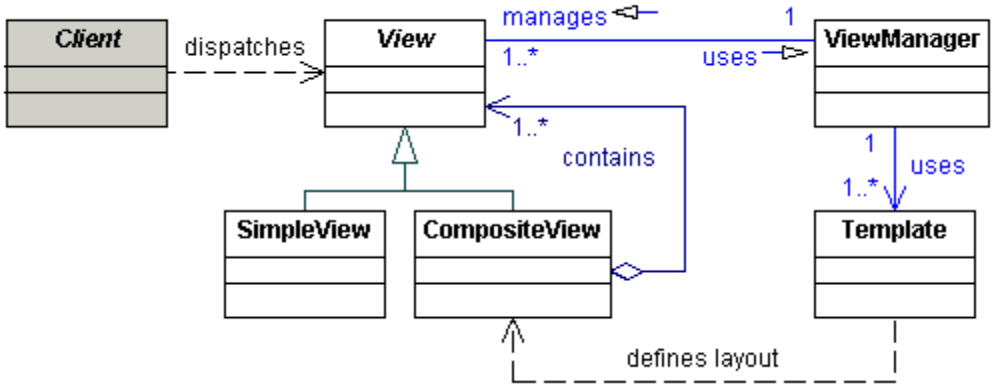


Ilustración 14: Patrón Composite View

2.14. Dispatcher View

Permite que una vista se encargue de gestionar una petición y generar la respuesta utilizando una cantidad mínima de lógica de negocio. Es un patrón que se debe aplicar con mucho cuidado, puesto que puede generar acoplamiento entre la lógica de negocio y la presentación.

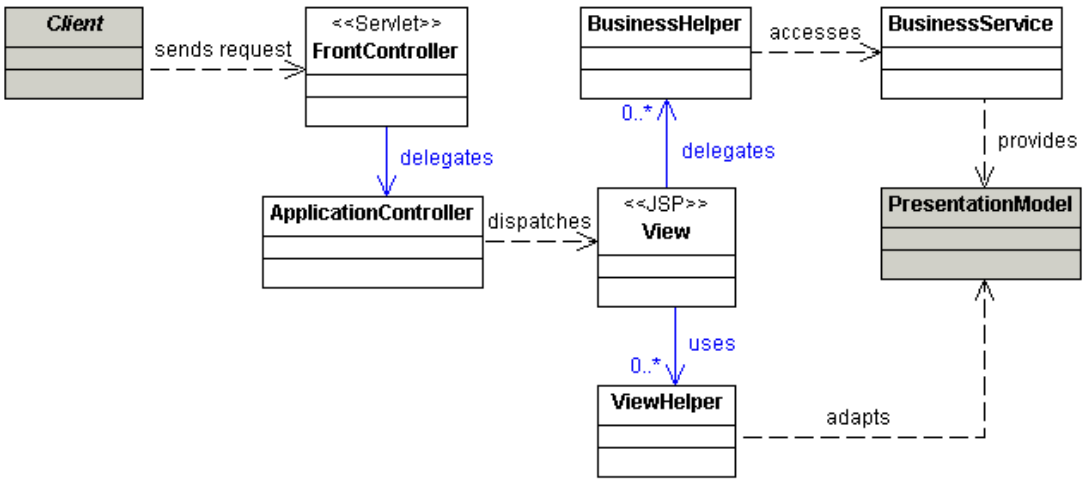


Ilustración 15: Patrón Dispatcher View

2.15. Service to Worker

Combinación de varios de los anteriores patrones (Front Controller, Application Controller, View Helper y Dispatcher View) para centralizar el control y la gestión de las peticiones, para gestionar la lógica de negocio y obtener una respuesta antes de entregarle el control a la vista. Básicamente, la vista a mostrar depende de la lógica de negocio, y por eso se requiere ejecutar dicha lógica antes.

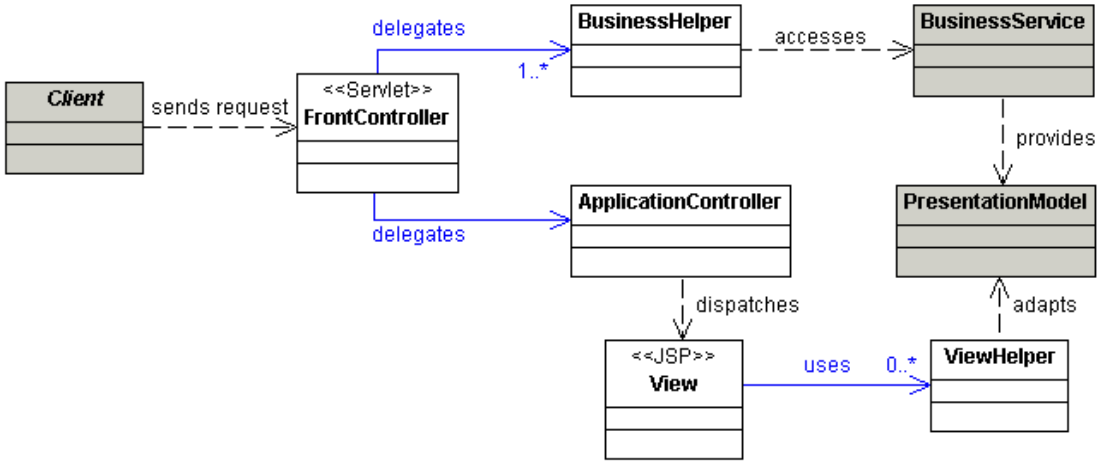


Ilustración 16: Patrón Service to Worker

3. Estudio de Frameworks

3.1. Introducción

Un framework, o marco de trabajo, es una plataforma de software para el desarrollo de aplicaciones. De forma más técnica, es una capa software de abstracción que provee al programador de un conjunto de funcionalidades, genéricas y adaptables, que puede utilizar para crear aplicaciones.

De forma análoga a las bibliotecas de software, diferentes frameworks existen para cubrir diferentes necesidades o problemas. En contraposición a una biblioteca, un framework ofrece un conjunto mucho más variado de software, entre los que se encuentran compiladores, APIs, bibliotecas y herramientas diversas que dan apoyo al desarrollo de aplicaciones.

Utilizar un framework significa adaptarse a su funcionalidad, lo cual requiere un periodo de aprendizaje. Sin embargo, la abstracción y funcionalidad que ofrece compensa con creces el tiempo dedicado a conocerlo. Una vez superada esa fase, el programador puede olvidarse de aspectos programáticos de bajo nivel y centrarse en la lógica de su aplicación.

En el caso de los frameworks de presentación, por ejemplo, el marco de trabajo se encargará de la gestión de la tecnología HTML, de la gestión del Servlet, del control de la navegación, de la generación de las vistas, etc. Son una serie de características muy interesantes que convierten la labor de crear aplicaciones web en un proceso más rápido, más productivo y menos tedioso, al poderse centrar el programador en definir e implementar la lógica del negocio.

Es precisamente este aporte el que convierte a los frameworks de aplicación en una herramienta útil e interesante, y merecedora de un estudio a fondo. Con esta investigación se pretende observar los diferentes paradigmas y soluciones que aportan los frameworks existentes, con sus virtudes y defectos. Dadas las limitaciones temporales del presente trabajo, y la enorme oferta de frameworks de presentación en el mercado, es preciso obtener un criterio de calidad para elegir una muestra significativa con la que realizar el análisis.

En el mercado de las aplicaciones Java empresariales existe una oferta de frameworks muy variada, con filosofías y enfoques diferentes. Al ser un mundo en constante evolución, resulta difícil saber cuáles son los mejores frameworks, sobre todo cuando se pueden encontrar opiniones muy diversas y enfrentadas sobre los criterios de calidad existentes.

Aunque no es, ni mucho menos, un criterio objetivo, el nivel de uso y la popularidad pueden ser criterios perfectamente válidos a la hora de considerar un framework como exitoso, y también por tanto, como criterios de selección para realizar un estudio.

- Se ha decidido empezar el estudio por un framework, **Struts**, ya abandonado, pero que ha sido muy utilizado y ha tenido una gran influencia en el desarrollo de frameworks más modernos.

- **Struts**[2] se ha elegido por ser uno de los más populares y utilizados, y tal vez el más representativo del grupo de frameworks que priman la configuración sobre la convención.

- **Spring** es un framework de pila completa, que no solo da apoyo a la programación de la presentación, sino también a las otras capas de la aplicación. Es un framework muy utilizado, sobre todo por su capa de negocio, pero la completa integración y su excelente motor de inversión de control le han convertido en una popular alternativa a los demás frameworks de presentación.

- **Tapestry** presenta un modo de funcionamiento orientado a componentes y eventos. Es parte de una familia de frameworks que rompe completamente con la forma de trabajo que inició Struts. Es muy productivo, y está ganando en popularidad y uso, por lo que se ha decidido incluirlo en el estudio.

A la hora de realizar la investigación, se hará especial hincapié en estudiar una serie de elementos, presentes pero diferentes en todos ellos, que ayudarán a remarcar las ventajas y desventajas que presentan:

- Arquitectura y características generales
- Ciclo de vida de una petición
- Configuración de las aplicaciones

Una vez finalizada la investigación y la obtención de los datos requeridos, se elaborará un análisis comparativo de los cuatro frameworks, cuyos resultados se utilizarán directamente en las fases posteriores del proyecto. Fundamentalmente servirá para definir la arquitectura, las características y las funcionalidades que presentará el nuevo framework que se creará.

3.2. Struts

Struts, conocido como Struts 1 en contraposición a Struts[2], es un framework para desarrollar aplicaciones web JEE. Está basado en el patrón Modelo-Vista-Controlador, y ha disfrutado de gran popularidad desde su aparición en el año 2000.

Este framework se centra en proporcionar un Controlador, y facilita el acceso a una amplia variedad de tecnologías existentes para que el programador desarrolle la Vista. Para construir el Modelo hay prácticamente total libertad, Struts no define una tecnología específica que utilizar y puede adaptarse fácilmente a otras tecnologías existentes.

3.2.1. Arquitectura y características

La arquitectura de Struts puede verse resumida en la siguiente imagen:

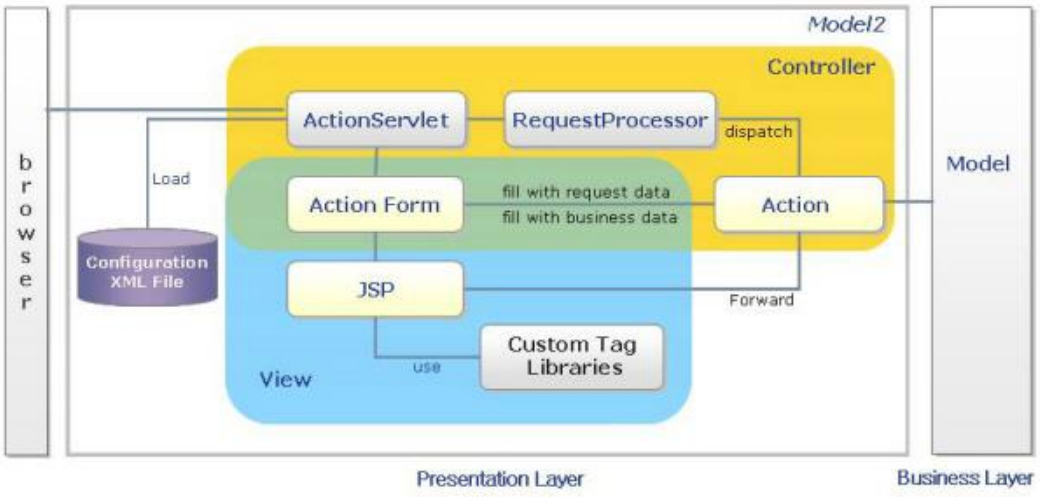


Ilustración 17: Arquitectura de Struts

- El Controlador recibe las peticiones y llama a la clase Action adecuada, en la que delega el tratamiento de la petición; esta Action es quien interacciona con el Modelo. Struts provee además los ActionForm, formularios con filtros que comprueban que los datos introducidos en su formulario son correctos antes de enviarlos a las Action.

  Este Controlador es un Servlet de tipo ActionServlet, que implementa el patrón Front Controller. Para decidir a quién reenviar las peticiones, el usuario define una serie de ActionMappings, que relacionan las diferentes URI de origen de la petición con diferentes Actions. Las Actions se encargan de enviar las peticiones a la lógica de negocio, y de entregar el control de la interacción de la interfaz a la Vista adecuada.

- La Vista habitualmente se crea utilizando la tecnología JavaServer Pages (aunque puede usarse Velocity, XSLT y otros) que permite utilizar HTML estático y una serie de etiquetas de acción a través de las cuales se puede generar contenido dinámico. El framework incluye una potente colección de etiquetas para crear interfaces, facilitando la interacción con los ActionForm y la internacionalización.

- El Modelo goza de completa libertad de programación Struts no entra en definir nada sobre él, pero proporciona facilidad de conexión con tecnologías para la capa de negocio.

Hoy en día se considera a Struts 1 como un framework anticuado; su mayor defecto reside en que las aplicaciones de usuario siguen estando fuertemente acopladas al Servlet. Dicho de otra manera, no proporciona suficiente abstracción para la programación de aplicaciones, exige al programador un conocimiento profundo de conceptos subyacentes como Servlets, peticiones HTTP, y un largo etcétera.

Otra carencia viene de las Action y ActionForms. No son clases sencillas, sino que deben heredar de la clase Action, lo cual convierte la programación con Struts en fuertemente acoplada. Además, al depender del resto del framework, hace estas clases difíciles de testar, y necesaria la utilización de objetos *mock*.

Por otro lado, las Actions implementan el patrón Singleton, por el cual solo existe una instancia de cada acción. Esto obliga a definir una seguridad especial en el acceso concurrente a dichos objetos.

Muy recientemente, el 21 de Septiembre de 2013, Struts fue oficialmente abandonado, y se recomendó a sus usuarios cambiar a Struts[2].

3.2.2. Ciclo de vida de una petición

Una petición en Struts tiene el siguiente ciclo de vida:

1. El cliente envía un formulario HTML a través del método doPost, de forma que el Controlador ActionServlet pueda tratarlo.
2. El Controlador delega el proceso de la petición en el RequestProcessor. Este mapea la petición con el ActionForm adecuado. Si es necesario, crea el ActionForm, y lo rellena con los datos de la petición. En ese momento valida los datos de la petición, y si son correctos los envía a la Action de usuario adecuada.

3. El Action obtiene los datos del ActionForm y realiza una petición a la lógica de negocio. Opcionalmente, se rellena una HelperBean, y se hace una petición a la Vista que presentará los datos.
4. La Vista recupera los datos de la HelperBean, en caso de utilizarse, y de la ActionForm, y presenta la página web.

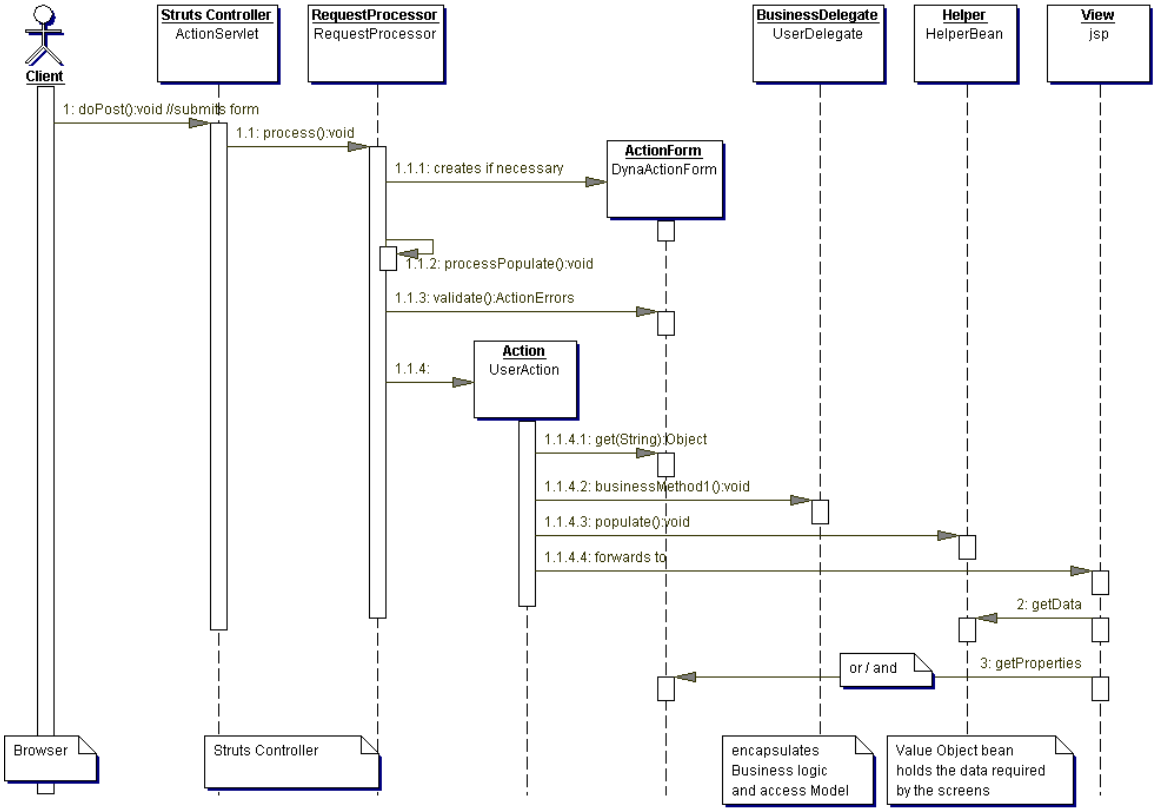


Ilustración 18: Diagrama de secuencia de una petición Struts

3.2.3. Configuración

Para poder relacionar el Modelo, la Vista y el Controlador, Struts utiliza un archivo de configuración en XML llamado struts-config.xml. Entre otras funciones, permite definir las reglas de navegación de la aplicación, es decir, cómo debe responder el sistema tanto a las peticiones web del usuario como a los resultados de la lógica del negocio. En concreto, define tres tipos de recursos:

- ActionForms: almacenan la información de las peticiones del cliente para que pueda ser usada por las diferentes clases de la lógica de la aplicación.
- ActionMappings: relacionan una petición del cliente con una Action específica.

- ActionForwards: seleccionan la página web que se utilizará para presentar el resultado de tratar una petición por la lógica del negocio.

Un archivo de configuración struts-config.xml típico mostrará la siguiente estructura:

```
<?xml version="1.0"?>
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts Configuration 1.1//EN"
        "http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd">
<struts-config>
    <!-- ========== ActionForm Definitions ================================ -->
    <form-beans>
        <form-bean name="userForm" type="org.apache.struts.action.DynaActionForm">
            <form-property name="property1" type="java.lang.String"/>
        </form-bean>
    </form-beans>

    <!--  ========== Global Forward Definitions ============================== -->
    <global-forwards type="org.apache.struts.action.ActionForward">
        <forward name="next" path="/forwardedPage.jsp" />
    </global-forwards>

    <!-- ========== Action Mapping Definitions ============================== -->
    <action-mappings type="org.apache.struts.action.ActionMapping">
        <!-- Process a user logon -->
        <action path="/user" type="org.rollerjm.presentation.UserAction" name="userForm" scope="session">
            <forward name="next" path="/forwardedPage.jsp"/>
        </action>
    </action-mappings>

    <!-- ========== Applications resources ================================== -->
    <message-resources parameter="org.rollerjm.presentation.struts.resources.ApplicationResources"/>
</struts-config>
```

Ilustración 19: Ejemplo de archivo de configuración de Struts

3.3. Struts²

A pesar de su nombre, no se trata en absoluto de una evolución de Struts, sino que procede de un framework completamente independiente inicialmente llamado WebWork 2, pero al fusionarse con la comunidad de Struts se cambió el nombre por Struts². Combina, por tanto, características de Webwork 2 y Struts 1.

Struts² implementa un framework utilizando el patrón Modelo-Vista-Controlador orientado a acciones, pero también se caracteriza por la utilización de forma generalizada de una serie de filtros. Es comúnmente aceptado que la programación para Struts² es más sencilla, rápida y sufre de menos problemas de acoplamiento que utilizando la primera versión.

3.3.1. Arquitectura y características

Struts[2] aporta una gran variedad de características muy útiles para la programación de aplicaciones web. A las ya existentes en Struts 1 hay que añadir las siguientes:

- Formularios sencillos (POJO). Ya no es necesario utilizar ActionForm, sino que puede utilizarse cualquier clase Java para guardar los datos de las peticiones.
- Actions sencillas (POJO). Puede usarse cualquier tipo de clases Java para implementar las acciones, no deben heredar de Action. Esto facilita los juegos de pruebas, puesto que no hay que usar objetos mock. Además, las Action no siguen el patrón Singleton, se instancia una por cada petición, lo que evita los problemas de accesos concurrentes.
- Juego de etiquetas mejorado, que reduce las necesidades de programación en las componentes de la interfaz de usuario.
- Compatibilidad completa con etiquetas AJAX. Para el programador, su tratamiento será el mismo que las etiquetas propias de Struts.
- Inyección de dependencias en las Actions. Puede realizarse de forma sencilla a través de su integración con Spring, aunque pueden utilizarse otros motores.

La arquitectura de Struts[2] es más compleja que la de su predecesora, e incluye un completo entramado de clases que se puede ver en la siguiente imagen:

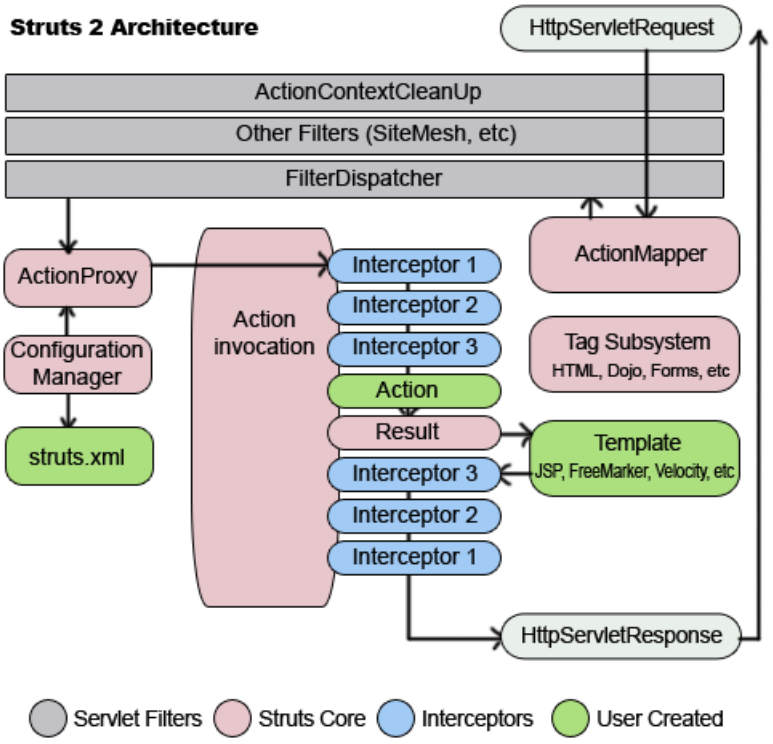


Ilustración 20: Arquitectura de Struts[2]

Como se puede observar, las clases de Struts[2] pueden englobarse en tres grandes componentes, en los que se divide también su funcionalidad:

Filtros:

Destaca especialmente FilterDispatcher, al ser fundamentalmente el Controlador de Struts[2]. Procesa las peticiones de usuario, y las mapea con una Action a realizar.

El resto de filtros opcionales pueden hacen un tratamiento previo de las peticiones, del contexto, etc., aunque su funcionamiento es muy similar a la de los interceptores y tiende a utilizarse de forma prioritaria estos últimos.

Interceptores:

Son una serie de operaciones genéricas, aplicables a un amplio rango de situaciones en una aplicación web, y que por tanto se ejecutan de forma autónoma a los Action. Los interceptores pueden ejecutarse tanto antes como después del Action, e incluso pueden denegar su ejecución si se cumplen unas condiciones determinadas.

Para cada aplicación puede utilizarse una serie de interceptores diferente, puede también modificarse su orden de ejecución, e incluso pueden crearse nuevos interceptores definidos por el programador.

Interceptores de uso muy habitual pueden ser validar los datos, subir archivos, validar el usuario (login), realizar conversiones de tipos, etcétera.

Núcleo:

Formado por las Actions y la estructura que las envuelve. En esta estructura se destaca ActionProxy, la clase en que FilterDispatcher delega la ejecución de la lógica. Consulta la configuración de la aplicación, mapea la petición con una Action, y crea una instancia de ella (una ActionInvocation), a la que entrega el control.

ActionMapper es una interfaz en la que se puede definir un mapeo entre peticiones HTTP e invocación de Actions. En su versión más básica sirve para dar más control sobre los botones de las páginas web.

Configuration Manager se encarga de almacenar en memoria la configuración de la aplicación, que el programador de la aplicación habrá consignado en el archivo de configuración struts.xml.

Las Action son la parte principal del núcleo. Recogen las peticiones de usuario y propiamente las procesan, llamando a la lógica del negocio. Como ya se ha comentado, pueden ser simples clases Java, aunque también pueden implementar la interfaz Action o extender la clase ActionSupport.

Desde el punto de vista del patrón Modelo-Vista-Controlador, se pueden englobar las funcionalidades anteriores de la siguiente manera:

Controlador:

Formado por los filtros y los interceptores, siendo el cerebro FilterDispatcher.

Modelo:

Se implementa con las Actions, clases Java normales. Estas llamarán a la lógica del negocio, y según el resultado Struts2 invocará una u otra Vista.

Vista:

Formada por un objeto Result, que se presentará habitualmente como JSP, aunque podrán utilizarse otras representaciones, como Velocity, a conveniencia.

3.3.2. Ciclo de vida de una petición

Una petición, de forma resumida, tendría el siguiente ciclo de vida:

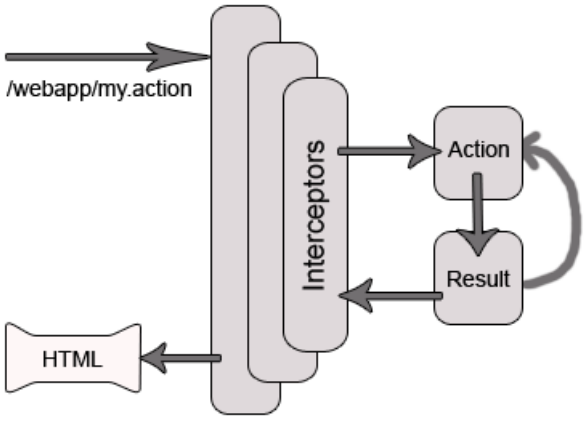


Ilustración 21: Resumen de una petición Struts2

1. La petición pasa a través de una serie de filtros. El FilterDispatcher consulta con el ActionMapper si hay que realizar alguna Action.

2. En caso afirmativo, se delega la ejecución en ActionProxy. Esta clase consultará el archivo de configuración de usuario struts.xml, crea una instancia de ActionInvocation y delega en ella el control.
3. En ese momento entran en juego los Interceptors, que realizan una serie de tareas previas a las Action, como validación de datos, gestionar el login de usuario, subida de ficheros, etc.
4. Posteriormente se invoca la Action y se ejecuta la lógica de negocio. Con ello se generará un Result, con el cual se generará la Vista, que puede ser de nuevo tratada por los interceptores. Finalmente, la Vista se entregará al usuario.

A modo de ejemplo, puede observarse el diagrama de secuencia de una petición que es objeto de validación por parte de los interceptores:

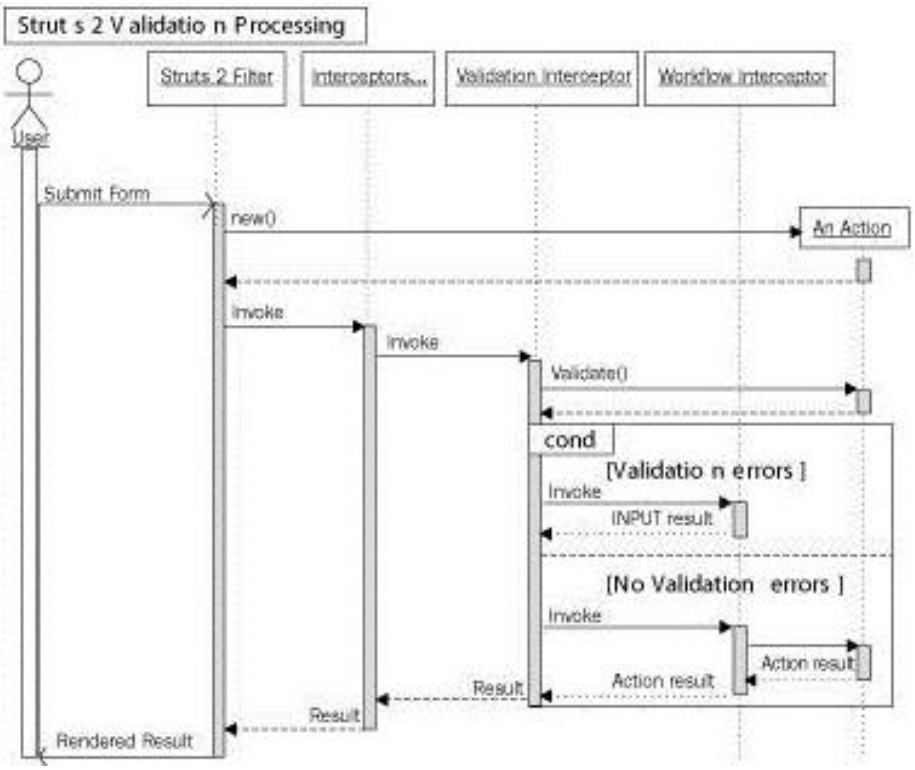


Ilustración 22: Diagrama de secuencia de una petición Struts2

3.3.3. Configuración

Para configurar las aplicaciones que usen el framework se utiliza un único archivo llamado “struts.xml”. Es muy similar al de Struts 1, y habrá que definir, en formato XML, la correspondencia ente las peticiones, las Actions a realizar, los Results a mostrar y los Interceptors que se aplicarán antes y después de gestionar las peticiones.

```
<?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.devMode" value="true" />
   <package name="default" extends="struts-default">
      <interceptors>
        <interceptor name="timer" class=".."/>
        <interceptor name="logger" class=".."/>
      </interceptors>
      <action name="login"
         class="tutorial.Login">
           <interceptor-ref name="timer"/>
           <interceptor-ref name="logger"/>
            <result name="input">login.jsp</result>
            <result name="success"
              type="redirectAction">/secure/home</result>
      </action>
  </package>
</struts>
```

Ilustración 23: Ejemplo de archivo de configuración de Struts[2]

3.4. Spring

Spring es un framework para aplicaciones Java, y también un contenedor de inversión de control. Se trata de un framework de "pila completa": no es solamente un framework de presentación de aplicaciones web JEE, sino una completa estructura que permite el desarrollo de cualquier tipo de aplicaciones Java. Sin embargo, donde mayor implantación ha cosechado es en el ámbito de la capa de sesión, al haberse convertido en una exitosa y mucho más sencilla alternativa al modelo Enterprise JavaBean de Oracle (al menos hasta la aparición de la versión 3.0 de EJB).

3.4.1. Características

Spring aporta un conjunto de características que lo convierte en una plataforma excelente para el desarrollo fácil y rápido de aplicaciones web. Entre ellas se pueden destacar:

- Una de las más importantes filosofías detrás de Spring es no ser intrusivo, permite que los usuarios puedan programar con la mayor independencia posible respecto al framework usado. Por ello, no es necesario que las aplicaciones implementen ninguna clase de Spring; dicho de otra manera, Spring apoya el desarrollo de aplicaciones empresariales utilizando clases Java sencillas, los llamados POJOs.

- Otro concepto muy importante tras Spring es la inversión de control, o inyección de dependencias. Este paradigma de la computación aboga por que el control y creación de las instancias no se definan en las clases, sino

que sean proporcionadas y controladas por una entidad externa más abstracta (en este caso, el contenedor). De esta forma se consigue más independencia entre las clases. El beneficio inmediato es que, al ser definida la dependencia por el controlador, puede ser intercambiada en diferentes implementaciones sin tener que modificar las clases, lo que aumenta la reutilización.

- Spring permite también la programación orientada a aspectos a través de su framework. Este paradigma trata de separar las funcionalidades que se repiten varias veces a lo largo de las aplicaciones en módulos propios e independientes, de forma que se eliminen la dispersión del código y las dependencias entre módulos. Un ejemplo clásico de este tipo de tareas repetitivas y cruzadas es el login.

- Por último, Spring sigue el paradigma de convención sobre configuración, tratando de minimizar tanto los archivos de configuración como el número de decisiones que el programador debe tomar. Solo deben especificarse aquellos aspectos que apartan de la convención. Por ejemplo, si en el modelo hay una clase llamada “Coches”, estará mapeada a una tabla llamada “Coches” en la base de datos.

3.4.2. Arquitectura

Spring ofrece un completo juego de módulos, frecuentemente utilizados de forma conjunta para fabricar aplicaciones web completas, pero pudiendo también utilizarse tan solo la parte que el programador necesite en cada ocasión. En la siguiente imagen pueden verse los diferentes módulos de Spring englobados en diferentes grupos:

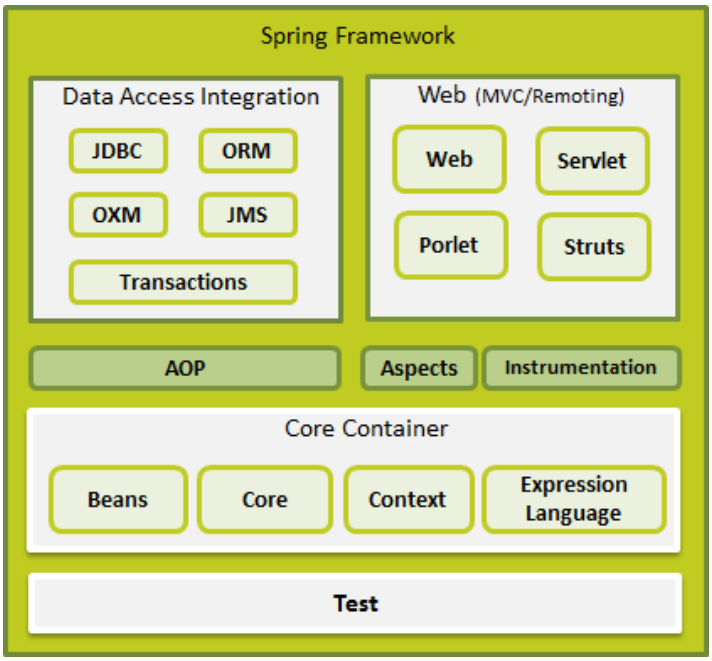


Ilustración 24: Módulos de Spring

Aunque la parte que más interesa en este estudio es el grupo de módulos web, se enumerarán las capacidades de los demás módulos, al tener relación directa.

Contenedor del núcleo

Está compuesto por los módulos Beans, Core, Context y Expression Language.

- Core es el núcleo del framework: se ocupa de su funcionamiento general. Entre otras tareas, controla la inversión de control y la inyección de dependencias.
- Bean es el módulo se encarga de la creación y gestión de objetos (*beans*). Contiene BeanFactory, un contenedor que se encarga de instanciar, configurar y coordinar los diferentes *beans*.
- Context permite acceder a mensajes, a recursos (tales como URLs o archivos), también facilita la propagación de eventos y la carga de múltiples contextos. Su núcleo es la clase ApplicationContext.
- Expression Language proporciona un lenguaje (SpEL) que facilita la consulta y manipulación de objetos en tiempo de ejecución, junto a otras tareas como la invocación de métodos.

Acceso e integración de datos

Compuesto por los módulos JDBC, ORM, OXM, JMS y Transaction.

- El módulo JDBC ofrece una capa de abstracción sobre JDBC, de forma que facilita el acceso a esta API y reduce el trabajo al programador de aplicaciones con Spring.
- ORM proporciona capas de integración para diferentes APIs de mapeo objeto-relacional, como Hibernate o iBatis.
- OXM aporta otra capa de abstracción, en este caso facilitando implementaciones de mapeo objeto-XML para diferentes tecnologías, como JAXB, XMLBeans, o XStream.
- JMS es un servicio de mensajería Java que permite la creación, envío y recepción de mensajes entre componentes de una aplicación.
- El módulo Transaction facilita la gestión de transacciones para los objetos de las aplicaciones.

Web

Compuesto por los módulos Web, Web-Servlet, Web-Struts y Web-Portlet.

- El módulo Web consiste en una serie de tareas trasversales y genéricas, de las cuales pueden aprovecharse un gran número de aplicaciones, como por ejemplo subida de archivos. Además, inicializa el contenedor de inversión de control.
- Web-Servlet es el modulo que contiene el framework Modelo-Vista-Controlador, que se verá más en detalle.
- Web-Struts es un módulo caído en desuso, que permite la integración de una aplicación Struts en Spring.
- Web-Portlet es la versión del framework Modelo-Vista-Controlador para portlets.

Otros

En esta categoría se reúnen todos aquellos módulos que no tienen un ámbito común: AOP, Aspects, Instrumentation, y Test.

- AOP es el módulo que provee la implementación de la programación orientada a aspectos. Facilita el desacoplamiento de los módulos y la separación de las funcionalidades transversales.
- Aspects permite la integración con AspectJ, un framework consolidado de programación orientada al aspecto.
- El módulo Instrumentation permite monitorizar las clases y facilita una implementación para cargar clases.
- Test facilita la realización de pruebas con JUnit o TestNG.

3.4.3. Spring Web MVC

La arquitectura de Spring MVC puede verse resumida en la siguiente imagen:

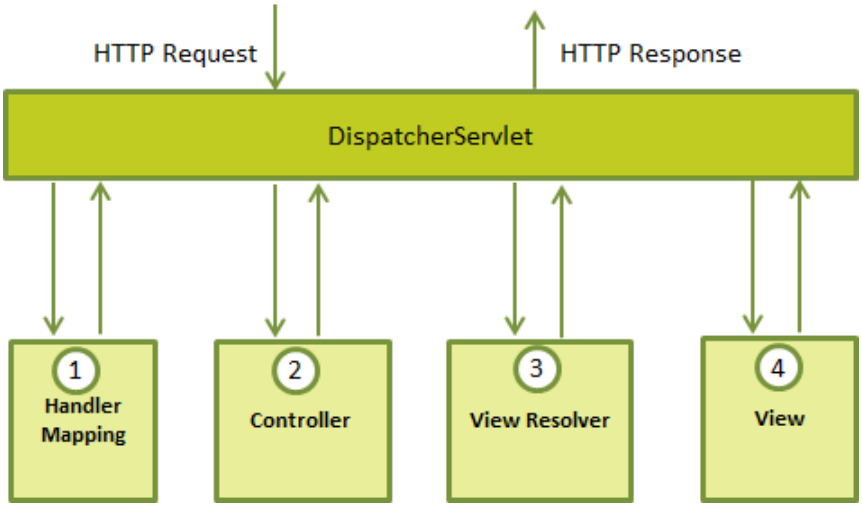


Ilustración 25: Arquitectura Spring MVC

Controlador

- El núcleo del controlador se llama DispatcherServlet. Es un Servlet que sigue el patrón Front Controller, por lo que se encarga de recibir todas las peticiones de cliente y de decidir quién debe gestionarlas. Al estar completamente integrado con el contenedor de inversión de control, puede además utilizar el resto de ventajas de Spring.
- HandlerMapping se encarga de buscar entre los Controller aquel que ha de recibir el control del flujo ante una determinada petición web. Al contrario que en Struts, este mapeo no se basa en archivos de configuración, sino en una serie de convenciones, además de anotaciones que se realizan en los Controller.

- Controller es un objeto que realiza una acción sobre una petición, a la manera de los Action de Struts Estos controladores son quienes se encargan de comunicarse con la lógica de negocio, con el modelo. Son objetos Java normales en los que se marcan unas anotaciones para poder mapear las peticiones.

- Spring también dispone de interceptores, a la manera de Struts[2]. De forma predefinida habrá algunos activados, pero pueden tanto anularse, como cambiar el orden de ejecución, como programarse otros nuevos.

Vista

- Spring encapsula las vistas como *beans* especiales. Estos *beans* almacenan la información de la vista, y a través de un ViewResolver se seleccionará y generará una presentación específica para mostrarlos.

- ViewResolver es una interfaz, de la cual Spring aporta varias implementaciones diferentes, aunque se pueden crear nuevas a discreción. Su función es ayudar a DispatcherServlet a identificar qué vistas deben mostrarse como respuesta a una petición, mapean el nombre de una vista con la vista real. Además, permiten mostrar la vista utilizando varias tecnologías diferentes, como JSP, Velocity o XLST.

Modelo

- Como ya se ha comentado con anterioridad, Spring utiliza clases sencillas, objetos POJO en todos sus niveles, también en la lógica de negocio.

3.4.4. Ciclo de vida de una petición

Una petición Spring, desde el punto de vista exclusivo del framework MVC, tendría siguiente ciclo de vida:

1. Una petición entrante HTTP llega a DispatcherServlet, que consulta a HandlerMapping el responsable de gestionar la petición.

2. El gestor consistirá en un Controller, y posiblemente unos interceptores que se encarguen del pre y post procesamiento de la petición.

3. El Controller se encarga de llamar a la lógica de negocio y recibirá la vista, encapsulada en un *bean*. El Controller entonces entregará la vista y el control al *DispatcherServlet*.

4. El DispatcherServlet utilizará *ViewResolver* para seleccionar la vista que se ha de mostrar.

5. Finalmente, DispatcherServlet envía los datos del modelo a la vista, que se mostrará al cliente.

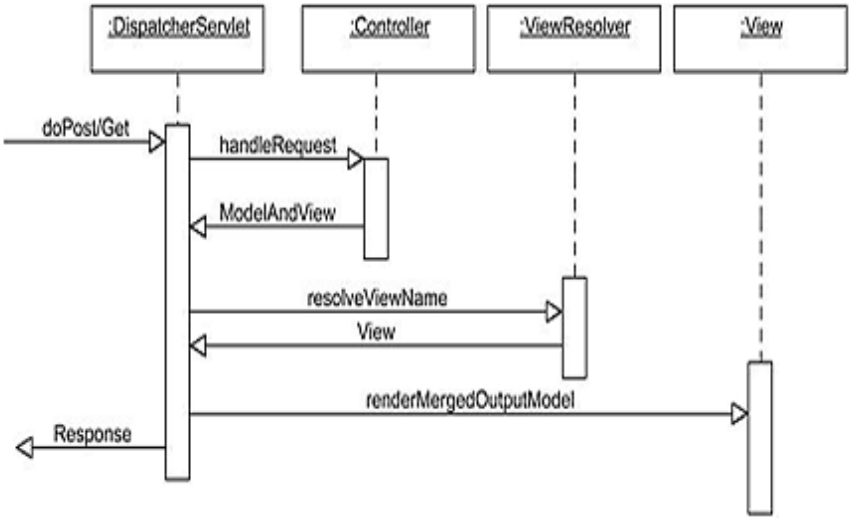


Ilustración 26: Petición Spring simplificada

3.4.5. Configuración

Como ya se ha comentado, Spring utiliza el paradigma de convención sobre configuración. El controlador DispatcherServlet se encarga de activar el módulo DefaultAnnotationHandlerMapping, que busca las anotaciones @RequestMapping en las clases que tienen la anotación @Controller. De esta manera, no es necesario mapear las peticiones web con las clases que se encarguen de tratarlas. Por ejemplo, para contestar una petición a la página "/hello" se crearía el siguiente controlador:

```
@Controller
@RequestMapping("/hello")
public class HelloController{

   @RequestMapping(method = RequestMethod.GET)
   public String printHello(ModelMap model) {
      model.addAttribute("message", "Hello Spring MVC Framework!");
      return "hello";
   }
}
```

Ilustración 27: Controlador en Spring

Opcionalmente, este comportamiento puede modificarse, para permitir modificar los valores predefinidos de las propiedades, como por ejemplo los interceptores a usar. Por ejemplo, la siguiente configuración añadiría un interceptor a la aplicación:

```
<beans>
  <bean id="handlerMapping"
        class="org.springframework.web.servlet.mvc.
               annotation.DefaultAnnotationHandlerMapping">
    <property name="interceptors">
      <bean class="example.MyInterceptor"/>
    </property>
  </bean>
<beans>
```

Ilustración 28: Configuración de un nuevo interceptor en Spring

3.5. Tapestry

Tapestry es un framework para aplicaciones JEE basado en componentes y orientados a eventos, de forma opuesta al desarrollo más clásico basado en acciones.

A la hora de construir una aplicación web, Tapestry la considera como un conjunto de páginas web. Cada una de estas páginas estará dividida en componentes, los cuales a su vez pueden estar formados por más componentes. Un componente es un binomio formado por una página web y una clase Java asociada, que tratará los eventos procedentes de esa página.

Para construir las páginas web se utilizan plantillas, un subconjunto muy sencillo de XML. Al contrario que otras tecnologías, la lógica en las plantillas de Tapestry es muy reducida, es prácticamente XHTML. De esta forma no solo se facilita la separación entre la lógica y la presentación de los datos, sino que se impone. Las plantillas se encargarán casi exclusivamente de mostrar la información, y la lógica caerá en su clase asociada. Como consecuencia, se obliga también a modularizar la aplicación en componentes sencillos, lo que la convierte en mucho más fácil de mantener.

La comunidad de Tapestry ha definido cuatro grandes principios por los que regirse en la evolución de su producto:

1. **Estructura estática, comportamiento dinámico:** los tipos de componente de cada página se definen de forma estática, no se pueden añadir nuevos en ejecución. Con esto se aseguran los principios de agilidad y escalabilidad en el desarrollo de aplicaciones, que se podrían ver comprometidos.

2. **API adaptativa:** pueden adaptarse los nombres de los métodos, sus parámetros, etc. del framework, para que se adapte al código ya creado. Esto

es una clara contraposición a adaptar el código a cada framework y a sus diferentes evoluciones.

3. **Diferenciar las APIs públicas de las privadas:** todo es privado, salvo lo que necesariamente debe ser público. Este principio fue adoptado por la falta de delineación existente hasta la versión 4 de Tapestry.

4. **Asegurar la retro compatibilidad:** una aplicación basada en Tapestry debe seguir funcionando en las nuevas versiones. De nuevo, se adoptó este principio porque Tapestry 5 no fue compatible con las versiones anteriores, lo que derivó en un gran número de críticas y protestas.

3.5.1. Características

Tapestry dispone de una alta cantidad de funcionalidades. Además de las ya vistas previamente, las citadas a continuación también merecen ser destacadas:

- El diseño basado en componentes permite que el programador se centre en desarrollar clases, métodos y propiedades, mientras que puede abstraerse de los conceptos relacionados con la web, como URLs o parámetros. Esto se consigue a través de la capa de abstracción de Tapestry, y de la relación directa que fija entre las plantillas y las clases.

- Tapestry detecta los cambios sucedidos en las clases, componentes, plantillas, etc., y los actualiza automáticamente en la versión en ejecución, sin necesidad de detener la aplicación.

- Utiliza el paradigma de convención sobre configuración, de forma que no es necesario definir archivos de configuración para las aplicaciones. En su lugar, se utiliza la convención, como los binomios plantilla-clase componente, que deben tener el mismo nombre, y las anotaciones en las clases.

- Al igual que otros frameworks como Spring, dispone de un motor de inversión de control. La creación y el control de las instancias serán gestionadas por el contenedor. De esta manera, las aplicaciones del programador serán más robustas, escalables, fáciles de mantener y de reutilizar, además de que la realización de pruebas también será más sencilla.

- Una de sus características más avanzadas es la transformación de clases. A través de la manipulación de bytecode, y utilizando anotaciones, permite que clases y componentes sencillos (POJOs) sean convertidos, en tiempo de ejecución, en objetos más complejos y con más funcionalidad.

- En las últimas versiones, Tapesty permite la utilización de múltiples lenguajes de programación que puedan ejecutarse en una máquina virtual Java, como Groovy o Scala.

- Facilidad de interconexión con otros frameworks y tecnologías. Puede utilizarse Spring para la lógica de negocio, Hibernate o JPA para la persistencia, etc.

3.5.2. Arquitectura

Tapestry sigue el patrón Modelo-Vista-Controlador, utilizando además el patrón de diseño Front Controller. Utiliza además una arquitectura basada en componentes, lo que lo diferencia en gran medida de los frameworks estudiados hasta este momento. De forma muy simplificada, la arquitectura de Tapestry responde a la siguiente imagen:

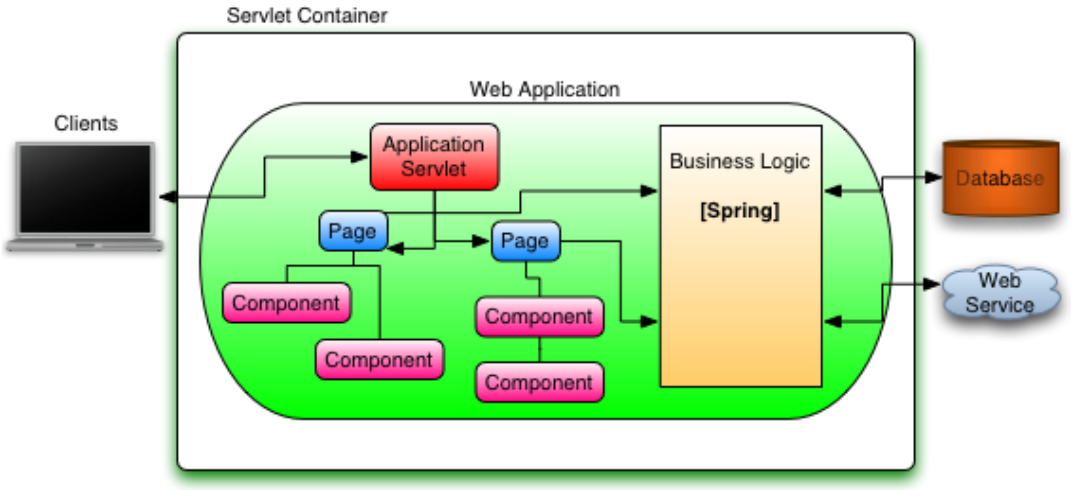


Ilustración 29: Arquitectura de Tapestry

De acuerdo al patrón MVC, se pueden separar sus funcionalidades de la siguiente manera:

Controlador

- El filtro de Tapestry actúa como controlador frontal, recibiendo todas las peticiones. De forma resumida, las peticiones disparan eventos, que serán tratados por las componentes adecuadas, tras lo cual las componentes serán renderizadas para mostrar un resultado. Como cada página puede estar formada por varios componentes, puede haber varios gestores de eventos para cada página.

- Las tuberías o cadenas de filtros son un aspecto fundamental en Tapestry. Realizan un tratamiento previo de las peticiones, y son fácilmente extensibles por el programador. Destaca especialmente la tubería RequestHandler, que comprueba si hay cambios en las clases para realizar una actualización en ejecución, detecta el idioma del usuario para presentar el idioma adecuado, etc.

- El Despachador Maestro es una cadena de Despachadores, objetos que analizan la petición y, si es de su incumbencia, la capturan y procesan. Hay dos despachadores fundamentales:

  - ComponentEvent: identifica la petición con la componente a tratarla.

  - PageRenderer: renderiza una página para mostrarla como respuesta a una petición.

- Los componentes están formados por una plantilla y una clase asociada. Esta clase es quien se encarga de realizar una acción sobre una petición, como los Action de Struts También se encargan de comunicarse con la lógica de negocio, con el modelo. Son objetos Java normales a los que acompañan una amplia gama de anotaciones; estas anotaciones pueden tener muy diversas funciones, como inyectar dependencias, pero también sirven para poder mapear las peticiones.

Vista

- Las vistas separan completamente la lógica de la presentación, y utilizan sencillas plantillas XML para definir sus componentes. Las vistas estarán formadas por una o más componentes, que serán renderizadas por el motor de Tapestry (PageRenderer).

Modelo

- Tapestry no define cómo se debe trabajar en la lógica de negocio, lo deja abierto al criterio del programador, pero facilita el acoplamiento de sus aplicaciones a entornos Spring.

3.5.3. Ciclo de vida de una petición

Una petición Tapestry, de forma resumida, sigue el siguiente ciclo de vida:

1. Llega al filtro de Tapestry, que decidirá si es una petición para el contenedor o para Tapestry. Si es para Tapestry la dirige a la tubería RequestHandler.

2. RequestHandler preprocesa la petición con una serie de filtros, y si nos es una petición a una página estática o contiene errores, la envía a MasterDispatcher.

3. La petición contiene un evento, por lo que es capturada por ComponentEvent, y enviada a su tubería ComponentRequestHandler para su tratamiento. Se identificará el componente que debe gestionar el evento, y se llamará a la lógica del negocio.

4. Como respuesta a la acción se envía una redirección de URL, de forma que se renderiza una página que muestra la respuesta al evento.

Puede verse el ciclo completo de una petición en Tapestry en la siguiente imagen:

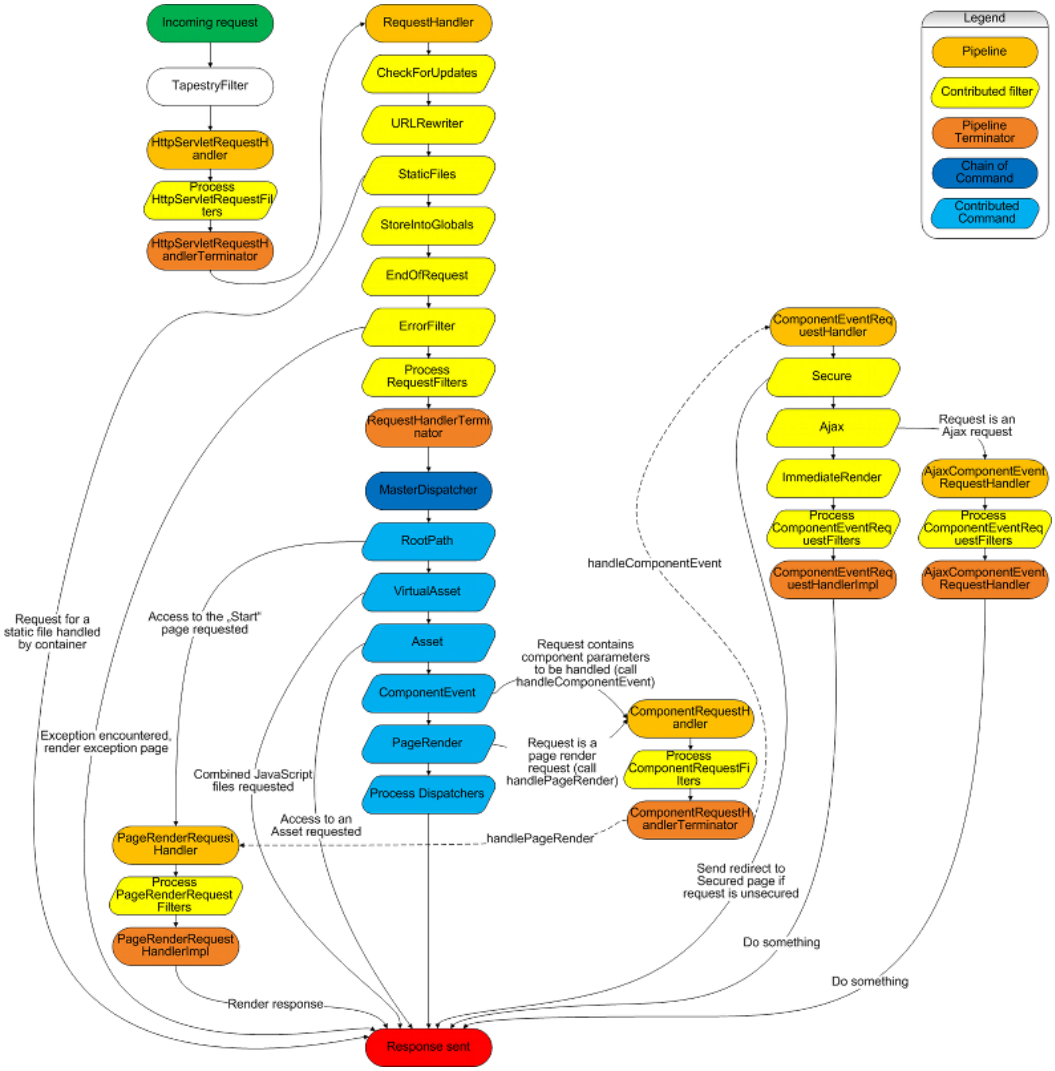


Ilustración 30: Ciclo de vida de una petición Tapestry

3.5.4. Configuración

Tapestry sigue el paradigma de convención sobre configuración, por lo que apenas es necesario definir configuraciones. En la configuración del contenedor del Servlet, Tapestry se configura como un filtro, de forma que captura todas las peticiones que van hacia Tapestry, y deja pasar las que van al contenedor (p.e. Tomcat).

Por lo demás, no existen archivos de configuración, sino plantillas con su clase asociada, que se encargará de gestionar sus eventos. Las clases tendrán mapeadas los input de las plantillas, las plantillas los getters de las clases, etc. Básicamente, cualquier parámetro de la plantilla precedido por el prefijo “t:” tendrá una equivalencia en la clase.

3.6. Comparativa de los frameworks

Tras el estudio realizado, parece indudable que la utilización de frameworks para la programación de aplicaciones web aporta una amplia gama de ventajas. Se han visto múltiples funcionalidades y modos de aproximarse a una solución, en varias ocasiones incluso han aparecido filosofías diametralmente opuestas. Para intentar sintetizar estas características, se presenta una tabla, en la que se recalcarán las diferencias existentes entre cada uno de los frameworks.

	Struts	Struts2	Spring MVC	Tapestry
Lenguajes soportados	Java	Java	Java	Java, Groovy, Scala
Orientación	Acciones y URLs	Acciones y URLs	Acciones y URLs	Componentes y eventos
Convención o configuración	Configuración	Configuración	Convención	Convención
Inversión de control	No	A través de otros motores	Motor propio	Motor propio
Acciones y formularios	Objetos complejos	POJOs	POJOs	POJOs
Facilidad de reutilización	Baja	Media	Media	Alta
Vistas	JSP, Velocity, etc.	JSP, Velocity, etc.	JSP, Velocity, etc.	Basadas en plantillas
Filtros o interceptores	No	Sí	Sí	Sí
Nivel de aceptación	En extinción	Muy alta	Muy alta	Media

Tabla 1: Comparativa de frameworks.

- La inversión de control y la inyección de dependencias son cualidades que hoy en día se exigen a cualquier framework que quiera ser tomado con seriedad. Aportan flexibilidad y permiten el uso de uso de objetos simples para definir acciones y formularios, de forma que sean fácilmente portables a otros frameworks. Adicionalmente, es mucho más sencillo implementar juegos de

pruebas para objetos POJO, puesto que se pueden realizar de forma independiente al funcionamiento del framework y el Servlet.

- Los filtros o interceptores son otra funcionalidad muy atractiva. Básicamente consisten en acciones que se pueden invocar antes o después de la propia acción asociada a la petición de usuario. Por un lado, los propios frameworks aportan un interesante abanico de filtros, como la validación de datos, la identificación de usuarios, la carga y descarga de archivos, etc. Por otro lado, se permite al programador definir y programar sus propios filtros, aspecto que abunda en la posibilidad de reutilización del código.

- Con respecto a cómo definir el funcionamiento de la aplicación, aunque Struts[2] siga apostando por un archivo de configuración donde se anoten las relaciones entre acciones, formularios y vistas, el camino tomado por otras soluciones más modernas es la convención. Se trata, pues, de evitar complejos archivos que no cumplen mayor estándar que el definido por el propio framework, y que exigen un periodo adicional de aprendizaje.

  La convención sobre configuración es, sin duda, una aproximación más natural. Funciona perfectamente cuando el programador no se sale de la propia convención, y acciones, formularios, vistas, componentes y eventos comparten el mismo nombre, pero no está exenta de problemas. Cuando el programador debe declarar aspectos no convencionales debe utilizar o bien archivos de configuración o recurrir al uso de anotaciones en las clases. Las anotaciones sufren el mismo problema que los archivos de configuración, pues no están estandarizadas y varían de un framework a otro. Esto produce una curva de aprendizaje, además de que puede volver farragoso el código fuente.

  De cualquier manera, aunque hoy por hoy haya que seguir especificando ciertos aspectos del funcionamiento de la aplicación, el uso de convención facilita en gran manera la tarea del programador, por lo que es una característica que aporta calidad a un framework.

- Con respecto al aspecto dinámico de las vistas o páginas web, tres de los cuatro frameworks vistos (y la mayoría de los existentes en el mercado Java) confía en el uso de tecnologías de plantillas, como JSP, Facelets o Velocity. Aunque es una tecnología ampliamente usada y aceptada, implica varias dificultades. Por una parte, mezcla lógica con presentación, al tener código incrustado en las vistas. Por otro lado, el código de las páginas no es Java, sino algún lenguaje script o similar, para los cuales los entornos de programación no ofrecen demasiada ayuda. Además, no facilita la reutilización, y si se utiliza un mismo fragmento de código en varias páginas hay que declararlo explícitamente en cada una de ellas.

Por último, el desarrollo de juegos de pruebas para el código incrustado no resulta en absoluto fácil de programar.

Tapestry, desde otro punto de vista, confía en el uso de componentes. Estos artefactos son páginas XHTML con clases asociadas. Al contrario que con las plantillas, hay muy poco código incrustado en la página; de hecho, con los componentes Tapestry fuerza a separar la vista de la lógica. De esta forma prácticamente desaparece la problemática para realizar pruebas. Además, estos componentes se programan una vez, pero se utilizan de forma repetida en la misma o en distintas aplicaciones. De esta forma, aumenta la reutilización, pero también la productividad, pues en caso de querer modificar un fragmento de código solo habrá que hacerlo en la componente adecuada.

- El aspecto de calidad más discutible consiste en la elección de la orientación del framework. La utilización de acciones y URLs es un clásico, funciona bien, tiene una amplia aceptación y sigue de forma fiel el paradigma de la arquitectura web, basada en URLs. Por otro lado, la orientación a componentes y eventos presenta una capa de abstracción que permite al programador olvidarse de URLs y centrarse en la programación. Se trata, pues, de una mayor aproximación hacia las aplicaciones de escritorio.

- Decidir qué framework entre los estudiados es mejor sería un planteamiento demasiado atrevido. Aunque las características técnicas son un aspecto ponderable, no lo es menos la experiencia de usuario, y esta es una perspectiva que este estudio no puede ofrecer. Con los datos estudiados, no obstante, es fácil ver que Struts 1 es un framework que se ha quedado anticuado. En la actualidad se demandan unas funcionalidades superiores, a la vez que una mayor facilidad de programación y portabilidad. Struts[2], aunque muy utilizado, ofrece menos funcionalidad que los otros dos frameworks. Está orientado a configuración y ni siquiera tiene su propio motor de inyección de dependencias.

  Tapestry es la oferta más novedosa, y su orientación a eventos y componentes resulta más sencilla que las tradicionales URLs. Su capacidad de manipular y reescribir bytecode, a través de anotaciones, hace que sencillas clases POJO se conviertan en potentes clases en tiempo de ejecución. Posiblemente es el framework que permite un desarrollo de aplicaciones más sencillo y reutilizable.

  Spring, aunque más tradicional que Tapestry, ofrece un marco estable, potente y muy considerado entre la comunidad de programadores web. Parte de este éxito se debe a que Spring ofrece un framework de calidad de pila completa, pero suficientemente flexible como para poder utilizar otros frameworks en la capa que el programador elija. Un factor que confirma su calidad es el hecho de que

otros frameworks ofrecen integración con Spring, como Struts[2] con su motor de inversión, o Tapestry, al permitir inyectar beans de Spring en su lógica.

- A modo de resumen, se ha elaborado un cuadro en el cual se enumera una serie de ventajas e inconvenientes de cada uno de los frameworks. Cabe resaltar que, según el punto de vista, determinadas ventajas pueden parecer inconvenientes, y a la inversa. Para elaborar la tabla se ha tratado de adoptar el punto de vista de un programador de aplicaciones web con poca experiencia en el uso de frameworks MVC.

	Ventajas	Inconvenientes
Struts	• Bibliografía muy extensa. • Utiliza una tecnología muy consolidada.	• Anticuado. • No usa POJOs. • Mucha configuración. • No dispone de interceptores. • No facilita la inversión de control
Struts[2]	• Mucha bibliografía. • Muy extendido. • Fácil integración con otros frameworks. • Fácil testeo.	• Usa archivos de configuración. • IoC no detecta problemas en tiempo de compilación.
Spring MVC	• Framework de pila completa. • MVC sin configuración. • POJOs. • Facilidad de testeo.	• Problemas derivados de IoC. • JSP implica programar, a pesar de los tags
Tapestry	• Componentes y vistas sencillos y modulares. • Sin configuración. • Facilidad de testeo. • Fácil de extender. • Más natural y productivo de programar una vez se aprende.	• Escasa bibliografía. • Requiere aprendizaje previo por el cambio de filosofía.

Tabla 2: Ventajas e inconvenientes de los frameworks

4. El Framework Myst

Tal como se propuso en el enunciado, tras realizar el estudio comparativo se ha de realizar un framework JEE de presentación. Myst se basa, fundamentalmente, en el patrón Modelo-Vista-Controlador. Por ello, deberá facilitar a las aplicaciones que lo utilicen la separación entre la lógica del negocio, la presentación de los datos y la gestión de las peticiones. Al ser un framework de presentación no entrará a definir el Modelo, por lo que se delega en el programador de aplicaciones la decisión de utilizar los frameworks de negocio y persistencia que prefiera.

Antes de decidir las funcionalidades que implementará el framework, es preciso tomar dos decisiones con respecto a la filosofía del framework.

- Como se ha visto previamente, hay dos aproximaciones predominantes para el tratamiento de las peticiones, orientar el framework a las acciones, u orientarlo a componentes y eventos. Aunque, sin duda, la orientación a componentes aporta un gran número de ventajas, se ha decidido orientarlo a acciones, sobre todo por la mayor bibliografía disponible. Esta decisión tiene su consecuencia inmediata en el diseño de las aplicaciones usuarias del framework: deberán implementar una acción para cada diferente petición, y habitualmente también una vista.

  La orientación a acciones también aporta considerables ventajas, entre las que se pueden destacar:

  - Las aplicaciones que utilicen el framework mostrarán una clara separación entre la vista y el modelo, por lo que facilitará en gran manera la programación.
  - Las acciones y vistas se controlarán desde un módulo centralizado, por lo que se podrá modificar su comportamiento sin cambiar la lógica de la aplicación.
  - Sus componentes (acciones, vistas, *beans* de formulario) serán fáciles de extender y reutilizar.

- La otra decisión con respecto a la filosofía del framework es optar por convención o configuración. Finalmente, dada la abundante bibliografía disponible y el límite de tiempo para implementarlo, se ha elegido la configuración, por lo que cada aplicación vendrá acompañada de un fichero donde se especifiquen las acciones con sus formularios, vistas, etc. asociados.

La elección de un modelo basado en la configuración también aporta ciertas ventajas:

- Facilidad de descripción de la navegabilidad a través de reglas. Puede incluso modificarse el funcionamiento de un programa sin modificar el código fuente, solo es necesario realizar pequeñas modificaciones en la configuración.

- Evita la necesidad de memorizar reglas de programación (convenciones) específicas del framework, que pueden variar en gran manera al cambiar de un framework a otro.

4.1. Análisis

El estudio realizado de varios frameworks de presentación ha sido provechoso para revelar qué funcionalidades debe implementar nuestro framework, y qué patrones son adecuados para modelarlo.

4.1.1. Juego de funcionalidades

Entre las funcionalidades extraídas, se ha decidido optar por un juego similar al proporcionado por Struts[2].

- **Control declarativo del flujo de la aplicación**

  La secuencia de peticiones, acciones y vistas se mantendrá de forma independiente al código, dentro del archivo de configuración de la aplicación.

- **Inversión de control**

  De forma similar a Spring, el control y creación de las instancias no se definirán en las clases interesadas, sino que serán proporcionadas y controladas por el controlador de aplicaciones. Para conseguirlo la aplicación dispondrá de un motor de inyección de dependencias, que será el encargado de crear las instancias y asociarlas a las clases que las utilicen.

- **Uso de objetos Java sencillos**

  Las acciones, formularios y filtros serán construidos como objetos POJO. Por ello tendrán un alto grado de reutilización y resultarán sencillos de probar, al poderse realizar los juegos de pruebas de forma independiente al

framework. Por supuesto esta característica redundará en aplicaciones menos acopladas al framework, y por tanto más fáciles de portar.

- **Validación y conversión automática de los datos de entrada**

  Se facilitará, de forma previa a la gestión de las peticiones, el tratamiento, conversión o validación de los datos. También se permitirá, a través de los filtros, que el programador de aplicaciones realice las validaciones adicionales que considere necesarias en cada caso. Si durante cualquier validación se produce algún error se informará al usuario a través de la gestión de errores.

- **Control de las vistas a través de un juego de etiquetas**

  Se proporcionará un juego de etiquetas para mostrar y capturar datos, de forma que se evite, en la medida de lo posible, el uso de programación dentro de las vistas. Para la creación de las vistas se utilizará la tecnología JSP.

- **Procesamiento previo y posterior a la acción**

  Se permitirá la ejecución de filtros de forma previa y posterior a la acción. El gestor de filtros decidirá el orden en que se aplican y también cuándo se ejecuta la acción. También podrá decidir que la acción no llegue a ejecutarse.

- **Internacionalización**

  Las aplicaciones que utilicen Myst podrán fácilmente definir varios idiomas en que mostrarse al usuario. La forma de realizarlo será a través de la parametrización, esto es, creando parámetros para cada objeto de texto que se muestre en las páginas, que se guardarán en diferentes archivos de idiomas. Según el idioma que elija el usuario, se mostrarán las páginas utilizando una u otra configuración.

- **Gestión de errores**

  Se capturarán, en la medida de lo posible, los posibles errores sucedidos en el uso o funcionamiento del framework. De esta manera, el funcionamiento de las aplicaciones no se verá interrumpido, y se mostrarán los errores al usuario en un formato comprensible, tanto en una vista como en log.

Será de especial importancia capturar y mostrar los errores cometidos en la programación de aplicaciones.

- **Extensibilidad**

  Permitirá extender las funcionalidades proporcionadas, por ejemplo, con acciones nuevas, nuevos tipos de resultados (o vistas), nuevos gestores de errores, etc.

4.1.2. Patrones implementados

Además del ya comentado patrón MVC, Myst utiliza otra serie de patrones que se han revelado de utilidad para la creación de frameworks de presentación.

Patrón Context Object

Cada vez que se reciba una petición, se creará un contexto. Para que sea accesible fácilmente por el framework completo se encapsulará en un *bean*, haciendo transparentes a las clases y métodos la problemática del protocolo http.

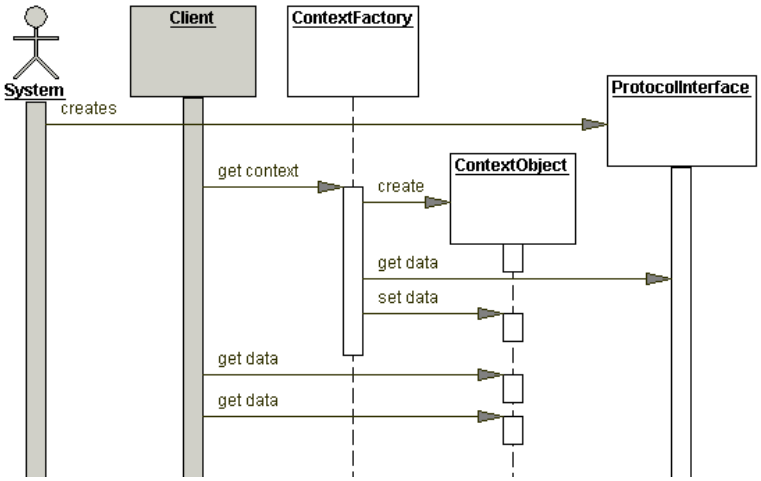


Ilustración 31: Context Object - Diagrama de secuencias

Patrón Service to Worker

Como ya se ha visto con anterioridad, fusiona los patrones Front Controller, Application Controller, Dispatcher View y View Helper para aportar una solución prácticamente global.

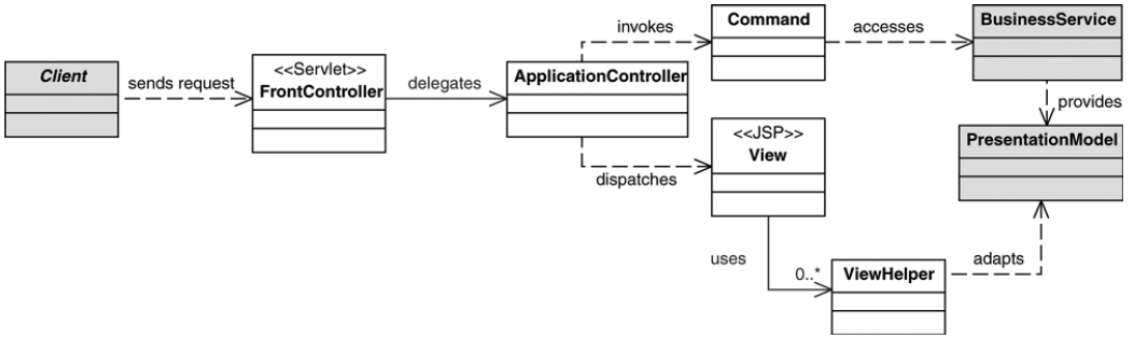


Ilustración 32: Service to Worker- Diagrama de clases

Front Controller será el punto único de entrada al framework, y se encargará de gestionar las peticiones. Entre las diferentes opciones de implementación estudiadas (Servlet, filtro) se ha decidido implementar el controlador como un Servlet, por lo que se configurará en el archivo web.xml del contenedor de Servlets. Al cargar el Servlet en el contenedor, se cargará también la configuración de la aplicación en una clase auxiliar.

El Servlet, pues, recibirá las peticiones del cliente y usará el patrón Context Object, como se ha visto antes, para crear un objeto donde almacenar el contexto de la petición. Delegará el tratamiento de la petición en Application Controller.

Application Controller recibirá el contexto de la petición y la configuración de la aplicación, donde dispondrá de los mapeado entre peticiones, comandos y vistas. Actuará como un despachador, recibiendo peticiones e invocando al comando adecuado para que la trate. En base a la respuesta recibida entregará el control a la vista adecuada.

El Command recibirá la invocación del Application Controller, con lo que llamará a la lógica de negocio. Almacenará los datos obtenidos en un objeto auxiliar (View Helper), que pondrá a disponibilidad de la vista a través del contexto de la petición.

View será llamada por Application Controller, y con los datos disponibles en View Helper poblará la página JSP que se enviará como respuesta.

Para ver su funcionamiento con mayor claridad, mostraremos su diagrama de secuencias:

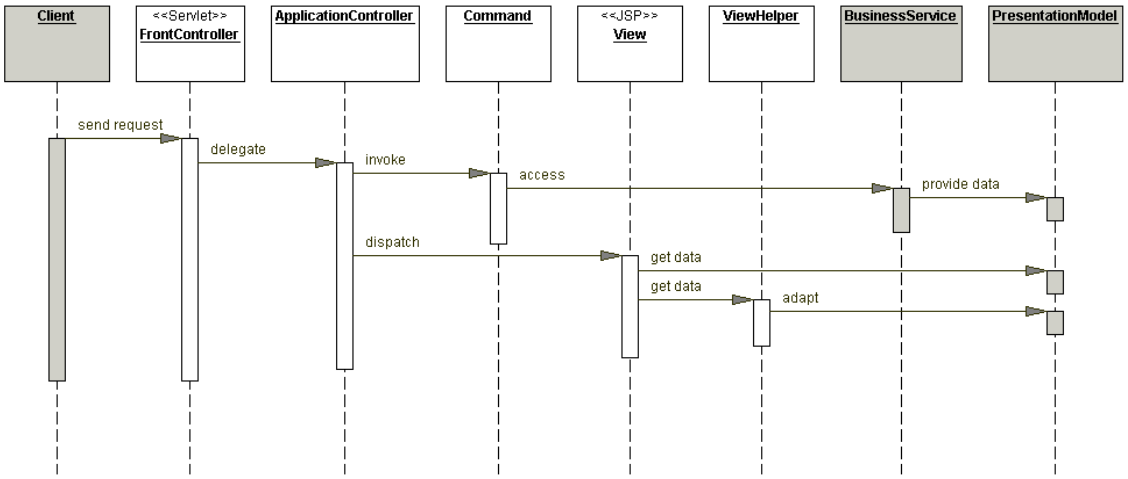


Ilustración 33: Service to Worker - Diagrama de secuencias

Patrón Singleton

Application Controller solo se instanciará una vez al cargar el Servlet, y se utilizará esa única instancia para todas las peticiones.

Patrón Command

Los diferentes comandos extenderán un Command genérico, de forma que el mapeo entre peticiones y comandos sea similar para todas los Command. Se heredarán métodos abstractos que serán invocados por el Application Controller.

Patrón Intercepting Filter

Los comandos podrán ser ampliados por un tratamiento previo y/o posterior, gracias al uso de filtros.

Patrón Composite View

Se utilizará una tecnología de vistas que permita la creación de vistas complejas utilizando componentes más sencillos.

4.2. Diseño

Tras las conclusiones obtenidas en la fase de análisis, y después de las nuevas funcionalidades descubiertas durante la fase de desarrollo, se propone el siguiente diseño para implementar el framework Myst. Como introducción se presentará el diagrama de clases, después se seguirá con una explicación detallada de las clases más

importantes que lo componen, y finalmente se mostrarán diagramas de secuencias de los dos procesos básicos del framework.

Como puede comprobarse, los nombres de las clases intentarán ser lo más parecido posible a los proporcionados por los patrones, de forma que su identificación resulte más clara. No se busca la creatividad a la hora de nombrar las clases, sino una mayor inteligibilidad.

4.2.1. Diagrama de clases

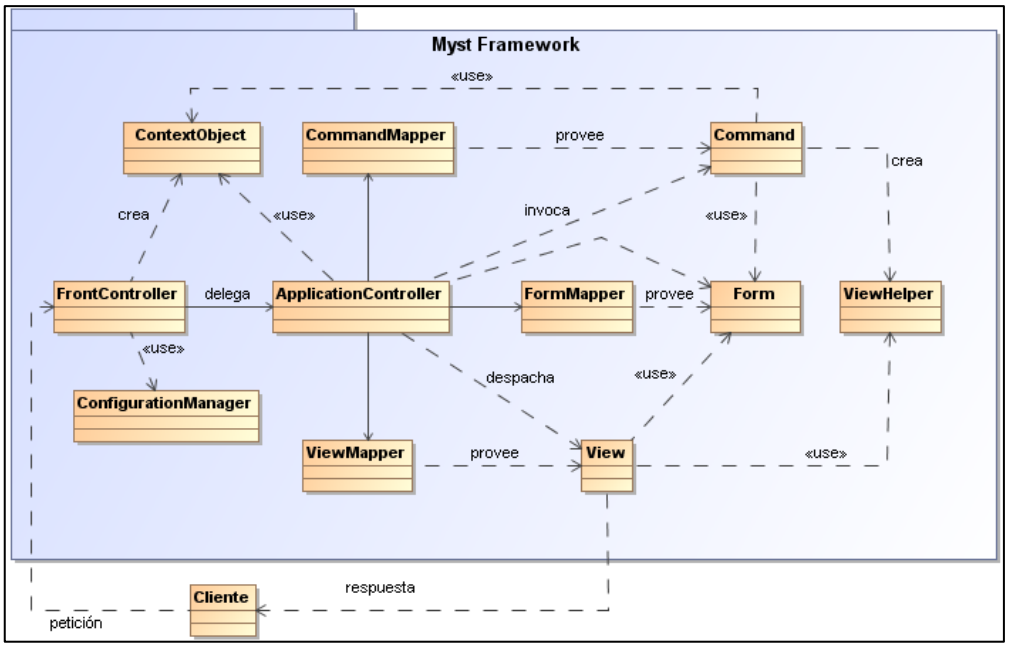


Ilustración 34: Diagrama de clases de Myst

El diagrama muestra el framework, que estará englobado dentro de un solo componente, de forma que sea accesible a las aplicaciones que lo utilicen como una biblioteca Java. El diagrama simplifica parte de la relación entre las clases por motivo, principalmente, de claridad visual:

- No muestra las clases externas y bibliotecas utilizadas por el framework.
- Oculta la lógica tras la lectura de la configuración inicial por ConfigurationManager, y su instanciación de los diferentes objetos de mapeo (CommandMapper, FormMapper y ViewMapper).
- No muestra las clases de error, ni las gestoras de errores, ni las funcionalidades no básicas disponibles en el framework, como los filtros o la internacionalización.

- No muestra la lógica de las vistas, que podrán utilizar varias bibliotecas de Tags, entre ellas la proporcionada por el framework.
- Prácticamente todas las clases hacen uso del objeto ContextObject que se crea con cada petición.

4.2.2. Descripción de las clases

Las clases descritas a continuación formarán el núcleo del framework. Le darán la funcionalidad básica y permitirá a cualquier programador utilizarlas para crear sus propias aplicaciones web.

FrontController

Es el controlador del framework, único punto de entrada para todas las aplicaciones que lo utilicen. Sus funciones son, durante la carga del Servlet, iniciar la carga de la configuración e instanciar el controlador de aplicaciones (ApplicationController).

Además, gestiona todas las peticiones de los clientes. Para ello genera una configuración del contexto (ContextObject) donde guarda los datos de la petición, que envía al controlador de aplicaciones para delegar el tratamiento de la misma.

Implementa el patrón FrontController.

ContextObject

Clase auxiliar que almacena los datos del contexto de la petición, de forma que las demás clases del framework (y las aplicaciones que lo utilicen) puedan acceder a los datos de la petición de forma transparente a la complejidad de HTTP.

Además, permite enviar y recibir información importante entre las diferentes clases que lo usan, como almacenar una referencia al ViewHelper para que esté disponible en la generación de vistas.

Implementa el patrón Context Object.

ConfigurationManager

Clase cuyos métodos se invocan una única vez, durante la carga del Servlet. Se encarga de leer el archivo de configuración de la aplicación, guardándolos en objetos de memoria. Dado que el framework utiliza un control declarativo

para el flujo de las aplicaciones, es a través de este archivo como se declara el mapeo entre peticiones, comandos y vistas. Al cargarlo en memoria se generan los objetos CommandMapper, FormMapper y ViewMapper, que el controlador de aplicaciones usará de forma asidua en su funcionamiento

También es su función la carga de las opciones definidas en archivo de configuración del contenedor de Servlets, como el nombre del archivo de configuración de la aplicación a utilizar.

ApplicationController

Es el despachador, el núcleo de la gestión de peticiones. Recibe el contexto de la petición del controlador, y a través de los mapas decide qué comando debe gestionar la petición. Instancia el comando, le envía la petición, y en función de su respuesta resuelve qué vista debe mostrarse al cliente, la cual genera.

Implementa los patrones ApplicationController y Singleton.

CommandMapper

Provee al ApplicationController de la correspondencia entre las peticiones y los comandos que las resuelven. No ofrecerá solamente el tipo de Command a utilizar, sino un objeto CommandData, con más información.

CommandData

Objeto auxiliar que almacena la clase de acción para tratar una petición, el nombre del Form a utilizar, y el ViewMapper con las posibles vistas a mostrar, en función del resultado de la petición.

Command

Clase que se encarga propiamente del tratamiento de una petición. Recibe, a través del ApplicationController, el contexto de la petición y un objeto Form, de donde obtiene los datos a tratar.

Con esta información llama a la lógica de negocio, y obtendrá un resultado, en función del cual devolverá un código u otro al ApplicationController, con lo que se decidirá la próxima vista a mostrar. Además, guardará los datos de la respuesta en un objeto ViewHelper, que se utilizará en la generación de la vista para poblarla con contenido.

Esta clase Command se ofrece como una interfaz, que el programador de la aplicación puede implementar tantas veces como diferentes comandos necesite definir. También pueden definirse commands POJO, que simplemente han de implementar los métodos necesarios para poder ser invocados por el framework.

Implementa el patrón Command.

FormMapper

En el archivo de configuración de la aplicación, cada *bean* de formulario tendrá un nombre asociado, para facilitar la labor del programador de aplicaciones. Para poder resolver la correspondencia entre el nombre del *bean* y su clase, se ofrece al ApplicationController este objeto de mapeo.

Form

Bean que almacena los datos del formulario de entrada, en un formato adecuado para su tratamiento en un entorno Java. Esta clase se ofrece como clase abstracta, que el programador de aplicaciones podrá que programar tantas veces como *beans* de formulario deba definir. También pueden definirse forms POJO, que simplemente han de implementar los métodos necesarios para poder ser invocados por el framework.

Provee métodos get y set para que el motor de inyección pueda leer o asignar sus atributos.

ViewMapper

En el archivo de configuración de la aplicación, cada petición corresponde a un Command, y a una o varias vistas, dependiendo del resultado de la petición. Por ello, dentro del objeto CommandData se incluye este ViewMapper, que permite relacionar la respuesta de la lógica de negocio con una vista. Concretamente, se relacionará la respuesta con un objeto ViewData, que almacenará más información.

Además, la aplicación también podrá definir unas vistas generales, que se aplicarán cuando el resultado de un Command no tenga mapeada ninguna vista específica.

ViewData

Objeto auxiliar que almacena el tipo de vista para mostrar una respuesta, la dirección de la URL a mostrar y posibles parámetros que el programador de aplicaciones quiera definir.

View

Clase que se encarga de generar la vista. Para ello, utiliza la información disponible en ContextObject y la respuesta de la lógica contenida en ViewHelper.

Se ofrece como una interfaz, de la cual también se proveen dos implementaciones concretas de las operaciones clásicas en HTML de reenvío (forward) y redirección (redirect). Por supuesto, el programador de aplicaciones es libre de programar cuantas implementaciones diferentes necesite.

Implementa el patrón Command.

Las páginas de la vista utilizarán la tecnología JSP, lo cual permitirá la creación de páginas estáticas o dinámicas. Para lograr el dinamismo se utilizarán bibliotecas de etiquetas y el lenguaje de expresión propio de JSP (EL). Al estar basado en HTML, se podrá utilizar también la composición de vistas complejas en base a componentes sencillas, implementando el patrón Composite View.

ViewHelper

También llamado asistente de vistas, es un objeto auxiliar que almacena los datos provenientes de la lógica de negocio para que View pueda poblar la página web que se entrega al cliente. Esta información se guardará también en el contexto de la petición para que las páginas JSP puedan acceder a ella con comodidad a través de Tags.

Implementa el patrón ViewHelper.

ErrorMapper

Dado que las excepciones son, irónicamente, un hecho frecuente en la programación, es importante su correcta gestión. Al igual que con las vistas, habrá un ErrorMapper para cada Command, de forma que cada tipo de error o excepción pueda ser tratado por el gestor preferido. También habrá un

ErrorMapper global para la aplicación, que recoja los errores no definidos en cada Command.

ErrorData

Objeto auxiliar similar al de las vistas, que almacena el tipo de excepción, el gestor para tratarlo y una serie de parámetros, como la vista a utilizar para mostrar la posible traza de errores al usuario.

ExceptionHandler

Recibe las excepciones acontecidas y realiza con ellas el tratamiento que considere adecuado, como mostrar la información por pantalla, guardar la traza en un archivo de error, etc.

Implementa el patrón Command.

Durante la fase de implementación, y producto del desarrollo ágil, se vio la conveniencia de añadir nuevas funcionalidades, prácticas y sencillas de añadir a la estructura del framework. Son los filtros.

FilterMapper

En el archivo de configuración de la aplicación, cada Command puede tener un número ilimitado de filtros de entrada y salida, que se ejecutarán de manera previa y/o posterior al Command. El objeto FilterMapper permitirá encontrar, a través de sus nombres, los Filters que implementan estas funcionalidades.

Filter

Clase que se encarga de implementar un filtro. Un filtro es, básicamente, un Command intermedio que permite realizar acciones frecuentes antes o después del propio Command. Por tanto, generalmente no debiera acceder a la lógica del negocio.

Implementa el patrón Command.

Habrá más clases, pero al no ser centrales serán especificadas y detalladas en la implementación. Por una parte habrá clases con métodos que servirán de apoyo al funcionamiento del framework; por otra parte se facilitarán implementaciones básicas y necesarias de algunos de los componentes, como las vistas o la gestión de

errores; por último, se proveerán implementaciones de muestra que aporten funcionalidad al conjunto.

4.2.3. Diagramas de secuencias

Para facilitar la comprensión del funcionamiento del framework, se muestran a continuación los diagramas de secuencias de las dos operaciones realizables. La operación de carga del framework en el contenedor se realizará de forma única, mientras que la gestión de una petición se realizará tantas veces como peticiones reciba el framework.

Carga del framework en el contenedor de servlets

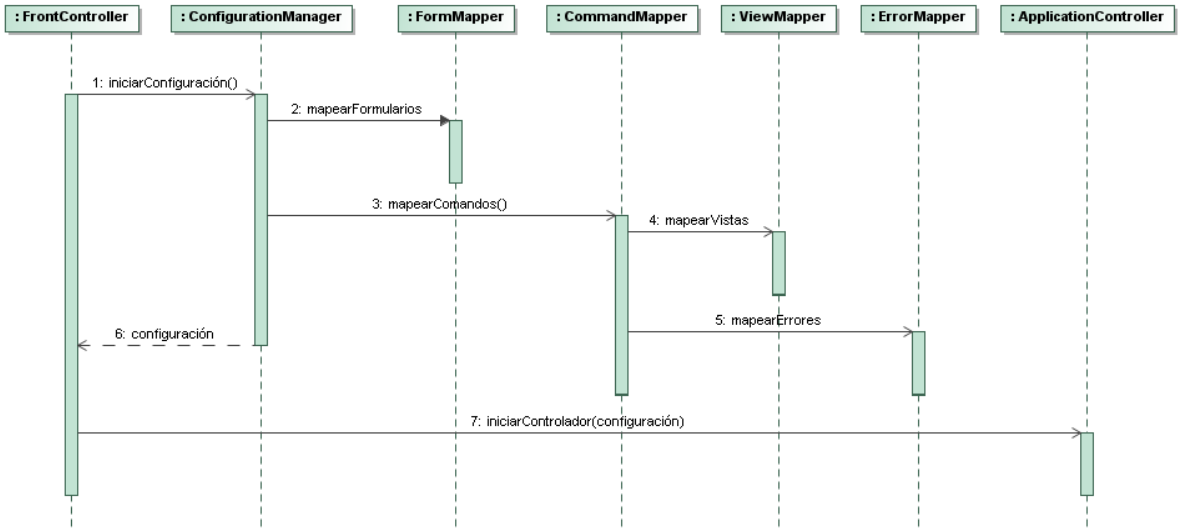


Ilustración 35: Diagrama de secuencias - Carga del framework

1. El controlador envía a ConfigurationManager la orden de que inicie la configuración de la aplicación.
2. ConfigurationManager comenzará por realizar una equivalencia entre los nombres de los Form y las clases que los implementan, y la guardará en el objeto FormMapper.
3. A continuación, creará una correspondencia entre los nombres de las peticiones y los objetos CommandData que encapsulan la clase a tratarlas, los Forms para encapsularlas, y las diferentes vistas para mostrar los resultados.

4. La correspondencia entre los resultados y las vistas se incluirá dentro del mapeo de comandos, ya que los nombres de los resultados se repetirán, y dependerán del comando que los haya generado.

5. La correspondencia entre comandos y gestores de errores también se incluirá una vez por cada comando, para permitir definir gestores diferentes a la medida de cada tipo de error y comando.

6. Una vez terminada la configuración, se entrega a FrontController.

7. FrontController inicia el despachador, ApplicationController.

En el diagrama se ha ocultado, por claridad, la creación de los mapas globales de vistas y errores, que se realizará de idéntica forma a la creación del objeto FormMapper.

También se ha evitado mostrar la creación de los mapas de los filtros. Por cada acción podrá haber un mapa de filtros de entrada y de salida, y se cargará en el punto 3, a la vez que la vistas de cada comando.

Gestión de una petición

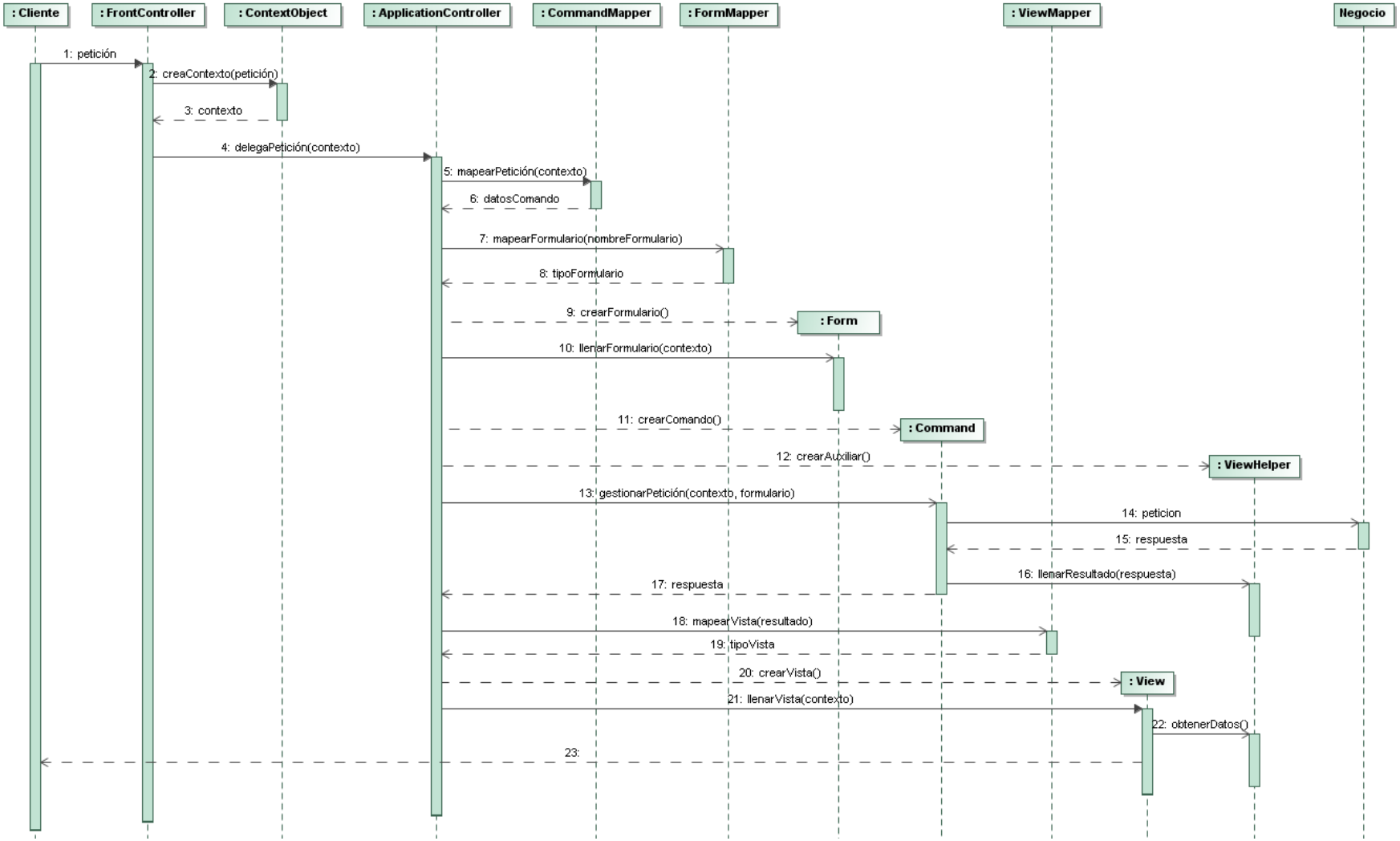


Ilustración 36: Diagrama de secuencias - Gestión de una petición

Llega la petición

1. El cliente envía una petición a la aplicación. FrontController la recibe.
2. FrontController crea un nuevo ContextObject, en el que almacenar la información del contexto de la petición.
3. FrontController obtiene el objeto de contexto.
4. FrontController envía el contexto a ApplicationController, delegando en él la gestión de la petición.

Obtención del comando para gestionarla

5. ApplicationController envía a CommandMapper el contexto, para consultar qué Command debe tratar la petición.
6. CommandMapper devuelve un objeto de tipo CommandData, con la clase de comando, el nombre del FormBean, los FilterMapper y el ViewMapper de esa petición.
7. ApplicationController envía el nombre del FormBean a FormMapper, para obtener la clase de formulario que se utiliza para encapsular la petición.
8. FormMapper devuelve la clase de formulario.
9. ApplicationController crea un nuevo FormBean del tipo adecuado.
10. Le envía el contexto para que almacene sus datos en el objeto. Si ocurren fallos se almacenarán para mostrarlos al cliente.
11. ApplicationController crea un nuevo Command del tipo adecuado.
12. ApplicationController crea un nuevo ViewHelper, y lo añade al contexto.

Gestión de la petición

13. ApplicationController envía el contexto y el Form al comando para que gestione la petición.
14. Command llama a la lógica de negocio.
15. Recibe la respuesta.

16. Envía a ViewHelper los datos de la respuesta para que se pueble.

17. Command envíe el tipo de resultado de la respuesta a ApplicationController.

Generación de la vista

18. ApplicationController envía al ViewMapper asociado a la petición el tipo de resultado.

19. Con él, ViewMapper puede devolverle el tipo de vista que debe utilizar para mostrar el resultado.

20. ApplicationController instancia una nueva vista, del tipo adecuado.

21. Envía a la nueva vista el contexto, para que tenga acceso a los datos de la respuesta y de la dirección a la que enviarla.

22. View accede a los datos de la respuesta, disponibles en ViewHelper.

23. View envía la respuesta al cliente. Termina la gestión de la petición.

Por motivos de claridad visual se ha evitado mostrar la posible ejecución de filtros. Estos se ejecutarán de forma previa y posterior al propio Command, aunque no accederán (normalmente) a la lógica de negocio.

4.3. Implementación

4.3.1. Decisiones de diseño

Aunque el mundo de la informática ha tratado casi desde sus orígenes convertir la reutilización en uno de sus paradigmas, no es menos cierto que, en el terreno del aprendizaje informático, una forma fundamental de progreso es enfrentarse personalmente a los problemas y solucionarlos.

Es por este motivo que parte de las funciones que aporta el framework ya están provistas en populares bibliotecas y, sin duda alguna, mucho mejor solucionadas. Sin embargo, en aras del aprendizaje se ha preferido implementar desde cero clases y métodos que nos recordarán a otras herramientas, como FormTools recuerda a la conocida biblioteca BeanUtils, o ciertos Tags que no aportan mucho sobre los ya existentes en la biblioteca estándar de Tags de Java, pero que funcionan correctamente y dotan a Myst de la funcionalidad esperada.

4.3.2. Framework Myst

La implementación del framework ha sido realizada utilizando una metodología de desarrollo rápido, en la que las pruebas se han ido realizando lo más pronto posible, según se programaba el código. Sin embargo, dada la naturaleza del programa, un framework JEE de presentación, que no es una aplicación completa y que debe además ser desplegada en un Servlet, esta tarea no ha sido posible hasta tener un esqueleto básico.

Una vez alcanzada esta funcionalidad mínima, el desarrollo rápido ha facilitado detectar y solventar errores y encontrar nuevas necesidades muy pronto, lo cual ha permitido añadir nuevas funcionalidades a la etapa de diseño e implementarlas sin suponer un esfuerzo extraordinario.

El framework Myst está compuesto por los siguientes paquetes:

Packages

Package	Description
edu.uoc.myst.auxiliary	
edu.uoc.myst.command	
edu.uoc.myst.configuration	
edu.uoc.myst.configuration.xml	
edu.uoc.myst.core	
edu.uoc.myst.exception	
edu.uoc.myst.exception.handler	
edu.uoc.myst.filter	
edu.uoc.myst.form	
edu.uoc.myst.international	
edu.uoc.myst.view	
edu.uoc.myst.view.taglib	

Ilustración 37: Paquetes del framework Myst

A continuación se describirán todos los paquetes individualmente, describiendo sus clases y principales funcionalidades. Para un estudio más en profundidad se recomienda consultar la profusa documentación presente en el código o Javadoc del framework, donde todos los métodos están descritos con claridad.

Core

Núcleo del framework, formado por los controladores principales y por el objeto de contexto, que da unión a las diferentes partes del framework y a las aplicaciones que lo utilizan.

Class Summary

Class	Description
ApplicationController	Despachador o controlador de aplicaciones del framewrok Myst.
ContextObject	Clase auxiliar que almacena los datos del contexto de la petición, de forma que las demás clases del framework Myst(y sus aplicaciones) puedan acceder a los datos de la petición de forma transparente a las peticiones HTTP.
FrontController	Controlador frontal del framework Myst.

Ilustración 38: Clases del paquete Core

-FrontController

Implementa el controlador frontal del framework Myst. Realiza dos funciones fundamentales:

- Durante la inicialización del Servlet carga la configuración a través de ConfigurationManager, e instancia el ApplicationController o despachador.
- Además, gestiona todas las peticiones de los clientes de la aplicación usuaria de Myst. Para ello genera una configuración del contexto o ContextObject, donde guarda los datos de la petición, y la envía al despachador para delegar el tratamiento de la petición.

-ApplicationController

Despachador o controlador de aplicaciones del framework Myst.

Recibe de forma sucesiva las peticiones HTTP a través del controlador frontal. Utilizando la configuración de la aplicación decide qué comando debe gestionar cada petición, qué formulario lleva asociado, y si utiliza filtros antes y/o después de ejecutar el comando. Instancia el comando, le envía la petición, y en función de su respuesta resuelve qué vista debe mostrarse al cliente, y la genera.

En caso de acontecer excepciones, también decide a través de la configuración qué gestor de excepciones y qué vista deben utilizarse para tratarlas.

Para realizar su trabajo, ApplicationController realiza inversión de control, es decir, se encarga de controlar el flujo de las aplicaciones usuarias de Myst y de llamar a las clases de la aplicación cuando sea necesario, en lugar de ser

la aplicación usuaria quien llame al framework. Además, también utiliza un motor de inyección de dependencias, de forma que crea objetos y los inserta en otros. Por ejemplo, así es como asigna el formulario, el objeto de contexto y el asistente de vistas a los comandos.

Un beneficio adicional de este modo de funcionamiento es la posibilidad de utilizar objetos POJO como comandos, formularios y filtros, los cuales son de tipo desconocido en compilación, y solamente son identificados en ejecución. La llave de esta funcionalidad es la reflexión, una técnica que permite a un programa informático modificar su estructura de alto nivel en tiempo de ejecución.

-ContextObject

Clase auxiliar que almacena los datos del contexto de la petición, de forma que las demás clases del framework Myst, y las de las aplicaciones que lo utilicen, puedan acceder a los datos de la petición de forma transparente a las peticiones HTTP.

Además, al estar asociado a casi cualquier objeto del framework, facilita la comunicación entre las diferentes clases que lo usan, como almacenar una referencia al ViewHelper para que esté disponible en la generación de vistas, o la región del contexto, para la regionalización.

Configuration

Conjunto de clases que se encargan de cargar en memoria el archivo XML con la configuración de la aplicación, y de cargar los parámetros de configuración del Servlet del archivo web.xml.

Class Summary

Class	Description
ConfigurationManager	Clase que gestiona la configuración de la aplicación usuaria del framework Myst.

Class Summary

Class	Description
CommandData	Java class for commandData complex type.
CommandMapper	Java class for commandsType complex type.
CommandMapperAdapter	
ErrorData	Java class for errorType complex type.
ErrorMapper	Java class for errorsType complex type.
ErrorMapperAdapter	
FilterData	Java class for filterType complex type.
FilterMapper	Java class for filtersType complex type.
FilterMapperAdapter	
FilterNameData	Java class for filterNameType complex type.
FilterNameMapper	Java class for filtersNameType complex type.
FormData	Java class for formType complex type.
FormMapper	Java class for formsType complex type.
FormMapperAdapter	
MystConfiguration	Java class for mystConfiguration complex type.
ObjectFactory	This object contains factory methods for each Java content interface and Java element interface generated in the configuration2.xml package.
Parameter	Java class for parameterType complex type.
ParameterData	Java class for parameterType complex type.
ParameterMapper	Java class for parametersType complex type.
ParameterMapperAdapter	
ViewData	Java class for viewType complex type.
ViewMapper	Java class for viewsType complex type.
ViewMapperAdapter	

Ilustración 39: Clases del paquete Configuration

-ConfigurationManager

Clase que gestiona la configuración de la aplicación usuaria del framework Myst, así como los parámetros recibidos a través del archivo de configuración de la aplicación web.

Para tratar el archivo XML utiliza una serie de clases generadas por el plugin de Eclipse JAXB, que utiliza como fuente el archivo XML Schema de configuración del framework (MystSchema.xsl). Esto proporciona directamente todos los comandos, formularios, vistas, mapeados, etc. Para la validación y carga del archivo en memoria se utiliza la API de Java para el procesamiento de XML (javax.xml).

Command

Aunque la función de este paquete de clases es facilitar la creación de comandos, el framework Myst permite la utilización de objetos POJO como comandos. Este hecho beneficia la construcción de aplicaciones, que no estarán acopladas a Myst y serán fácilmente portables a otros frameworks o entornos.

Interface Summary

Interface	Description
Command	Interfaz que permite definir el comando asociado a una petición HTTP.

Class Summary

Class	Description
AbstractCommand	Clase abstracta que permite definir el comando asociado a una petición HTTP.

Ilustración 40: Clases del paquete Command

-Interface Command

Interfaz que permite definir el comando asociado a una petición HTTP.

Todo comando que utilice el framework Myst puede implementar esta clase, extender la clase abstracta AbstractCommand, o ser un objeto POJO que respete las siguientes especificaciones:

1. Si utiliza un formulario Form, debe definir el método público setFormBean.
2. Si utiliza un ViewHelper para facilitar las vistas, debe definir el método público setViewHelper.
3. Si además quiere tener acceso al contexto de la petición HTTP, debe definir el método público setContextObject.

-AbstractCommand

Clase abstracta que permite definir el comando asociado a una petición HTTP.

Todo comando que utilice el framework Myst puede extender esta clase, implementar la interfaz Command, o respetar las especificaciones indicadas previamente en la interfaz.

Form

Como en el caso del paquete Command, su función es facilitar la creación de formularios o beans, pero puede optarse con tranquilidad por usar objetos POJO como comandos.

Interface Summary

Interface	Description
Form	Interfaz que permite definir el formulario asociado a un comando.

Class Summary

Class	Description
AbstractForm	Clase abstracta que permite definir el formulario asociado a un comando.

Ilustración 41: Clases del paquete Form

-Interface Form

Interfaz que permite definir el formulario asociado a un comando.

Todo formulario que utilice el framework Myst puede implementar esta clase, extender la clase abstracta AbstractForm, o respetar las siguientes especificaciones:

1. Debe tener un conjunto de atributos igual al de los parámetros de la petición HTTP asociada, con los mismos nombres.
2. Debe definir métodos públicos setter y getter para todos sus atributos.
3. Si además quiere tener acceso al contexto de la petición HTTP, debe definir el método público setContextObject(ContextObject)

Form es muy similar a un bean Java, pero no es necesario que sea serializable.

-AbstractForm

Clase abstracta que permite definir el formulario asociado a un comando.

Todo formulario que utilice el framework Myst puede extender esta clase, implementar la interfaz Form, o respetar las especificaciones indicadas anteriormente en la interfaz.

View

Paquete de clases que facilitan la creación de nuevas vistas, ofrecen los objetos ViewHelper de ayuda a la presentación de vistas, y además ofrecen dos vistas básicas implementadas: los métodos forward y redirect.

Interface Summary

Interface	Description
View	Interfaz que permite definir las diferentes vistas.

Class Summary

Class	Description
AbstractView	Clase abstracta que permite definir las diferentes vistas.
ForwardView	Vista que realiza una operación de reenvío.
RawView	Vista que realiza una escritura "en crudo" directamente en la salida.
RedirectView	Vista que realiza una operación de redirección.
ViewHelper	Objeto que facilita el acceso a la información del negocio desde páginas JSP.

Ilustración 42: Clases del paquete View

-Interface View

Interfaz que permite definir las diferentes vistas. Toda vista que utilice el framework Myst debe implementar esta clase, o extender la clase abstracta AbstractView.

-AbstractView

Clase abstracta que permite definir las diferentes vistas. Toda vista que utilice el framework Myst debe extender esta clase, o implementar la interfaz View.

-ForwardView

Vista que realiza una operación de reenvío.

Un reenvío es realizado de forma interna por el servlet, por lo que pasa desapercibido para el navegador y la URL original se mantiene intacta. Una recarga del navegador en la página resultante repetirá la petición original, con la misma URL.

No debe utilizarse si la petición original modifica datos en un almacén.

-RedirectView

Vista que realiza una operación de redirección.

En una redirección la aplicación solicita al navegador que obtenga una segunda URL, distinta de la original. Por tanto, una recarga del navegador en la página resultante no repetirá la petición original, sino que obtendrá la segunda URL.

-RawView

Vista que realiza una escritura "en crudo" directamente en la salida. Se utiliza, por ejemplo, para mostrar por pantalla las excepciones que no tienen un gestor asignado.

-ViewHelper

Objeto que facilita el acceso a la información del negocio desde páginas JSP. Permite guardar los elementos deseados de la lógica en un mapa, a los que se accede cómodamente desde las páginas JSP con el tag “form” de la biblioteca Myst.

Al almacenar los elementos, convierte los objetos tipo Collection en arrays, y los objetos tipo Form en mapas, a través de los métodos de clase de FormTools. Esta operación se realiza de forma recursiva sobre todas las componentes de los elementos.

Exception

Conjunto de clases que aportan una serie de excepciones derivadas del uso erróneo del framework Myst, una interfaz para gestionarlas, y una implementación básica de gestión de errores.

Exception Summary

Exception	Description
ConversionException	Excepción causada cuando un objeto no puede ser convertido a ningún tipo primitivo Java.
FilterException	Excepción causada cuando un filtro devuelve un resultado erróneo.
FormLoadException	Excepción causada cuando suceden uno o más fallos en la carga de datos en un formulario.
MystException	Excepción genérica del framework Myst.
NoGlobalHandlerException	Excepción causada cuando sucede una excepción y no hay ningún gestor, local o global, que pueda tratarla.
NoParameterException	Excepción causada cuando un tag trata de acceder a un elemento de la lógica de negocio que no existe.
NoPathException	Excepción causada cuando una ruta a un recurso no accede a un recurso real.
NotFormattableException	Excepción causada cuando un objeto de una clase no puede ser regionalizado automáticamente por el filtro `InternationalFormatFilter`.
NotSuchCommandException	Excepción causada cuando la clase de un comando no puede ser encontrada en la aplicación.
NotSuchFormException	Excepción causada cuando un formulario no ha sido correctamente definido y asignado a su comando en el archivo de configuración.
NoViewException	Excepción causada cuando el resultado de un comando no tiene ninguna vista local ni global definida en el archivo de configuración.
WrongPatternException	Excepción causada cuando una ruta a un recurso no respeta la sintaxis definida por el método `getElement` de `RegExTools`.

Interface Summary

Interface	Description
ExceptionHandler	Interfaz que permite gestionar una excepción sucedida en la aplicación o el framework.

Class Summary

Class	Description
InternationalExceptionHandler	Gestor de excepciones que obtiene los mensajes regionalizados de todas las excepciones sucedidas para mostrarlos por pantalla.

Ilustración 43: Clases del paquete Exception

-MystException

Excepción genérica del framework Myst. El resto de excepciones del framework la extenderán, de forma que sean fácilmente identificables. Todas ellas implementan un método, getLocalizedMessage(Locale), que presenta la versión regionalizada del mensaje de error.

En estos momentos presentan mensajes de error en inglés y español, tal como está definido en el bundle de idiomas ExceptionBundle.

-Interface ExceptionHandler

Interfaz que permite gestionar una excepción sucedida en la aplicación o el framework.

Típicamente, un gestor de errores debe escribir la información que quiera mostrar al usuario en el objeto ViewHelper, de igual forma a la presentación de una vista sin errores. El framework se encargará después de inyectar este ViewHelper en la vista que se haya elegido en la configuración de la aplicación.

Todo gestor de excepciones que utilice el framework Myst debe implementar esta interfaz.

-InternationaExceptionHandler

Gestor de excepciones que obtiene los mensajes regionalizados de todas las excepciones sucedidas para mostrarlos por pantalla.

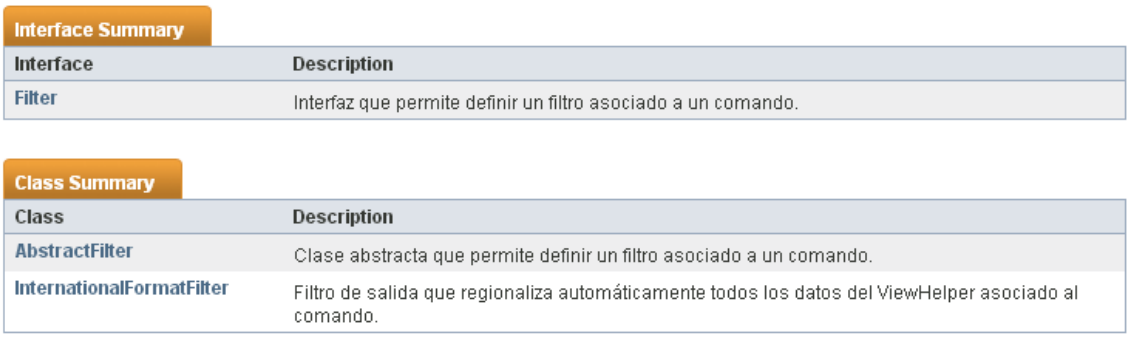
Se distribuye conjuntamente a la vista “internationalError.jsp”, que presenta adecuadamente dicha información al cliente HTTP.

Filter

Paquete que permiten definir y crear filtros. Los filtros son prácticamente iguales a los comandos, se pueden considerar un subconjunto de ellos. Son reutilizables, pero deben ir siempre asociados a un comando; por tanto, no definen vistas, ni formularios, ni gestores de errores, sino que utilizarán los del comando asociado.

Como en el caso de los comandos, los filtros pueden utilizar las clases proporcionadas o definir objetos POJO, sin apenas acoplamiento con el framework.

El paquete también se aporta un filtro de muestra, útil en la regionalización.



Interface Summary

Interface	Description
Filter	Interfaz que permite definir un filtro asociado a un comando.

Class Summary

Class	Description
AbstractFilter	Clase abstracta que permite definir un filtro asociado a un comando.
InternationalFormatFilter	Filtro de salida que regionaliza automáticamente todos los datos del ViewHelper asociado al comando.

Ilustración 44: Clases del paquete Filter

-Interface Filter

Interfaz que permite definir un filtro asociado a un comando.

Todo filtro que utilice el framework Myst puede implementar esta clase, extender la clase abstracta AbstractFilter, o implementar un objeto POJO desde cero.

Si dicho objeto POJO debe tener acceso al contexto de la petición HTTP, tiene que definir el método público setContextObject(ContextObject) . Si debe tener acceso al formulario, bien sea el Form de entrada, en el caso de los filtros de entrada, o el objeto ViewHelper, para los filtros de salida, tiene que definir en ambos casos el método público setFormBean(Object). En cualquier caso debe definir el método execute(), donde se especificará la lógica.

-AbstractFilter

Clase abstracta que permite definir un filtro asociado a un comando.

Todo filtro que utilice el framework Myst puede extender esta clase, implementar la interfaz Filter, o respetar las especificaciones indicadas en la interfaz.

-InternationalFormatFilter

Filtro de salida que regionaliza automáticamente todos los datos del ViewHelper asociado al comando.

Para poder ser regionalizados, los datos deben ser tipos primitivos de Java; de tipo Date; compatibles con la especificación de un Form (con datos

accesibles por getters); arrays de tipos primitivos, Dates o Forms; o cualquier combinación de los anteriores.

Utiliza la biblioteca de regionalización I18N.

Auxiliary

Paquete compuesto por herramientas que proveen funcionalidades necesarias para diversas partes del framework. Además, proporciona una clase centralizada con las constantes utilizadas por las demás clases.

Class Summary

Class	Description
Constants	Biblioteca de constantes del framework Myst.
FormTools	Clase que provee un conjunto de herramientas para facilitar el tratamiento y la conversión de objetos de tipo Form.
RegExTools	Clase que proporciona herramientas para el tratamiento de expresiones regulares.

Ilustración 45: Clases del paquete Auxiliary

-FormTools

Conjunto de herramientas, basadas en la reflexión, que permiten realizar diversas operaciones con objetos conformes a la definición de un Form. Entre sus métodos más importantes destacan:

- **formToMap:** convierte un objeto conforme a Form en un mapa, de forma que pueda ser accedido por desde la vistas a través de la lógica del framework.
- **mapToForm:** permite cargar los datos de un mapa en un objeto conforme a Form, que debe definir un setter para cada atributo del mapa. Con este método se cargan los parámetros de las peticiones HTTP en los Form asociados a los comandos.

-RegExTools

Define un único método, getElement(), que permite obtener el último elemento de una ruta dentro de un mapa.

La ruta proporcionada es navegable, y se puede acceder a cualquier nivel de profundidad. Por ejemplo, para acceder al atributo pétalo de un elemento flor la ruta sería:

flor.petalo

Si pétalo fuera un array con varios elementos, también se pueden acceder individualmente, como a su cuarto elemento:

flor.petalo[3]

Y si en cada pétalo hubiera una mosca:

flor.petalo[3].mosca

International

Class Summary

Class	Description
I18N	Clase que facilita la regionalización del framework Myst, y de las aplicaciones que lo utilizan.

Ilustración 46: Clases del paquete International

-I18N

Clase que facilita la regionalización del framework Myst, y de las aplicaciones que lo utilizan. Para ejecutar su lógica utiliza dos aplicaciones proporcionadas por Java:

- **ResourceBundle:** clase que permite acceder a archivos de idiomas, donde se definen varios pares nombre – valor. Según la región definida en ese momento, una petición de un nombre a ResourceBundle obtendrá el valor en el archivo de la región adecuada.
- **Clases Format (NumberFormat y DateFormat):** al proporcionarles un número, fecha u hora, devuelven la versión estandarizada en la región definida en ese momento.

Taglib

Sub paquete del paquete View, se explica aparte por claridad.

Conjunto de etiquetas que amplían la funcionalidad de las vistas JSP, y le facilitan el acceso a la información generada por la aplicación.

Class Summary

Class	Description
FormTag	Tag que permite acceder a un elemento individual del mapa del objeto ViewHelper asociado al comando.
I18NTag	Tag que permite regionalizar un objeto.
IterTag	Tag que permite realizar iteraciones sobre vectores del mapa definido en el objeto ViewHelper.
SetTag	Tag que permite asignar a una variable un objeto del mapa del ViewHelper.

Ilustración 47: Clases del paquete Taglib

-I18NTag

Tag que permite regionalizar una cadena. La sintaxis de uso en una petición JSP es la siguiente:

<myst:i18n message="cadena" bundle="nombreDelBundle"/>

Donde cadena es el nombre del objeto que se quiere regionalizar, y nombreDelBundle el bundle donde se encuentra dicho String.

-SetTag

Tag que permite asignar a una variable un objeto del mapa del ViewHelper. La sintaxis de uso en una petición JSP es la siguiente:

<myst:set var="variable" element="elemento"/>

Donde elemento es la ruta del objeto que se quiere obtener, y variable la variable que almacenará el objeto. Esta etiqueta permite evitar accesos repetidos a un mismo objeto.

-FormTag

Tag que permite acceder a un elemento individual del mapa del objeto ViewHelper asociado al comando. La sintaxis de uso en una petición JSP es la siguiente:

<myst:form element="elemento"/>

Donde elemento es la ruta al elemento que se quiere obtener

-IterTag

Tag que permite realizar iteraciones sobre vectores del mapa definido en el objeto ViewHelper. La sintaxis de uso en una petición JSP es la siguiente:

```
<myst:iter var="variable" element="elemento">
        $<variable>
<myst:iter/>
```

Donde elemento es la ruta del vector que se quiere obtener, y variable la variable donde irá iterando el vector.

Típicamente, dentro del bucle se accederá a un elemento específico de la variable mediante un FormTag.

4.3.3. Manual de usuario

Myst se distribuye como una biblioteca, y también como un proyecto Java Maven. Cualquier aplicación que quiera utilizar este framework tendrá que añadir la biblioteca a sus dependencias, así como heredar también las dependencias Maven del framework.

El resto es bastante sencillo. Básicamente, el programador de aplicaciones web debe definir acciones, con formularios, filtros, vistas y páginas JSP asociadas, que implementen la lógica de presentación deseada, que habitualmente consistirá en realizar peticiones a la lógica del negocio. Dichas asociaciones deben especificarse explícitamente en un archivo de configuración de la aplicación, que se cargará durante la carga de la aplicación en el contenedor web.

De forma más detallada, el programador debe definir dos archivos de configuración de la siguiente manera:

Archivo de configuración del Servlet (web.xml)

Toda aplicación web debe configurar ciertos parámetros en este archivo de configuración. En el caso de las aplicaciones usuarias de Myst debe especificarse el Servlet FrontController, la clase del framework Myst que implementa el Servlet. Además, el framework admite tres parámetros:

- app-file: ruta del archivo de configuración de la aplicación.
- date-style: formato de fechas. Admite los valores SHORT, MEDIUM, LONG y FULL, tal como lo define la clase java.text.DateFormat.
- time-style: formato de horas. Admite los mismos valores que date-style.

Archivo de configuración de la aplicación

Su nombre y ruta serán los especificados en el parámetro “app-file” del archivo “web.xml”. De forma predeterminada se llamará “Myst.xml”, y se alojará en la carpeta “resources” de la aplicación.

Sigue una convención muy similar a la de Struts[2], y debe ser conforme al esquema XML “MystSchema.xsd”, disponible en la carpeta “resources/xsd” del código fuente. Antes de explicar su sintaxis en detalle, sirva a modo de anticipo este ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<myst xmlns="http://myst.uoc.edu/MystSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://myst.uoc.edu/MystSchema MystSchema.xsd">

<global-errors>
      <error type="java.lang.Throwable"
      class="edu.uoc.myst.exception.handler.InternationalExceptionHandler>
            <parameters>
                  <parameter name="next">/jsp/internationalError.jsp
                  </parameter>
            </parameters>
      </error>
</global-errors>

<global-views>
      <view name="WARNING" class="view.Warning">
            <parameters>
                  <parameter name="next">/jsp/warning.jsp</parameter>
            </parameters>
      </view>
</global- views>

<filters>
      <filter name="internationalFormat"
            class="edu.uoc.myst.filter.InternationalFormatFilter" />
      <filter name="FiltroFalso" class="filter.NoExisto" />
</filters>

<forms>
      <form name="testHeredado" class="form.InheritedTestForm" />
      <form name="testNoHeredado" class="form.NonInheritedTestForm" />
</forms>

<commands>
      <command name="idioma" class="command.ChangeLanguage">
            <pre-filters>
                  <filter name="internationalFormat" />
            </pre-filters>
            <post-filters>
                  <filter name="internationalFormat" />
            </post-filters>
            <views>
                  <view name="SUCCESS"
                        class="edu.uoc.myst.view.ForwardView">
                        <parameters>
                              <parameter name="next">/jsp/start.jsp
                              </parameter>
                        </parameters>
                  </view>
            </views>
            <errors>
                  <error type="exception.LanguagueException"
                        class="handler.LanguagueExceptionHandler">
                        <parameters>
                        <parameter name="next">/jsp/languagueError.jsp
                              </parameter>
                        </parameters>
                  </error>
            </errors>
      </command>
</commands>
</myst>
```

Ilustración 48: : Ejemplo de archivo de configuración de Myst

El archivo de configuración de una aplicación usuaria de Myst tiene cinco bloques:

- global-errors: optativo. Define gestores de errores globales, a los cuales se acudirá cuando un error no pueda ser atendido por los gestores definidos en el comando que lo provocan. Cada error define tres atributos:
  - type: obligatorio. Clase del error a gestionar.
  - class: obligatorio. Clase gestora del error.
  - parameters: optativo. Serie de parámetros, completamente flexibles, en los que se pueden especificar los datos que se quieran. Uno muy típico es definir la página a mostrar.

- global-views: optativo. Define vistas globales, a las cuales se acudirá cuando un valor de retorno de un comando no tenga una vista asociada en el propio comando. Cada vista define tres atributos:
  - name: obligatorio. Valor de retorno del comando.
  - class: obligatorio. Clase que genera la vista.
  - parameters: optativo. Serie de parámetros, completamente flexibles, en los que se pueden especificar los datos que se quieran. De nuevo es típico definir la página a mostrar.

- filters: optativo. Declara los diferentes filtros, tanto de entrada como de salida, que se usarán en los comandos. Cada filtro tiene dos parámetros obligatorios:
  - name: nombre dado al filtro, que se usará para referenciarle.
  - class clase que implementa el filtro.

- forms: optativo. Declara los diferentes formularios que se usarán en los comandos. Sus parámetros, obligatorios, son los mismos que los de los filtros:
  - name: nombre dado al formulario, que se usará para referenciarle.
  - class clase que implementa el formulario.

- commands: único bloque obligatorio. Cada comando declara las peticiones aceptadas por la aplicación, y los formularios, vistas, filtros y gestores de errores que se usarán para cada uno. Un comando, pues, define los siguientes atributos:
  - name: obligatorio. Nombre del comando, que coincide con la petición HTTP que usará dicho comando.
  - class: obligatorio. Clase que implementa el comando.

- form: optativo. Formulario que recoge los datos de entrada de la petición.
- pre-filters: optativo. Filtros de ejecución previa al comando.
- post-filters: optativo. Filtros de ejecución posterior al comando
- views: optativo. Como los global-views, pero específicos de cada comando, y de sus valores de retorno.
- errors: optativo. Como los global-errors, pero específicos de cada comando, y de sus excepciones.

Cómo internacionalizar una aplicación

Las clases aportadas por el framework facilitan en gran manera crear una aplicación en varios idiomas. Aunque el framework es completamente flexible, y el programador puede utilizar en cualquier momento los métodos de la clase I18N, se recomienda actuar de la siguiente manera:

-Si se quiere internacionalizar en la lógica

Definir un bundle de idiomas para cada clase que quiera internacionalizar datos. Utilizar el método localize(String) disponible en los command y forms que hereden de los proporcionados por el framework. Directamente accederá al bundle adecuado.

Además, el filtro internationalFormatFilter puede aplicarse a la salida de los comandos para formatear automáticamente números y fechas.

-Si se quiere internacionalizar en la vista

Definir los bundles de idiomas que se desee y utilizar el tag I18NTag para acceder a ellos, tal como aparece en la explicación de la clase en el apartado 4.3.2.

Importación del código fuente

El proyecto ha sido construido con la herramienta Maven, por lo que la importación del código fuente y las bibliotecas debiera ser prácticamente automática. En Elipse o cualquier otro IDE habrá que seleccionar abrir un proyecto existente, seleccionar adecuadamente la carpeta, y Maven descargará las dependencias necesarias directamente de los repositorios.

Es conveniente asegurarse de que todos los recursos se empaquetan correctamente al compilar el código, puesto que ciertas versiones de Maven

pueden tener comportamientos algo erráticos. El contenido completo de las siguientes carpetas de recursos debe ser desplegado en la carpeta Myst/target/classes:

Myst/src/main/resources (salvo los subdirectorios bundles y xsd)
Myst/src/main/resources/bundles
Myst/src/main/resources/xsd

De otra forma puede haber determinados archivos (como el esquema XSL, o la biblioteca de tags) que no se añadan correctamente a la biblioteca al compilar.

Puede compilarse el código con las opciones “package” o “install” de Maven.

4.3.4. Aplicación para pruebas: MystTest

Un framework es, sin duda, una herramienta útil, pero no es un programa completo. Este carácter fragmentario convierte la realización de juegos de pruebas tradicionales, automatizados, en una tarea, como poco, muy complicada. El núcleo del framework, ApplicationController, no es comprobable sin la creación de comandos, formularios, vistas y filtros ficticios.

Los programadores de aplicaciones web conocen también la dificultad para realizar juegos de pruebas, puesto que solamente cierta parte de la funcionalidad de las aplicaciones se puede probar de formar automática, siendo muchas veces necesaria la utilización de complejos objetos virtuales o mocks para comprobar la aplicación en conjunto. Otro apartado importante de comprobar, la interacción de usuario, es prácticamente imposible de automatizar, siendo un método más útil, completo y sencillo la realización de pruebas interactivas por personal ajeno al proyecto.

Estas razones son las que han llevado a realizar un aplicación de prueba, que tratará de demostrar el buen funcionamiento del framework ante todo tipo de situaciones, tanto buscando resultados esperados como provocando la generación de excepciones. Esta aplicación, llamada MystTest, será directamente comprobable como una aplicación web, cargada en el servidor de aplicaciones.

La aplicación de prueba se distribuye como aplicación web compilada, la cual puede cargarse en cualquier servidor web o de aplicaciones, además de como código fuente. Dicho código, además de las páginas JSP, se encuentra rigurosamente documentado, por lo que se invita a su consulta.

Importación del código fuente

La aplicación se ha construido como un proyecto Maven, por lo que los pasos para importar el proyecto son los mismos que para el framework. También habrá que añadir la biblioteca compilada "Myst.jar" al proyecto, añadiéndola a la carpeta de repositorio de la versión de Maven que utilicemos (o del IDE).

Puede compilarse el código con las opciones "install" o "package" de Maven.

Ejecución de la aplicación

Si desplegamos la aplicación compilada, por ejemplo, en Tomcat 7.0 nos encontraremos:

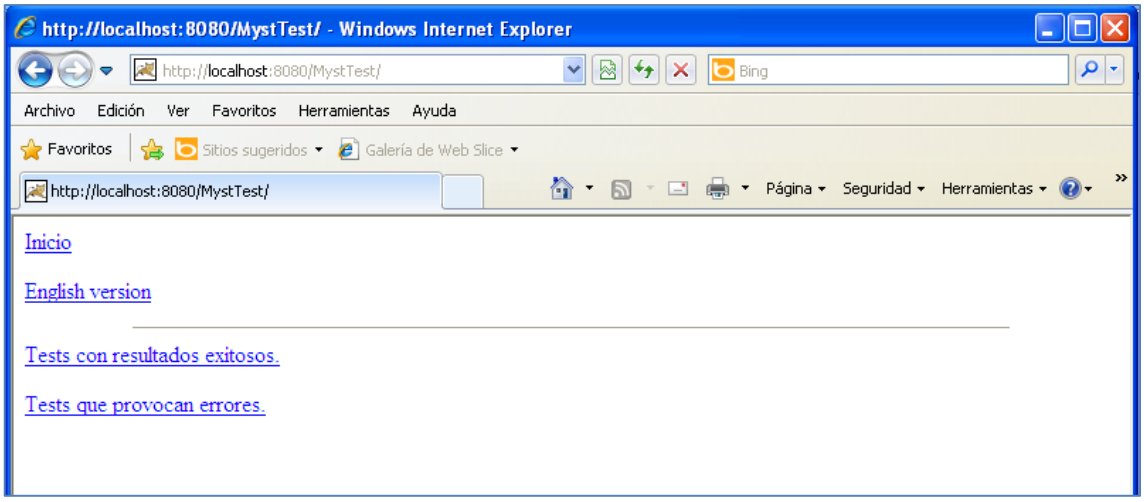


Ilustración 49: Menú de la aplicación de prueba

El menú nos deja elegir la versión entre dos series de tests:

Tests con resultados exitosos

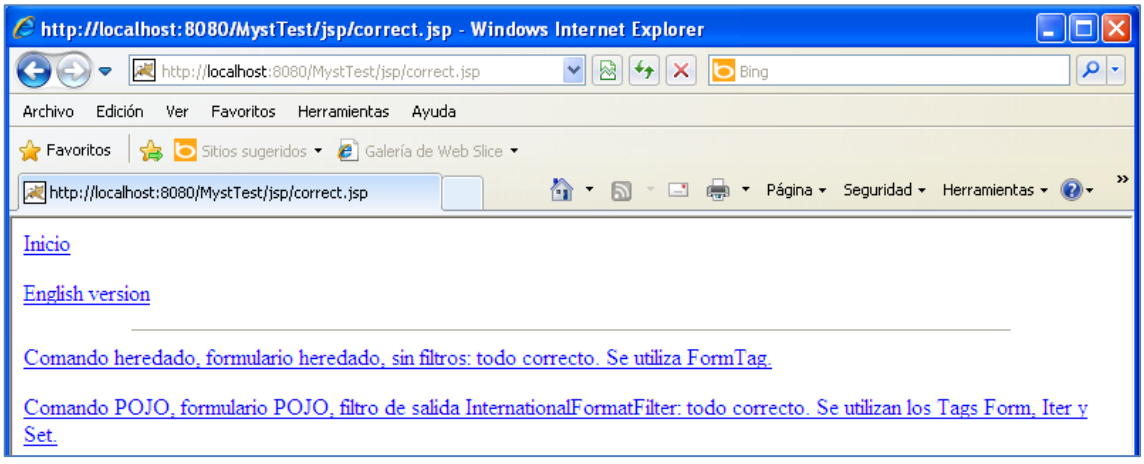


Ilustración 50: Tests con resultados exitosos

- “Comando heredado” utiliza un formulario y un comando que heredan de las clases abstractas facilitadas por el framework. Además, en la vista utiliza el tag FormTag. La prueba se realiza perfectamente.

- “Comando POJO” utiliza un formulario y un comandos que no heredan de ninguna clase del framework, sino que son objetos POJO conformes al estándar definido por el framework. Utiliza además el filtro InternationalFormatFilter, y tags FormTag, IterTag y SetTag en la presentación. Todo se realiza exitosamente.

Tests que provocan errores

Completa serie de pruebas que fuerzan, tanto en los comandos, las vistas, los formularios y la configuración de la aplicación, la aparición de errores. Como se puede comprobar, todos los tests se superan exitosamente, capturando el framework las excepciones y presentándolas por pantalla.

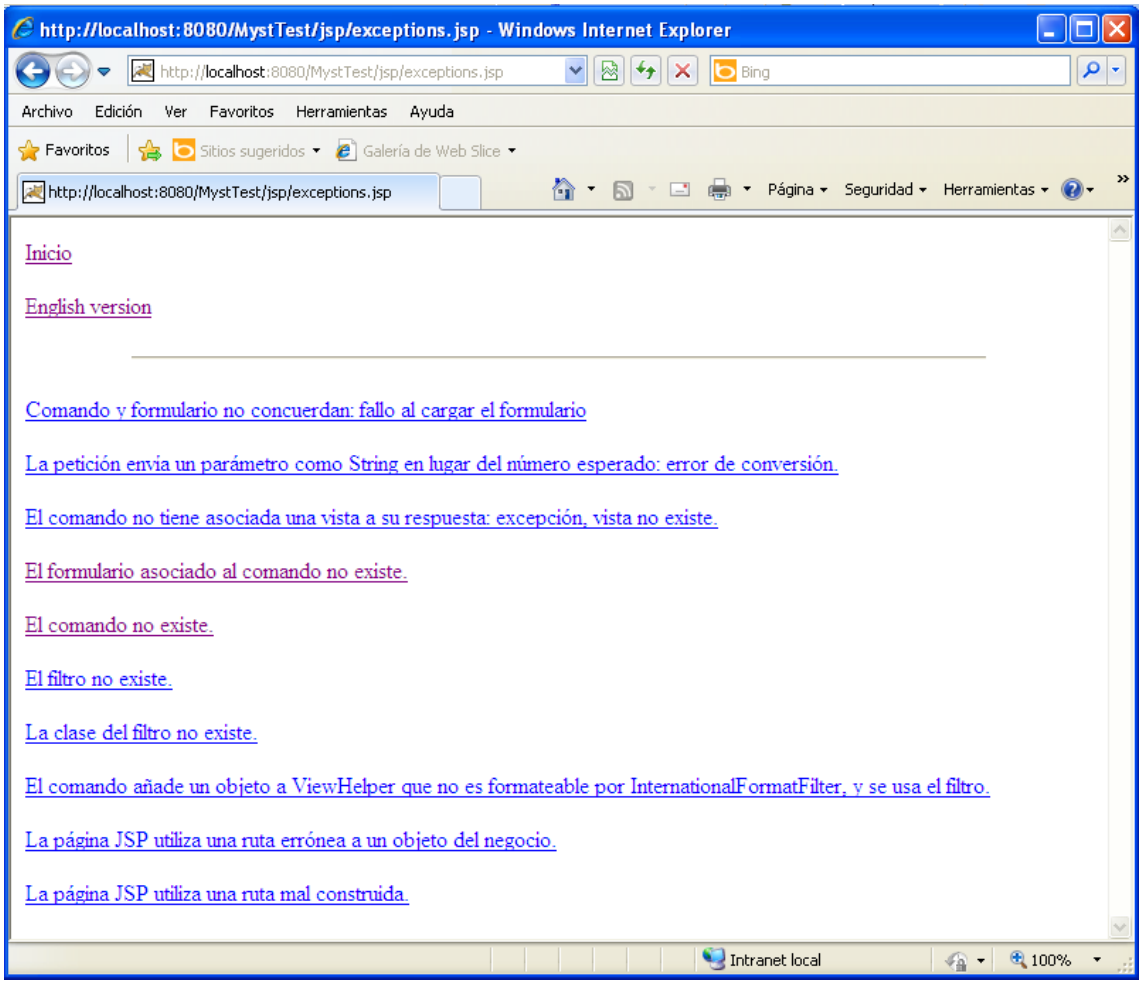


Ilustración 51: Tests que provocan errores

Además, la cabecera de las páginas permite alternar el idioma entre inglés y español, con lo que también se prueba la internacionalización de la aplicación. Por ejemplo, en el error de un formulario inexistente asociado a un comando:

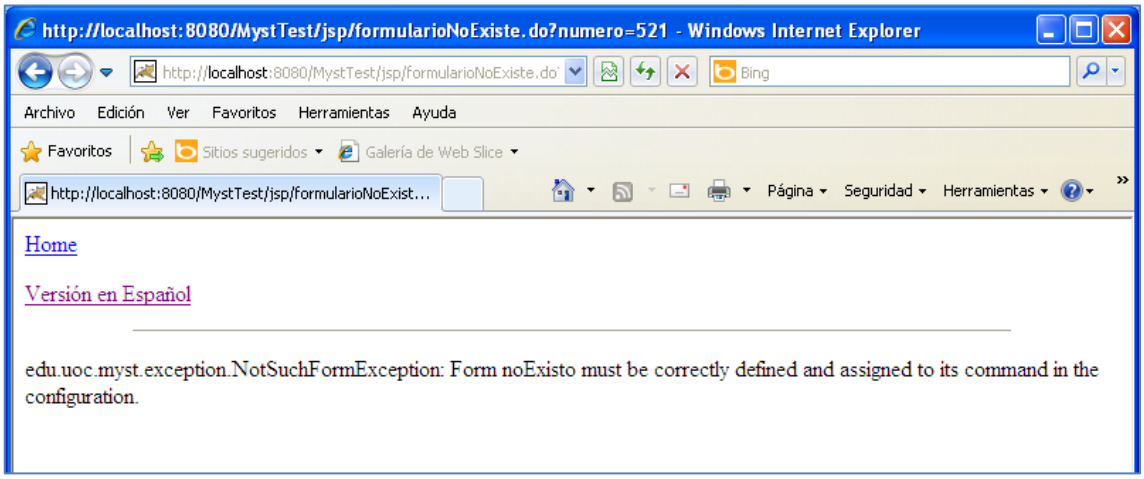


Ilustración 52: Ejemplo de error internacionalizado

Como se puede ver, aparece el mensaje de error correctamente regionalizado.

Resultado de las pruebas

Debo destacar que las pruebas realizadas entrarían dentro de la categoría de pruebas de integración, ya que no se realiza la comprobación de métodos de forma aislada, sino que prueba a la vez una combinación de métodos de varias clases.

El funcionamiento del framework ha resultado exitoso en todas las pruebas. Tanto las pruebas que buscaban un funcionamiento normal, como aquellas que buscaban provocar fallos, han sido superadas correctamente.

- Pruebas de funcionamiento esperado: 3 pruebas complejas realizadas y superadas.
- Pruebas de provocación de errores: 10 pruebas complejas realizadas y superadas.

Las pruebas son fácilmente reproducibles a través de la instalación y uso de la aplicación MystTest.

5. Aplicación de muestra: HelpDesk

5.1. Descripción del problema

Una empresa va a implantar un pequeño servicio de tecnologías de la información que ayude a resolver los problemas informáticos que sufran los trabajadores. Por ello han contratado varios ingenieros informáticos, los cuales han comenzado a trabajar sin ninguna herramienta específica.

En el momento en que se consulta a nuestra empresa, los informáticos han implantado un sistema en el cual los trabajadores comunican las incidencias al servicio TI por teléfono o email, utilizando programas de ofimática para gestionarlas.

En motivo de la consulta está causado por un incremento en la comunicación de problemas, lo cual ha llevado a la directiva a aceptar las demandas del servicio de TI de comprar una solución software que ayude en la gestión de las incidencias.

Tras una entrevista con el departamento de TI de la empresa, se han recabado los siguientes requisitos:

1. Necesitan una herramienta que permita el acceso de varios ingenieros a la vez, y que el acceso a los datos se realice sobre un único depósito de datos.

2. Hay dos tipos de informáticos que usarán la aplicación. Por una parte estará el ingeniero base, que tendrá una serie de incidencias asignadas que habrá de solucionar. Por otro lado hay administradores del sistema, que aparte de poder solucionar incidencias también se dedican a repartir el trabajo, o las nuevas incidencias, entre sus ingenieros.

3. Un ingeniero base podrá definir una incidencia como en proceso de resolución, o cerrarla. También podrá añadir un texto en el que defina la solución dada a la incidencia. Posteriormente, un administrador podrá evaluar la solución, y si no la ve adecuada reabrirá la incidencia.

4. Se ha insistido en que no necesitan una aplicación para introducir nuevas incidencias en el sistema. Prefieren utilizar su propio método.

5. La empresa no dispone de servidores muy potentes, por lo que nos han insistido en que es preferible que toda la lógica no esté instalada en la misma máquina.

5.2. Análisis

5.2.1. Diagrama de casos de uso

Tras el estudio de los requisitos se han encontrado diversos casos de uso, que han sido modelados en el siguiente diagrama.

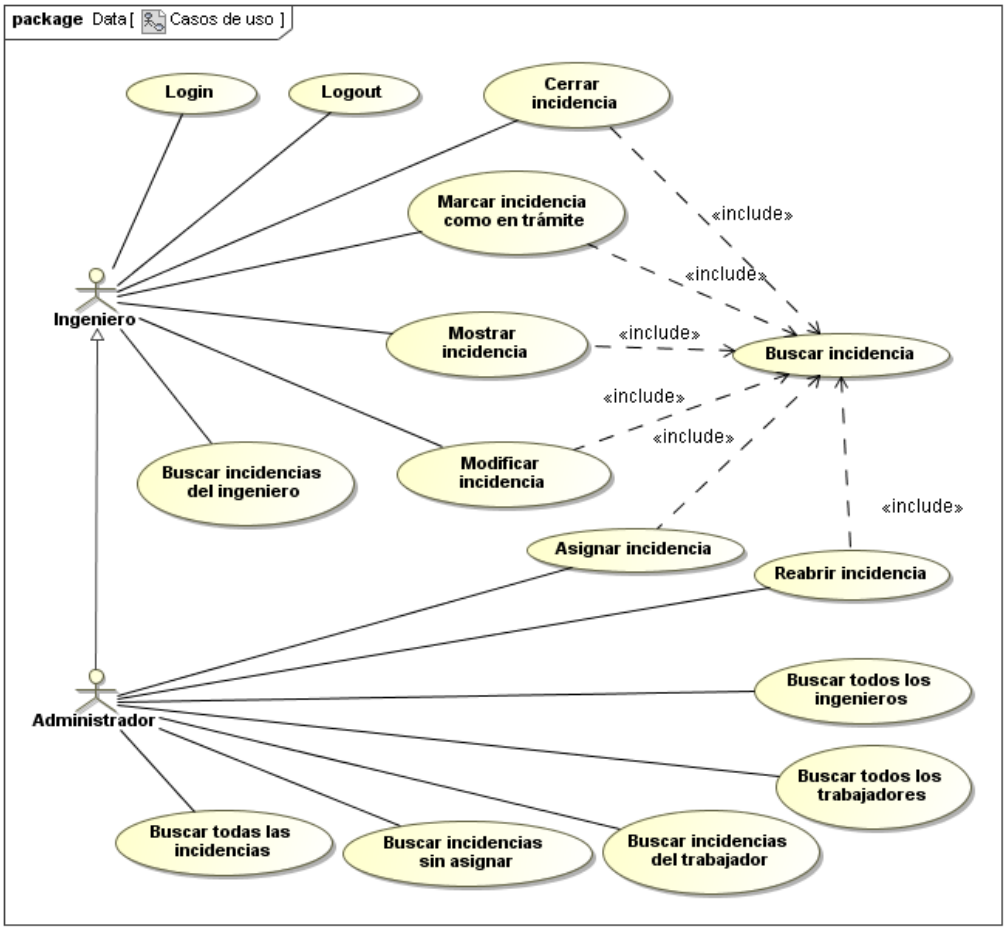


Ilustración 53: Diagrama de casos de uso

- Login y logout: servirán para identificar al ingeniero ante el sistema, y para desconectarle. Un ingeniero no identificado no podrá hacer uso del resto de funciones.
- Buscar incidencia: recupera los datos de una incidencia a través de su identificador.
- Mostrar incidencia, cerrar incidencia, marcar incidencia como en trámite, modificar incidencia (añadir el texto que describa la solución a la incidencia), asignar incidencia y reabrir incidencia: realiza la operación especificada accediendo a la incidencia a través de su identificador.

- Buscar incidencias del ingeniero: tiene dos modos de funcionamiento. Si la utiliza un ingeniero base, solo podrá buscar las incidencias que tiene asignadas. Si la utiliza un administrador, también podrá buscar las incidencias asignadas a cualquier otro ingeniero a través de su identificador.
- Buscar incidencias del trabajador: localiza las incidencias comunicadas por un trabajador a través de su identificador.
- Buscar todos los ingenieros, buscar todos los trabajadores: recupera los datos de los diferentes colectivos.
- Buscar todas las incidencias, buscar incidencias sin asignar: recupera los listados especificados.

Como se ha podido comprobar, una incidencia pasará por varios estados a lo largo de su vida, por lo resulta conveniente definirlos cuanto antes, así como las transiciones permitidas.

5.2.2. Diagrama de estados de una incidencia

Estos son los estados por los que podrá pasar una incidencia, y las transiciones que provocan los cambios.

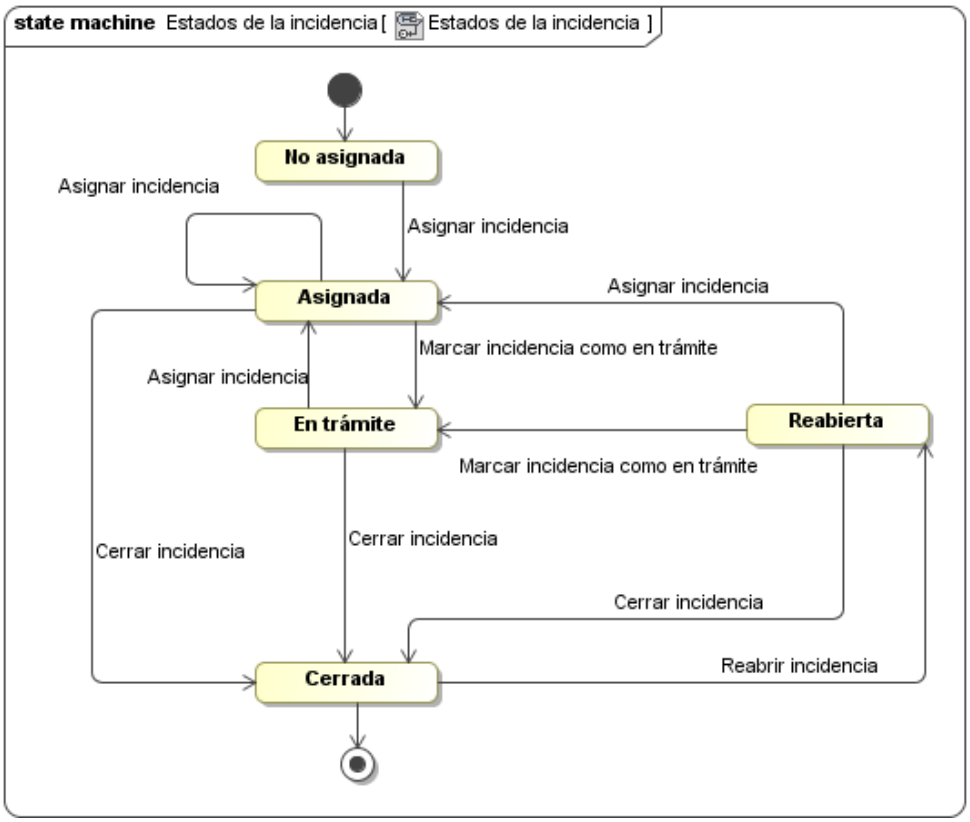


Ilustración 54: Diagrama de estados de una incidencia

Como puede observarse, hay cinco estados posibles.

- El estado “No asignada”, o inicial, es el que tiene una incidencia que ha sido comunicada, pero sobre la que no se ha realizado ningún trámite. Deberá ser asignada por un administrador a un ingeniero.
- El siguiente estado, “Asignada”, es el que tiene una incidencia que se ha encargado a un ingeniero que sea solucionada. Este estado puede alcanzarse desde muchos otros, dado que un administrador puede optar por cambiar el ingeniero encargado de tramitarla.
- El estado “En trámite” señala que el ingeniero ha leído la incidencia y ha empezado a trabajar en ella. Se puede alcanzar este estado desde “Asignada” o “Reabierta”.
- El estado “Cerrada” lo alcanza una incidencia que se da por solucionada. Puede cerrarse una incidencia desde cualquier estado, menos cuando aún no ha sido asignada.
- El último estado, “Reabierta”, tiene un carácter un poco especial. Puede ser que la solución propuesta a una incidencia no haya sido la adecuada, o que el trabajador comunique que la incidencia que comunicó realmente no ha sido solucionada. En ese caso, un administrador puede reabrir una incidencia, que automáticamente será asignada al mismo ingeniero que la solucionó.

5.2.3. Diagramas de actividades

Muchas de las funciones de la aplicación van a tener un comportamiento muy similar, y para poder observar este hecho claramente se han modelado varios diagramas de actividades.

Primero observaremos el diagrama de una acción “Login”, por ser de especial relevancia:

Actividad Login

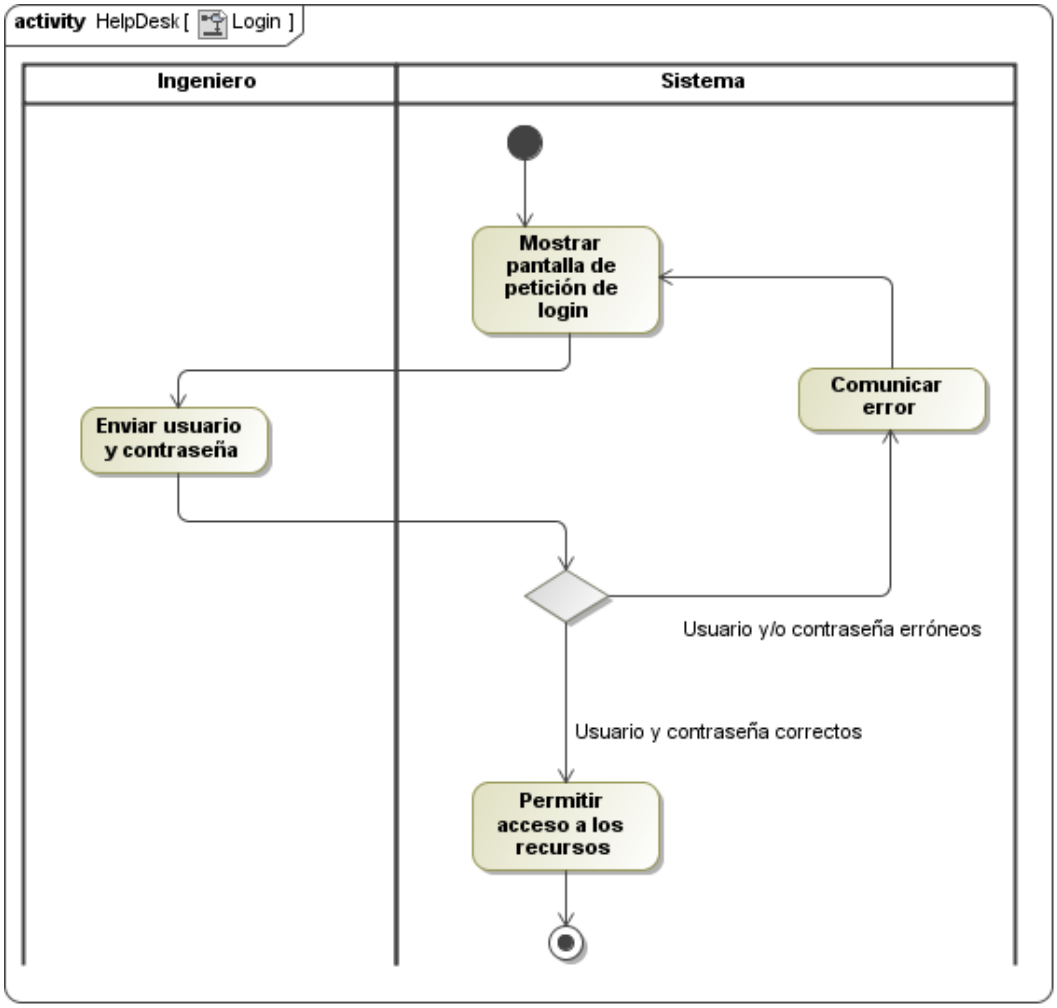


Ilustración 55: Diagrama de actividad - login

El esquema es bastante explicativo. Si el usuario y/o la contraseña no son correctas, se impide al usuario el acceso al sistema.

Seguidamente ilustraremos el diagrama de una petición para mostrar una incidencia.

Actividad: Mostrar incidencia

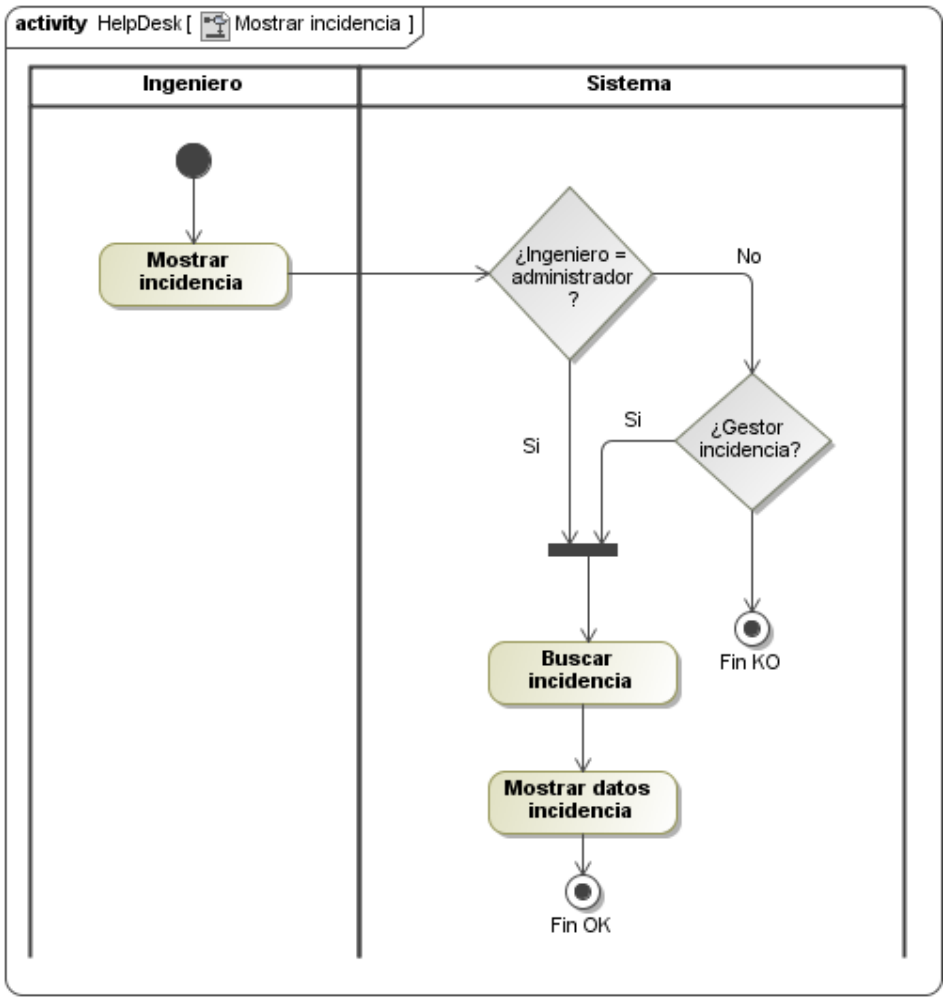


Ilustración 56: Diagrama de actividad - mostrar incidencia

Parte del diagrama será usado en varios procesos más, puesto que en varias peticiones se comprueba si el ingeniero es administrador o si ha sido asignado para gestionar la incidencia En caso afirmativo se busca la incidencia, para mostrarla, como en este caso, o para realizar operaciones sobre ella.

A continuación mostraremos dos peticiones con un comportamiento muy similar, “Marcar incidencia como en trámite” y “Cerrar incidencia”.

Actividad: Marcar incidencia como leída

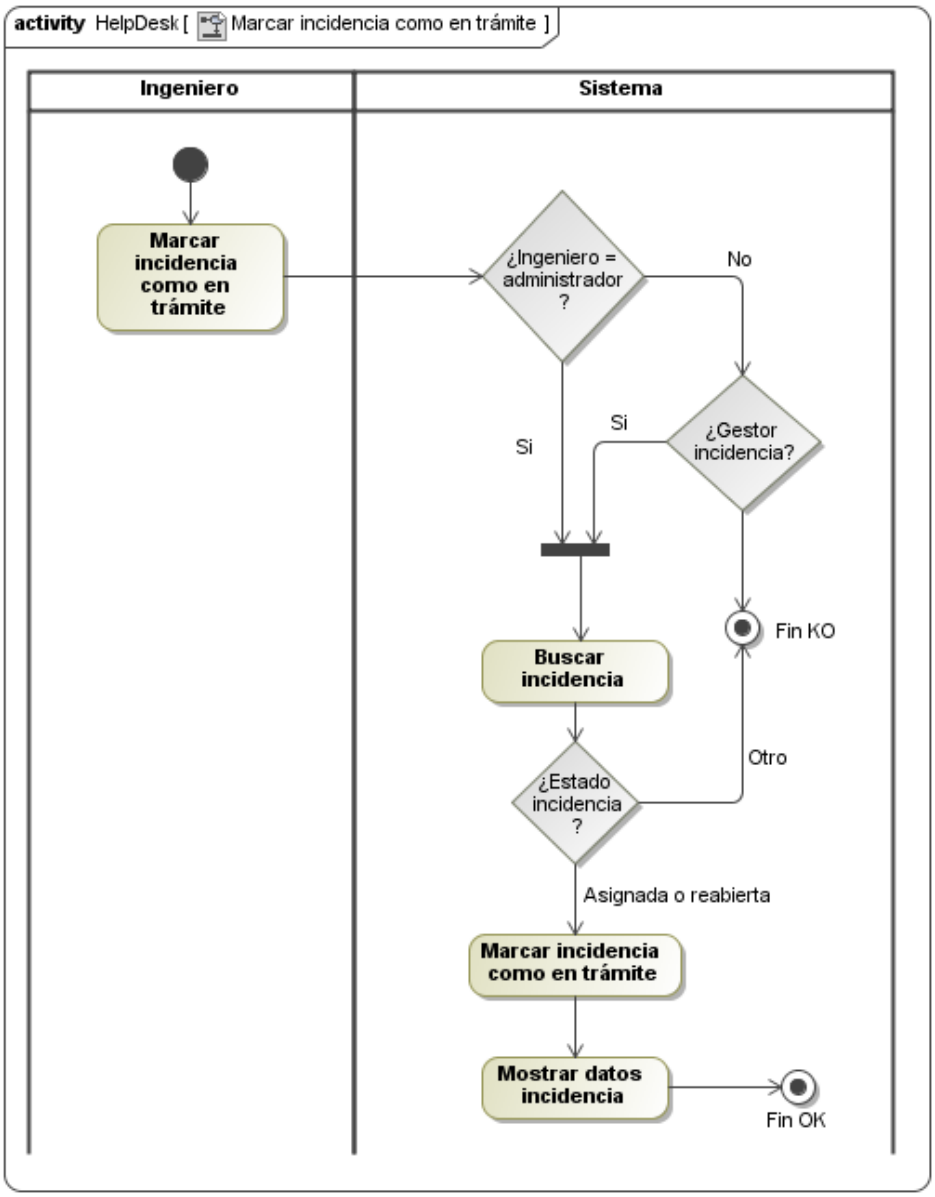


Ilustración 57: Diagrama de actividad - marcar incidencia como en trámite

Como se puede observar, “Marcar incidencia como en trámite” comparte con “Mostrar incidencia” las comprobaciones comentadas sobre el ingeniero. Además, en este caso, tras recuperar la incidencia se comprueba si su estado es “Asignada” o “Reabierta”, y solo en esos casos se resolverá la petición. Para finalizar, se mostrarán los datos de la incidencia ya modificados.

Actividad: cerrar incidencia

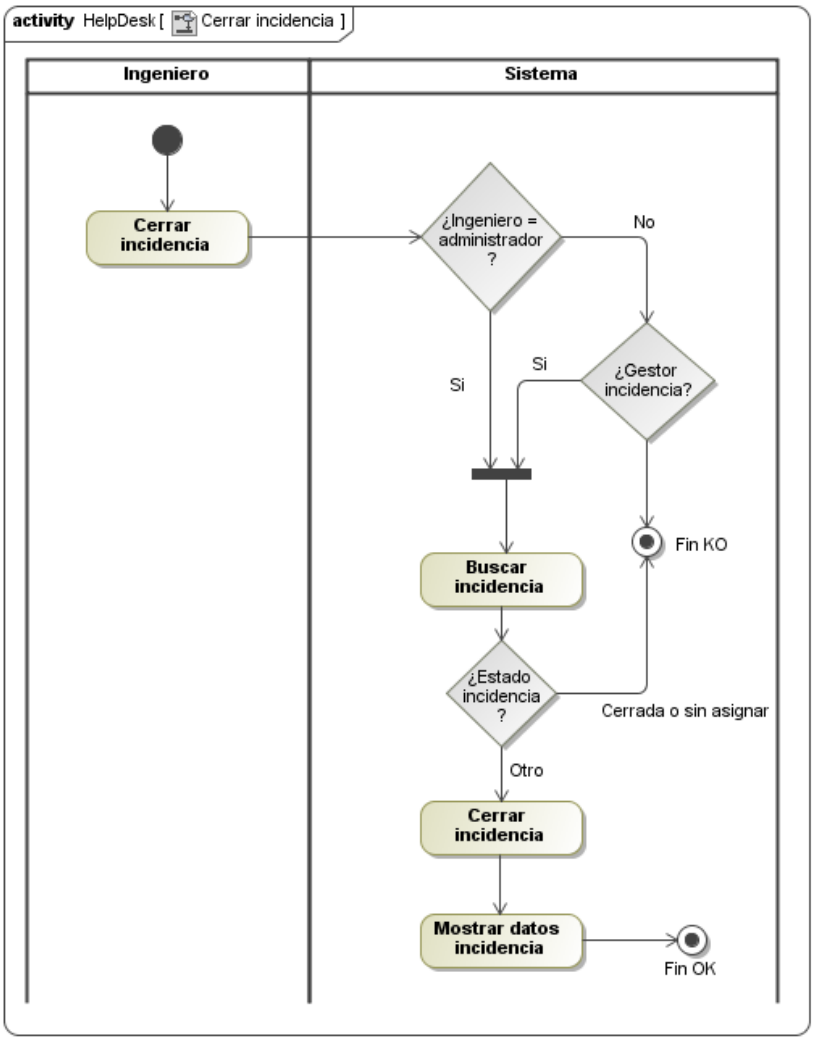


Ilustración 58: Diagrama de actividad - cerrar incidencia

Este esquema es prácticamente idéntico al de “Marcar incidencia como en trámite”, únicamente la comprobación sobre el estado de la incidencia es diferente. De la misma manera ocurre con la otra operación de ingeniero que actúa sobre una incidencia, “Modificar incidencia”. “Asignar incidencia” y “Reabrir incidencia” serán prácticamente iguales, pero no se comprobará que el ingeniero tenga la incidencia asignada, pues son operaciones de administrador. En cada uno de los casos la comprobación sobre el estado de la incidencia será el que señala el diagrama de estados previamente definido.

El resto de funciones de la aplicación (salvo logout, que resulta evidente) son exclusivas para administradores, y muy similares. Se realizarán a petición del ingeniero, se comprobará que es un administrador, y en caso afirmativo se realizará la tramita la petición. A modo de ejemplo, sirva este diagrama:

Actividad: Buscar todos los trabajadores

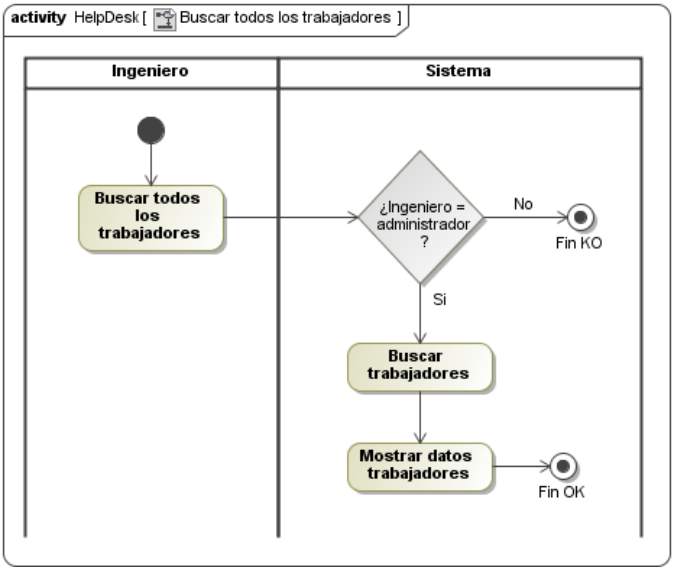


Ilustración 59: Diagrama de actividad - buscar todos los trabajadores

5.2.4. Diagrama de despliegue

Entre los requisitos identificados en la entrevista con el servicio de informática, quedó patente la reducida potencia de sus servidores, y por ello la necesidad de separar la aplicación en un número razonable de máquinas. Dado que la lógica de la aplicación es muy sencilla, y aunque habrá que estudiarlo en detalle en la etapa de diseño, se puede proponer ya una arquitectura de la aplicación por capas.

En concreto se optará por una aplicación web. Los ingenieros accederán a través de cualquier navegador, y la aplicación en sí estará repartida en tres máquinas (o más, si se dedican varias máquinas a proporcionar un mismo servicio). Una de ellas se encargará de la presentación, otra de la lógica de negocio, y la última de acceder a los datos. En el siguiente diagrama puede observarse la solución propuesta:

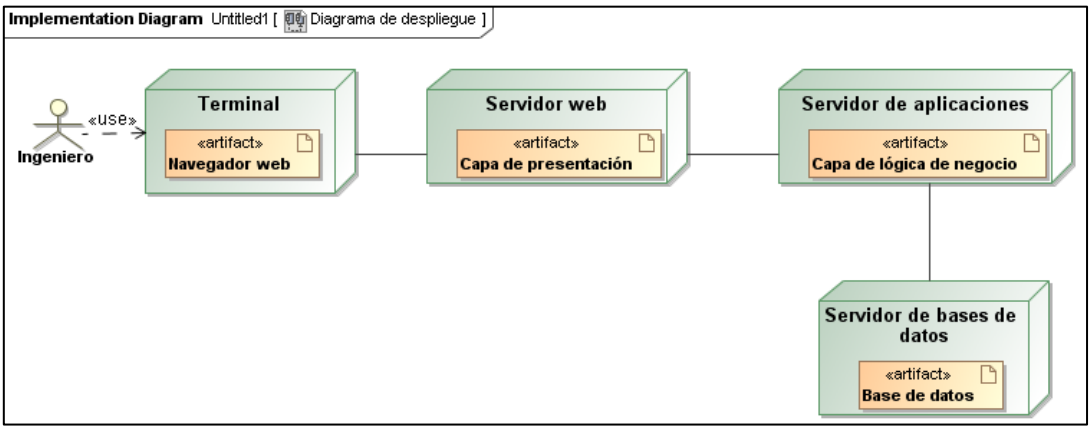


Ilustración 60: Diagrama de despliegue

5.3. Diseño

En la etapa de análisis se ha visto conveniente la creación de una aplicación por capas, que tenga varios de sus componentes diseminados en diferentes máquinas. Dado que se tratará de buscar la mayor separación posible, se ha optado por una arquitectura clásica en tres capas. Para realizar la implementación se ha pensado en la plataforma JEE, dado que aporta múltiples ventajas, como su fácil portabilidad o la gran variedad de tecnologías de apoyo, por lo que será la opción elegida.

El uso de JEE, una tecnología con cierta complejidad, hace recomendable la utilización de soluciones contrastadas para atacar la problemática. En concreto, el patrón de diseño Modelo-Vista-Controlador es una excelente solución para una de las partes más complejas de la aplicación, la separación de la presentación y la lógica de negocio.

El otro gran problema es el acceso a los datos, dado que se almacenarán de forma persistente en una base de datos relacional. Para implementar la lógica de sesión se utilizará a una de las tecnologías existentes, como Spring o EJB, que proporcionan un acceso sencillo a las API de persistencia de datos. De esta forma se evitará tener que implementar un mapeo objeto-relacional.

Dado que habrá que publicar los servicios de la lógica de negocio, y que puede haber múltiples clientes que quieran acceder a los recursos, se utilizará el patrón Fachada para crear una interfaz única de acceso. De esta forma se conseguirá una lógica del negocio portable y poco acopladas al resto del sistema

Tomadas estas decisiones, puede concretarse la arquitectura y tecnología a utilizar en cada capa:

- **Capa de presentación**

  JEE define la capa de presentación a través de un Servlet. Afortunadamente, nuestra empresa creó recientemente un framework de presentación para aplicaciones JEE, de nombre Myst, que implementa el patrón Modelo-Vista-Controlador y se adapta perfectamente a las características del problema. Myst implementará el Servlet central de la aplicación, por lo que la única tarea a desarrollar será crear los comandos que implementen la lógica, así como los formularios, vistas y filtros asociados.

- **Capa de lógica del negocio**

  Para la lógica del negocio se ha decidido utilizar la tecnología EJB 3.1, estándar en JEE, la cual permite crear beans de sesión simples, muy similares a objetos POJO, lo que facilitará en gran manera su implementación. Al poder situarse en

máquinas diferentes, cada bean creado definirá una interfaz remota que permitirá un fácil acceso a sus recursos. Dichos recursos serán accesibles gracias a la API JNDI, que permitirá buscarlos a través del nombre definido.

Estos beans publicarán una serie de métodos, los cuales podrán ser invocados por la capa de presentación, y que por su parte llamarán al gestor de la base de datos para realizar acciones sobre los datos.

- **Capa de datos**

  Los datos se almacenarán de forma persistente en una base de datos, y serán accesibles a través de la API JPA 2.0 de persistencia para aplicaciones JEE. De esta forma podremos almacenar objetos POJO de forma sencilla, gracias a su facilidad para realizar mapeo objeto-relacional. De forma específica, se utilizará la implementación de Hibernate.

5.4. Implementación

La programación ha quedado supeditada a las decisiones arquitectónicas de las etapas de análisis y diseño. Por ello, se han creado dos aplicaciones diferentes: HelpDesk-ejb para la capa de lógica del negocio, y HelpDesk-web para la capa de presentación. Las aplicaciones podrán residir en la misma máquina o en distintas máquinas, pero como entidades separadas. Se detallará a continuación la implementación de cada capa.

5.4.1. Capa de datos

Se utiliza un sistema de gestión de bases de datos relacionales, en este caso MySQL, dado que (pudiendo variar la versión) es el mismo que va a utilizar la empresa. Por ello, solamente habrá que diseñar la base de datos a utilizar y poblarla con unos datos iniciales. Tras estudiar las necesidades del problema, se ha creado la base de datos modelada en el siguiente diagrama:

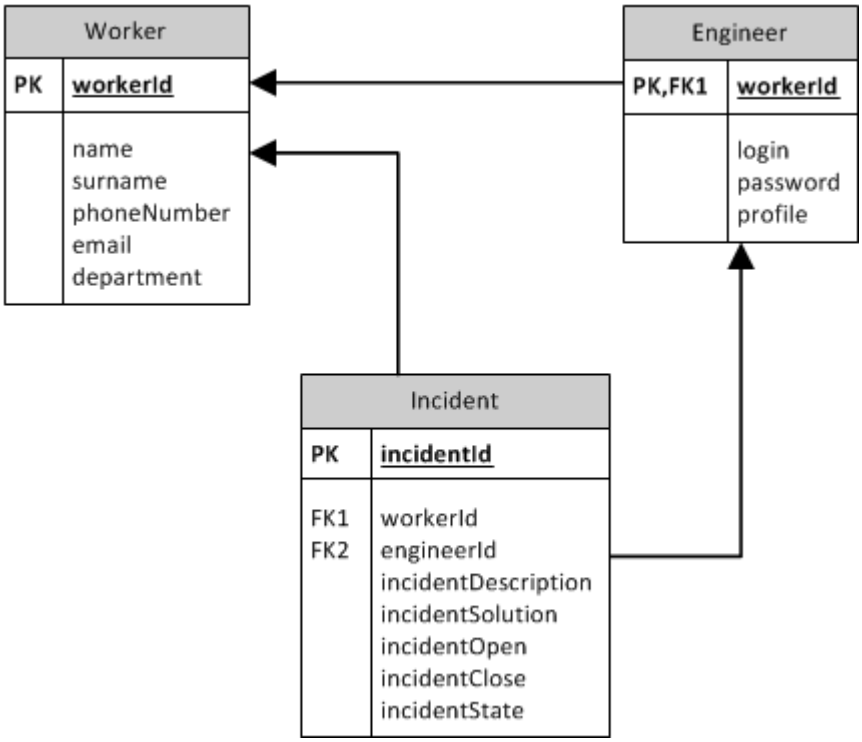


Ilustración 61: Diagrama Entidad-Relación de la base de datos

Como se puede ver es una base de datos muy sencilla. Simplemente hay que destacar que Engineer tiene una única clave foránea que referencia la clave principal de Worker, workerId. Incident tiene dos claves foráneas: workerId referencia a la clave principal de Worker, y engineerId referencia la clave principal de Engineer.

5.4.2. Capa de lógica del negocio

En esta capa se encontrará la aplicación de sesión HelpDesk-ejb. Dada la sencillez de las operaciones sobre los datos, se ha implementado un único bean de sesión, cuyos métodos serán accesibles a través de una interfaz remota. Además, se define una clase para cada entidad de la base de datos. Estas clases estarán separadas en dos paquetes, como se puede ver en la siguiente imagen:

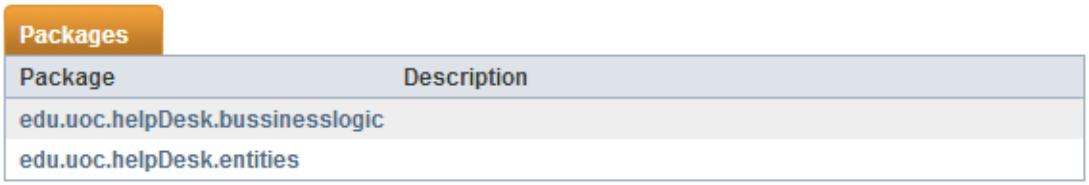


Ilustración 62: Paquetes de la aplicación HelpDesk-ejb

Con la aplicación se distribuyen también dos archivos de especial relevancia:

- persistence.xml: este archivo define la configuración de acceso a la base de datos a través de JPA. Disponible en la carpeta “src/main/resources/META-INF” del código fuente.
- help_desk.sql: archivo con unos pocos datos para cargar en el gestor de bases de datos y poder comprobar la funcionalidad. Disponible en la carpeta “sql” del código fuente de HelpDesk-ejb, además de como archivo separado.

Entities

Class Summary

Class	Description
Engineer	Clase que almacena una entidad de datos de tipo Engineer o Ingeniero.
Incident	Clase que almacena una entidad de datos de tipo Incident o incidente.
Worker	Clase que almacena una entidad de datos de tipo Worker o Trabajador.

Ilustración 63: Clases del paquete Entities

Con respecto a las entidades hay poco que destacar. Siguen el estándar JPA, y utilizan la estrategia “eager” a la hora de recuperar datos de la BBDD. De esta forma, cuando se recuperen los datos de una entidad que tiene una clave foránea también se recuperarán los datos de la entidad asociada. Este comportamiento responde a las necesidades de la aplicación, que necesitarán acceder en cada petición a los datos relacionados.

Bussinesslogic

Interface Summary

Interface	Description
HelpDeskEngineer	Interfaz remoto de acceso a los métodos de la lógica de negocio del programa HelpDesk.

Class Summary

Class	Description
HelpDeskEngineerBean	Bean de sesión que implementa los métodos de la lógica de negocio.

Ilustración 64: Clases del paquete Bussinesslogic

-Interface HelpDeskEngineer

Interfaz remoto de acceso a los métodos de la lógica del negocio. Esta clase se compilará en la biblioteca "HelpDesk-ejb-client.jar", la cual deberán utilizar las aplicaciones que quieran utilizar los recursos de este bean de sesión.

-HelpDeskEngineerBean

Bean de sesión que propiamente implementa los métodos de la lógica del negocio. Estos métodos son un calco de los casos de uso, por lo que no se ve necesaria una mayor explicación. En cualquier caso se invita a consultar el código fuente o el Javadoc, ampliamente documentados.

5.4.3. Capa de presentación

Esta capa proporciona la aplicación web HelpDesk-web. Como ya se comentó en la etapa de diseño, se ha optado por implementar la capa de presentación utilizando el framework Myst. Este ya ha sido suficientemente explicado en el apartado 4 de esta memoria, por lo que este apartado se centrará a detallar las nuevas clases creadas para dotar de funcionalidad a la aplicación.

Packages

Package	Description
edu.uoc.helpDesk.command	
edu.uoc.helpDesk.exception	
edu.uoc.helpDesk.exception.handler	
edu.uoc.helpDesk.filter	
edu.uoc.helpDesk.form	
edu.uoc.helpDesk.utility	

Ilustración 65: Paquetes de la aplicación HelpDesk-web

Además de las clases, hay que destacar el uso que la aplicación hace de ciertos archivos:

- web.xml: archivo de configuración del contenedor web. Define a Myst como el Servlet a utilizar cuando las peticiones web terminen en ".do".
- HelpDesk.xml: Es el archivo de configuración de la aplicación, que sigue la especificación del esquema "MystSchema.xsd". Relaciona peticiones con comandos, formularios, filtros, vistas y gestores de errores. Dada su longitud

no se reproducirá en esta memoria, sino que se invita a consultar el archivo directamente (carpeta “src/main/resources” del código fuente).

- HelpDeskBundle.properties: bundle con los diversos idiomas seleccionables en la aplicación, disponible en la carpeta “src/main/resources/bundles”. Se suministra en inglés y español.
- jboss-ejb-client.properties: define la localización del servidor de aplicaciones donde está desplegado el bean de sesión. Disponible en la carpeta “src/main/resources”.

Command

Class Summary

Class	Description
AssignIncidentCommand	Comando que asigna una incidenca a un ingeniero.
ChangeLanguageCommand	Coamndo que cambia la región de la sesión, para mostrar la apliación en el idioma adecuado.
CloseIncidentCommand	Comando que cierra una incidencia y le asigna como fecha de cierre la hora actual del servidor.
FindAllEngineersCommand	Comando que obtiene todos los ingenieros.
FindAllIncidentsCommand	Comando que obtiene todas las incidencias.
FindAllWorkersCommand	Comando que obtiene todos los trabajadores.
FindIncidentsEngineerCommand	Comando que obtiene las incidencias asignadas a un ingeniero.
FindIncidentsWorkerCommand	Comando que obtiene las incidencias comunicadas por un trabajador.
FindUnassignedIncidentsCommand	Comando que obtiene todas las incidencias sin un ingeniero asignado.
LoginCommand	Comando que permite a un ingeniero identificarse ante la aplicación (realizar un login) para poder utilizar sus servicios..
LogoutCommand	Comando que permite a un ingeniero identificarse ante la aplicación (realizar un login) para poder utilizar sus servicios..
ModifyIncidentCommand	Comando que modifica una incidencia, asignando una propuesta de solución.
ReopenIncidentCommand	Comando que reabre una incidencia.
ShowIncidentCommand	Comando que mustra una incidencia.
WorkIncidentCommand	Comando que modifica una incidencia, marcándola como en proceso de resolución.

Ilustración 66: Clases del paquete Command

Los comandos implementan con total fidelidad los casos de uso descritos en la fase de análisis. Cada uno de ellos llamará a uno o más métodos del bean de sesión, con lo que recuperará o modificará los datos necesarios de la base de

datos. Las comprobaciones sobre la identidad del ingeniero han sido eliminadas de los comandos, convirtiéndolas en filtros previos a la ejecución.

Form

Class Summary	
Class	**Description**
AssignIncidentForm	Formulario que almacena los identificadores de una incidencia y del ingeniero al cual se va a asignar su gestión.
FindIncidentsEngineerForm	Formulario que almacena el identificador de un ingeniero del cual se van a solicitar las incidencias que tiene asignadas.
FindIncidentsWorkerForm	Formulario que almacena el identificador de un trabajador del cual se van a solicitar las incidencias que ha comunicado.
LocaleForm	Formulario que almacena la región de un petición de cambio de idioma.
LoginForm	Formulario que almacena el nombre de usuario y la contraseña de una petición de login.
ModifyIncidentForm	Formulario que almacena la solución propuesta a una incidencia.
SelectIncidentForm	Formulario que almacena el identificador de una incidencia, sobre la cual se podrán realizar varias operaciones.

Ilustración 67: Clases del paquete Form

Los formularios, como ya se ha comentado en repetidas ocasiones, almacenan los parámetros de las peticiones del ingeniero, gracias a la inversión de control del framework. Dado que varias peticiones comparten parámetros, el formulario SelectIncidentForm será utilizado por varios comandos.

Filter

Class Summary	
Class	**Description**
IsEngineerOrRootFilter	Filtro de entrada que comprueba que el ingeniero que realiza una petición sobre un recurso es propietario del mismo, o bien es administrador.
IsOwnerOrRootFilter	Filtro de entrada que comprueba que el ingeniero que realiza una petición sobre una incidencia es el gestor de la misma, o bien es administrador.
IsRootFilter	Filtro de entrada que comprueba que el usuario es administrador.
ValidateLoginFilter	Filtro de entrada que comprueba que los datos de entrada del login son válidos.

Ilustración 68: Clases del paquete Filter

Como se ha comentado al hablar de los comandos, se han creado filtros para controlar el acceso a los métodos de la lógica de negocio.

- ValidateLoginFilter

Comprueba que los parámetros de una petición de login (nombre de usuario y contraseña) cumplen con unos requisitos que se han preestablecido. Básicamente, la longitud de ambos debe tener entre 6 y 20 caracteres, y el nombre de usuario debe comenzar por una letra. En caso de fallo del filtro se muestra por pantalla un mensaje de error genérico ("nombre de usuario o contraseña erróneos"), para no dar pistas a un posible intento de ataque de suplantación de identidad.

- IsRootFilter, IsOwnerOrRootFilter, IsEngineerOrRootFilter

Realizan las comprobaciones especificadas en la imagen. Estos filtros realmente existen para evitar accesos no autorizados a métodos del negocio, por lo que en caso de fallar no mostrarán un mensaje de error; de hecho, nos mostrarán absolutamente nada, únicamente la pantalla de inicio.

IsOwner e IsEngineer son muy similares. La diferencia radica en que IsEnginer dispone entre los datos pasados por parámetro del identificador del ingeniero asignado a la incidencia, mientras que IsOwner dispone del identificador de la incidencia, por lo que ha de buscar el identificador del ingeniero de la lógica de negocio.

Exception

Exception Summary

Exception	Description
NoRootException	Excepción causada cuando un ingeniero trata de acceder a recursos exclusivos para administradores.
WrongEngineerAccessException	Excepción causada cuando un ingeniero trata de acceder a recursos sobre los cuales no tiene responsabilidad ni permisos.
WrongIncidentAccessException	Excepción causada cuando un ingeniero trata de acceder a una incidencia sobre la cual no responsabilidades, ni permisos.

Ilustración 69: Clases del paquete Exception

Excepciones para manejar accesos no autorizados a recursos de la lógica de negocio. Son disparadas por los filtros definidos en la aplicación, y capturadas por los gestores de excepciones.

ExceptionHandler

Class Summary	
Class	Description
AccessExceptionHandler	Gestor de excepciones para excepciones de acceso erróneo.
LoginExceptionHandler	Gestor para excepciones de login.

Ilustración 70: Clases del paquete ExceptionHandler

Gestionan las excepciones causadas por un fallo en la ejecución de los filtros. Como se ha comentado, no realizarán apenas operaciones, pero sí asegurarán que la aplicación mantiene el comportamiento esperado.

Utility

Class Summary	
Class	Description
BeanAccess	Clase que proporciona acceso a la interfaz remota del bean de sesión HelpDeskEngineer.

Ilustración 71: Clase del paquete Utility

Clase que configura el acceso al bean de sesión a través del nombre JNDI de su interfaz remota.

La aplicación se distribuye del mismo modo en que ha podido se programada y comprobada, en una sola máquina, con el bean de sesión y la aplicación web desplegados por separado en un mismo servidor de aplicaciones JBoss. En caso de realizarse de forma diferente (por ejemplo, mediante un archivo EAR) habrá que modificar el nombre JNDI.

JSPs

Aunque no sea un paquete, no menos importantes son las páginas JSP que presentan las vistas.

Página	Descripción
header.jsp	Cabecera utilizada por las demás páginas.
leftMenu.jsp	Menú lateral utilizado por las demás páginas.
internationalError.jsp	Página que muestra los mensajes de error regionalizados.
blankList.jsp	Página utilizada cuando una petición de un listado devuelve un resultado vacío.
login.jsp	Página de login. No utiliza la misma cabecera que el resto de páginas, puesto que le haría entrar en un bucle de redirecciones.
menu.jsp	Página de inicio. Si el usuario no está identificado, la cabecera le reenviará a la página de login. Esta página de inicio simplemente muestra la cabecera y el menú lateral.
showEngineers.jsp	Página que muestra una tabla con los datos de los ingenieros. Permite mostrar las incidencias que tiene asignadas cada uno de ellos.
showWorkers.jsp	Página que muestra una tabla con los datos de los trabajadores. Permite mostrar las incidencias que ha comunicado cada uno de ellos.
showIncidents.jsp	Página que muestra una tabla con las incidencias que se hayan solicitado (las comunicadas por un trabajador, las asignadas a un ingeniero, las no asignadas o todas). Permite también seleccionar una incidencia en particular para verla en detalle.
showIncident.jsp	Página que muestra los datos de una incidencia, y permite gestionarla. Dependiendo de si el ingeniero es un administrador o un ingeniero normal, tendrá diferentes opciones.

Ilustración 72: Páginas JSP de la aplicación

- header y leftMenu siguen el patrón Composite View, al ser simplemente componentes de las otras páginas. header muestra la cabecera, con la selección del idioma, etc., y leftMenu el menú lateral que se utilizará en la interacción con el sistema.
- internationalError es la página de error proporcionada por el framework Myst, la cual se ha modificado para incluir la cabecera y el menú.
- login es la página de identificación, que por motivos de lógica interna no utiliza la cabecera.
- menu es la página inicial, que simplemente muestra el menú. Los usuarios normalmente interpretarán que la página de inicio es la de login, pues un usuario no validado es automáticamente redirigido a ella.

- showEngineers y showWorkers páginas exclusivas para administradores, que muestran los ingenieros o los trabajadores de la aplicación.
- showincidents muestra los resultados tabulados de una de las variadas peticiones de listado de incidencias.
- showIncident presenta la información asociada a una petición, así como diversos botones para gestionarla. Dependiendo de si el usuario es o no administrador, y también según el estado de la incidencia, presentará unas u otras opciones.

5.5. Manual

5.5.1. Instalación de HelpDesk

HelpDesk se distribuye como dos aplicaciones compiladas separadas, una aplicación web y otra de sesión, además de un script con las sentencias SQL necesarias para crear la base de datos. Ambas aplicaciones también se distribuyen en su código fuente como proyectos Maven.

Base de datos

Para la implementación se ha utilizado un gestor de bases de datos MySQL, aunque cualquier otro gestor de bases de datos relacionales debería servir. Debe darse un nombre a la base de datos (en el ejemplo es “ejb3”) y cargar las tablas proporcionadas en el archivo “help_desk.sql”.

Aplicación de sesión – HelDesk-ejb

La aplicación de sesión o lógica de negocio se distribuye compilada en dos bibliotecas JAR. Una de ellas, “helpDesk-EJB-1.0.jar”, contiene realmente la lógica, y debe ser desplegada en un servidor de aplicaciones JBoss. La otra biblioteca, “helpDesk-EJB-1.0-client.jar”, debe ser añadida a las bibliotecas utilizadas por la aplicación cliente para poder acceder a los servicios de la lógica de negocio.

Además es necesario definir correctamente el acceso a la base de datos a través del archivo “persistence.xml” que se despliega en el directorio META-INF de la aplicación. Su elemento “jta-data-source” debe tener el mismo valor que el elemento “jndi-name” de la fuente de datos definida en el archivo de configuración “standalone.xml” de JBoss (en la versión distribuida tiene el valor “java:/MySqlDS”).

Aplicación web – HelpDesk-web

La aplicación web se distribuye compilada en forma de archivo WAR, el cual puede ser desplegado en el mismo servidor de aplicaciones JBoss, aunque también debiera funcionar en cualquier servidor web, como Tomcat. En cualquier caso es necesario configurar adecuadamente el archivo "jboss-ejb-client.properties", que se despliega en la carpeta "WEB-INF/clases" de la aplicación. En él es necesario especificar la URL y el puerto en los que está instalada la aplicación de sesión.

Importación del código fuente

Si se prefiere importar el código fuente, también habrá que tener en cuenta que consiste en dos proyectos Maven.

- HelpDesk-ejb obtendrá todas las dependencias directamente de los repositorios.
- HelpDesk-web tendrá que añadir la biblioteca "helpDesk-EJB-1.0-client.jar" para poder utilizar los métodos de la aplicación de sesión, y la biblioteca "Myst-1.0.jar", para disponer del framework. Estas bibliotecas deben añadirse a la carpeta de repositorio de la versión de Maven que se esté utilizando.

Ambos proyectos pueden compilarse con las opciones "install" o "package" de Maven.

5.5.2. Configuración de JBoss

La aplicación de sesión mantiene una fuerte dependencia del servidor de aplicaciones donde está instalada, por lo cual hay que configurar adecuadamente dicho servidor para poder acceder a la capa de datos. Se ha optado por utilizar JBoss AS 7, el cual tiene unas necesidades específicas que se describirán a continuación.

1. Instalar el conector a la base de datos.

   Para cada gestor de base de datos a la que se quiera acceder hay que instalar un conector. En el ejemplo se describe cómo acceder a un gestor de base de datos MySQL 5.6.15; en caso de querer utilizar otro gestor diferente habrá que instalar el conector adecuado y variar determinados parámetros.

   Se descarga el conector "MySQL connector Java 5.1.27" y se descomprime en la carpeta "modules/com/mysql/main" de JBoss.

2. Crear el archivo “module.xml” en la misma carpeta “modules/com/mysql/main”, con el siguiente contenido:

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:module:1.0" name="com.mysql">
  <resources>
    <resource-root path="mysql-connector-java-5.1.27-bin.jar"/>
  </resources>
  <dependencies>
    <module name="javax.api"/>
  </dependencies>
</module>
```

Ilustración 73: Configuración de module.xml

3. Añadir al archivo “standalone.xml” de la carpeta “standalone/configuration” de JBoss una referencia al nuevo conector en el elemento <drivers>:

```
<driver name="h2" module="com.h2database.h2">
        <xa-datasource-class>
            org.h2.jdbcx.JdbcDataSource
        </xa-datasource-class>
    </driver>
    <driver name="mysqlDriver" module="com.mysql">
        <xa-datasource-class>
            com.mysql.jdbc.Driver
        </xa-datasource-class>
    </driver>
```

Ilustración 74: Configuración de driver en standalone.xml

Añadir en el mismo archivo un nuevo origen de datos, dentro del elemento <datasources>:

```
<datasource jndi-name="java:/nombre_JNDI"
    pool-name="MySQLDS" enabled="true" use-java-context="true">
    <connection-url>
        jdbc:mysql://URL_BBDD:puerto/nombre_de_la_base_de_datos
    </connection-url>
    <driver>mysqlDriver</driver>
    <security>
        <user-name>nomre_de_usario_de_la_BBDD</user-name>
        <password>contraseña_de_la_BBDD</password>
    </security>
</datasource>
```

Ilustración 75: Configuración de origen de datos en standalone.xml

"jndi-name" debe tener el mismo valor que el elemento "jta-data-source" del archivo "persistence.xml" (en la versión distribuida "java:/MySqlDS").

5.5.3. Manual de usuario

Al conectarnos a la aplicación sin haber iniciado sesión, aparece la pantalla de login:

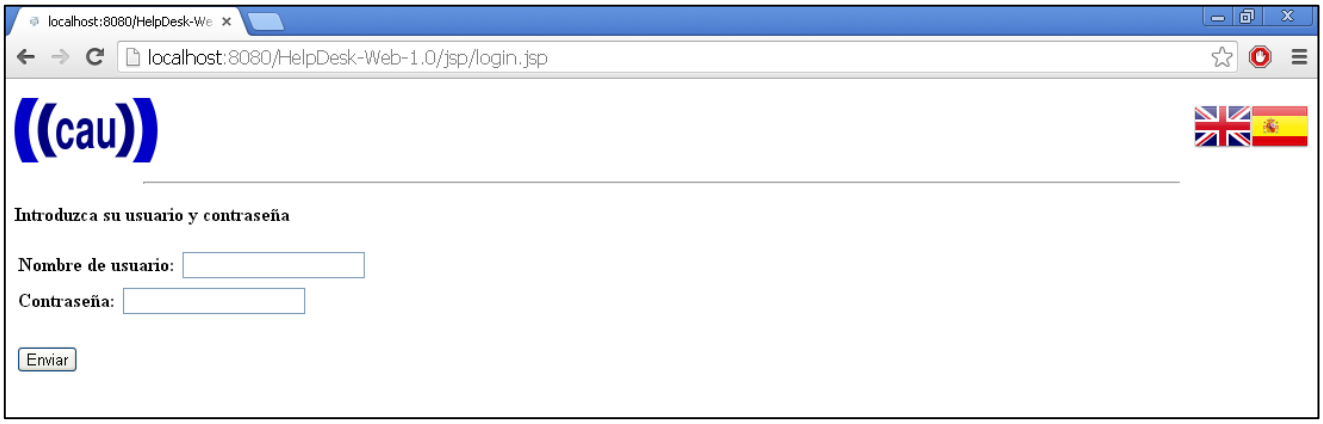


Ilustración 76: HelpDesk - pantalla de login

En ella podemos ver dos zonas. La cabecera, con el logo del programa, que será un enlace a la pantalla de inicio, y dos banderas para seleccionar el idioma de la aplicación. Al seleccionar un idioma el sistema también nos enviará a la pantalla de inicio. La otra zona permite identificar al usuario en el sistema. Si introducimos un usuario y contraseña incorrectos, el sistema lo notificará:

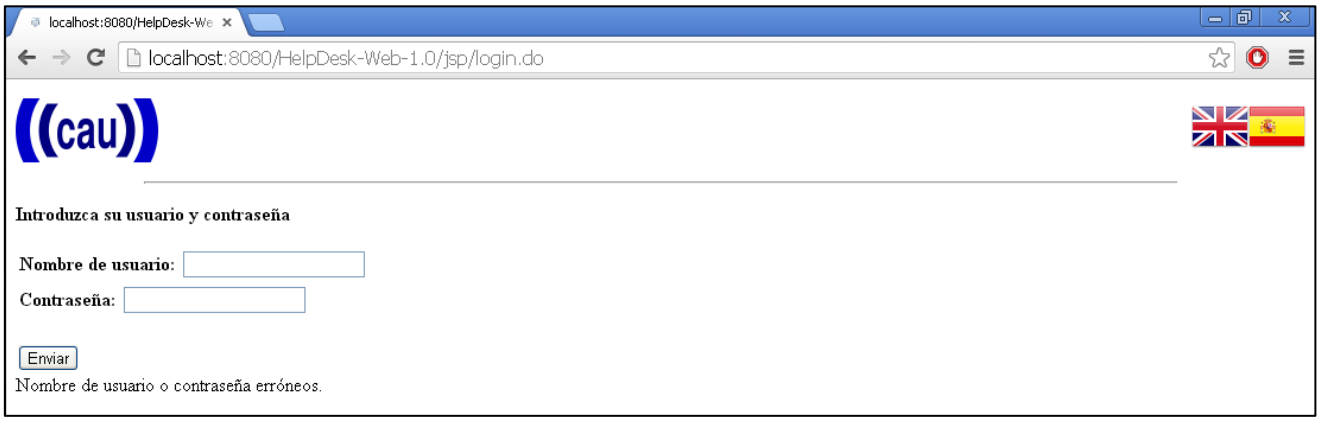


Ilustración 77: HelpDesk - fallo de login

Si nos identificamos correctamente, por ejemplo con los datos de la ingeniera Elisabeth Puertas, escribiremos "lizbel" como nombre de usuaria y contraseña. Nos recibirá la pantalla de menú:

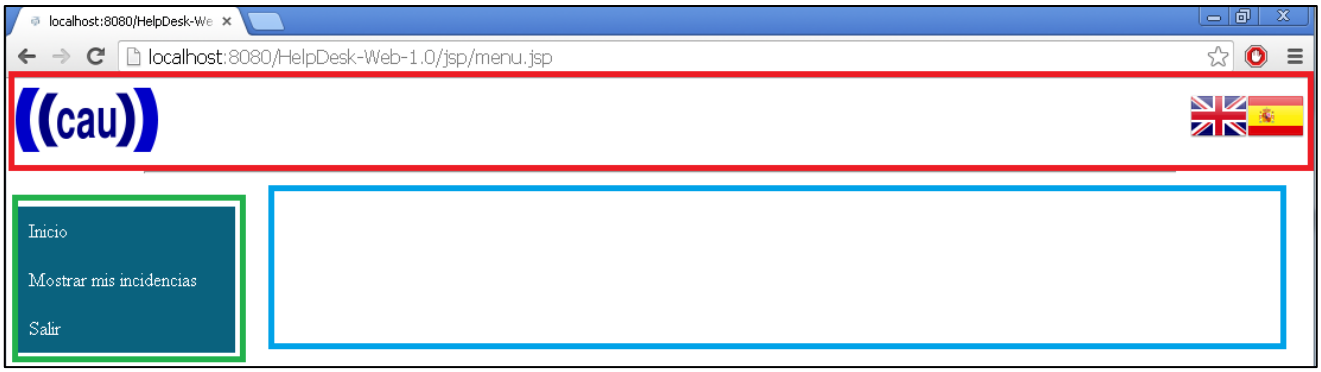


Ilustración 78: HelpDesk - pantalla de menú ingeniero

Se identifican tres zonas, que aparecerán en el resto del programa: la cabecera, el menú lateral y el área principal. Como la usuaria es una ingeniera normal, el menú sólo le permite consultar las incidencias que tiene asignadas, ir a la pantalla de inicio o salir de la aplicación (logout). Supongamos que consulta sus incidencias:

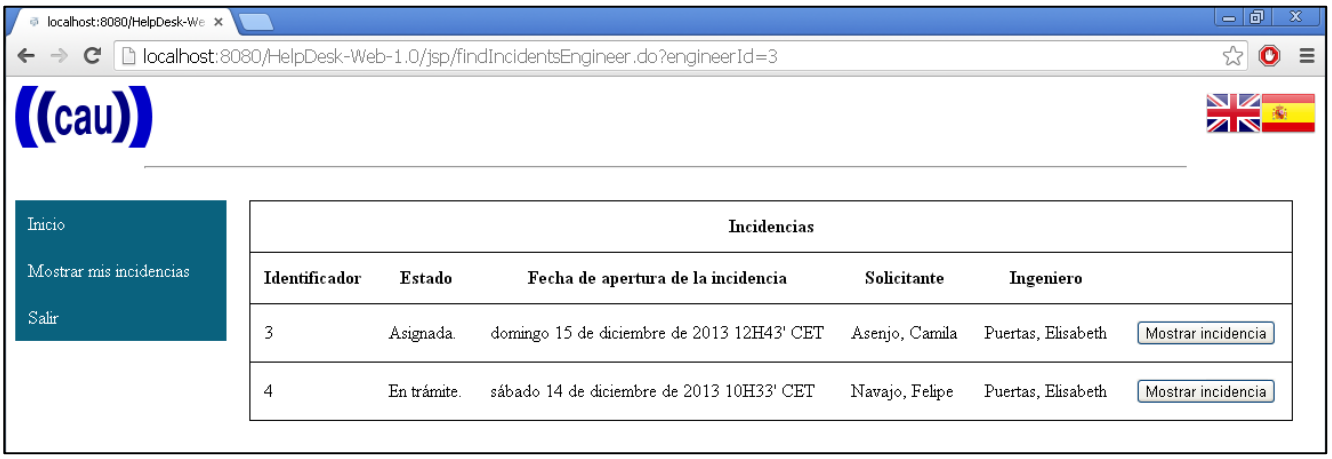


Ilustración 79: HelpDesk - pantalla de mostrar incidencias

Como puede comprobarse, se muestra una tabla con un resumen de cada incidencia. Esta pantalla será la misma para mostrar las incidencias asignadas a un ingeniero, las comunicadas por un trabajador, las no asignadas o todas las incidencias. En cualquier caso lo que puede hacerse es seleccionar una incidencia individual. Por ejemplo, la incidencia con identificador 3.

Ilustración 80: HelpDesk - pantalla de incidencia para una ingeniera

El área principal en este caso da una información completa sobre la incidencia, así como acerca de la solicitante, para que la ingeniera pueda contactar con ella. A la petición se le ha aplicado el filtro de salida internationalFormat, por lo que el formato de número y fechas está automáticamente ajustado a la zona seleccionada (España, en este caso).

La pantalla permite además realizar operaciones sobre la incidencia, pero únicamente las autorizadas para la ingeniera y el estado de la incidencia ("asignada", en este caso). Es decir, puede realizar las operaciones "marcar como en trámite", "marcar como cerrada" o "modificarla para introducir una solución" (botón "actualizar"). Por ejemplo, si se marca como "en proceso":

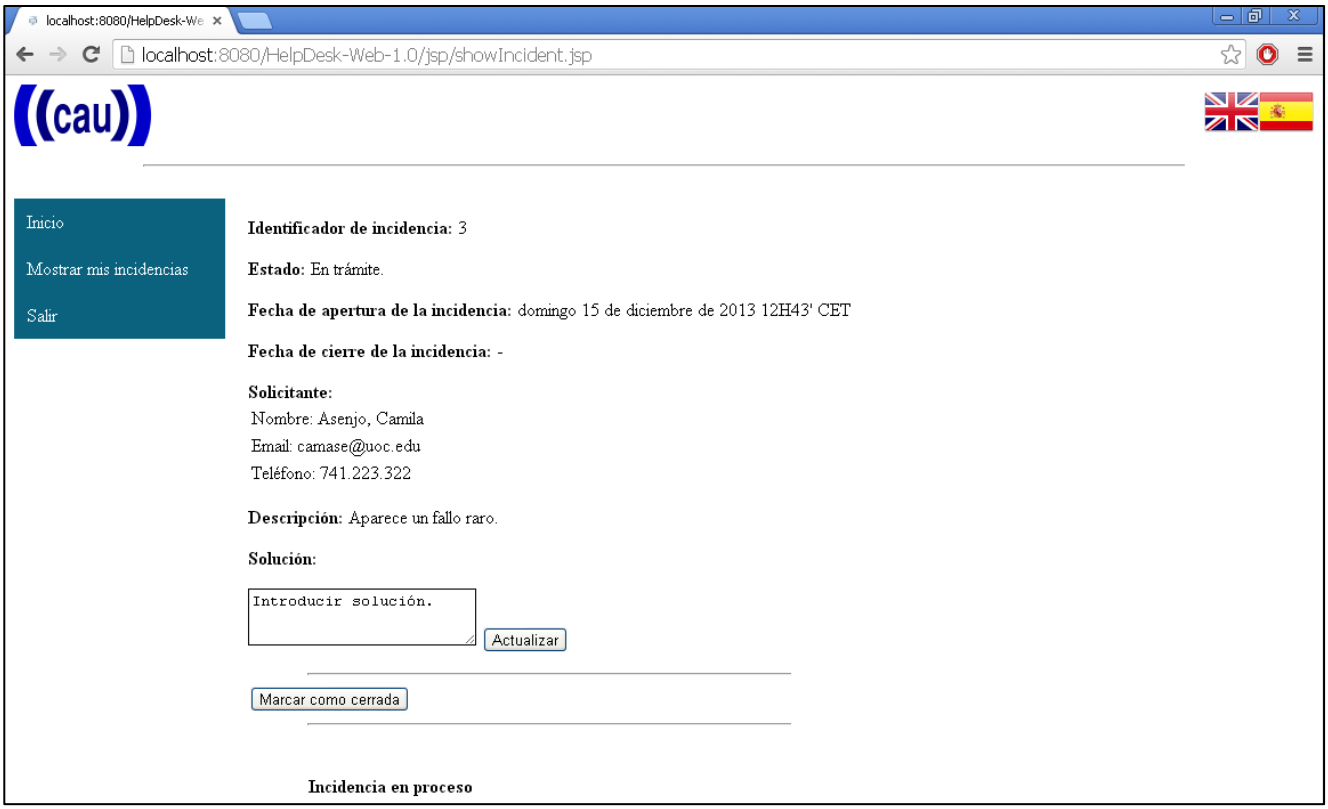


Ilustración 81: HelpDesk - pantalla de incidencia marcada como como en proceso

El estado de la incidencia cambia, así como las operaciones realizables sobre la misma. Además, al pie del área central se notifica el cambio realizado.

Salgamos de la sesión y entremos con los datos de un administrador, por ejemplo de Miguel Souto, con nombre de usuario y contraseña “msoutob”. Inmediatamente pulsaremos sobre la bandera inglesa para cambiar la lengua. Esta es la nueva pantalla con el menú de un administrador en inglés:

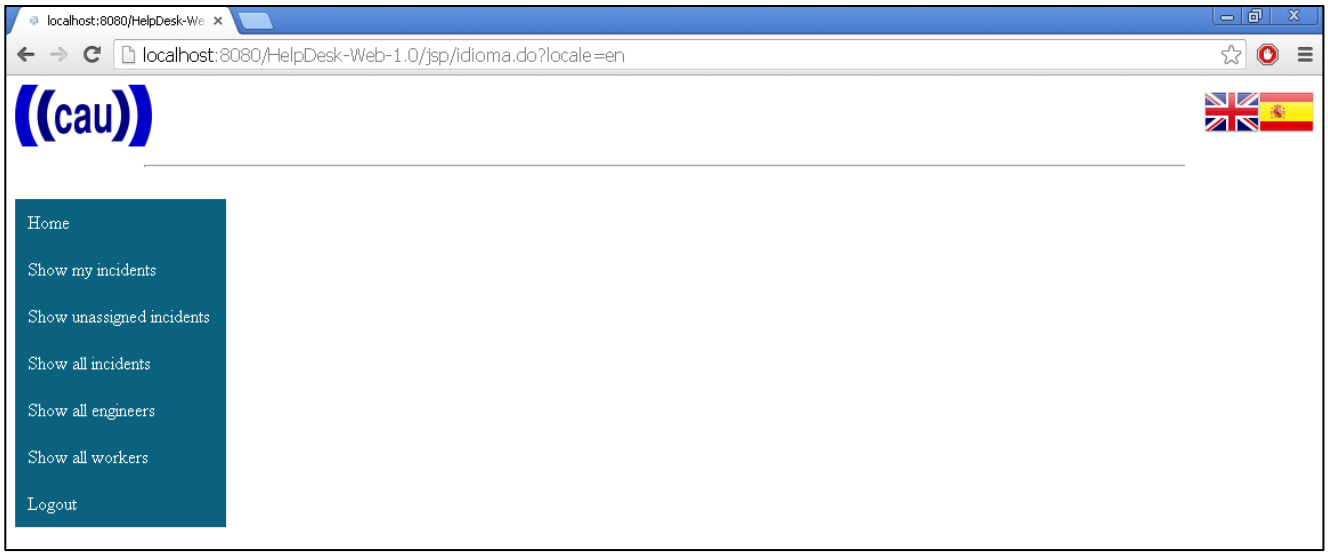


Ilustración 82: HelpDesk - pantalla inicial en inglés para administradores

Vamos a acceder a la incidencia 3, pero a través de la trabajadora que la comunicó. Primero seleccionamos mostrar a todos los trabajadores ("show all workers"):

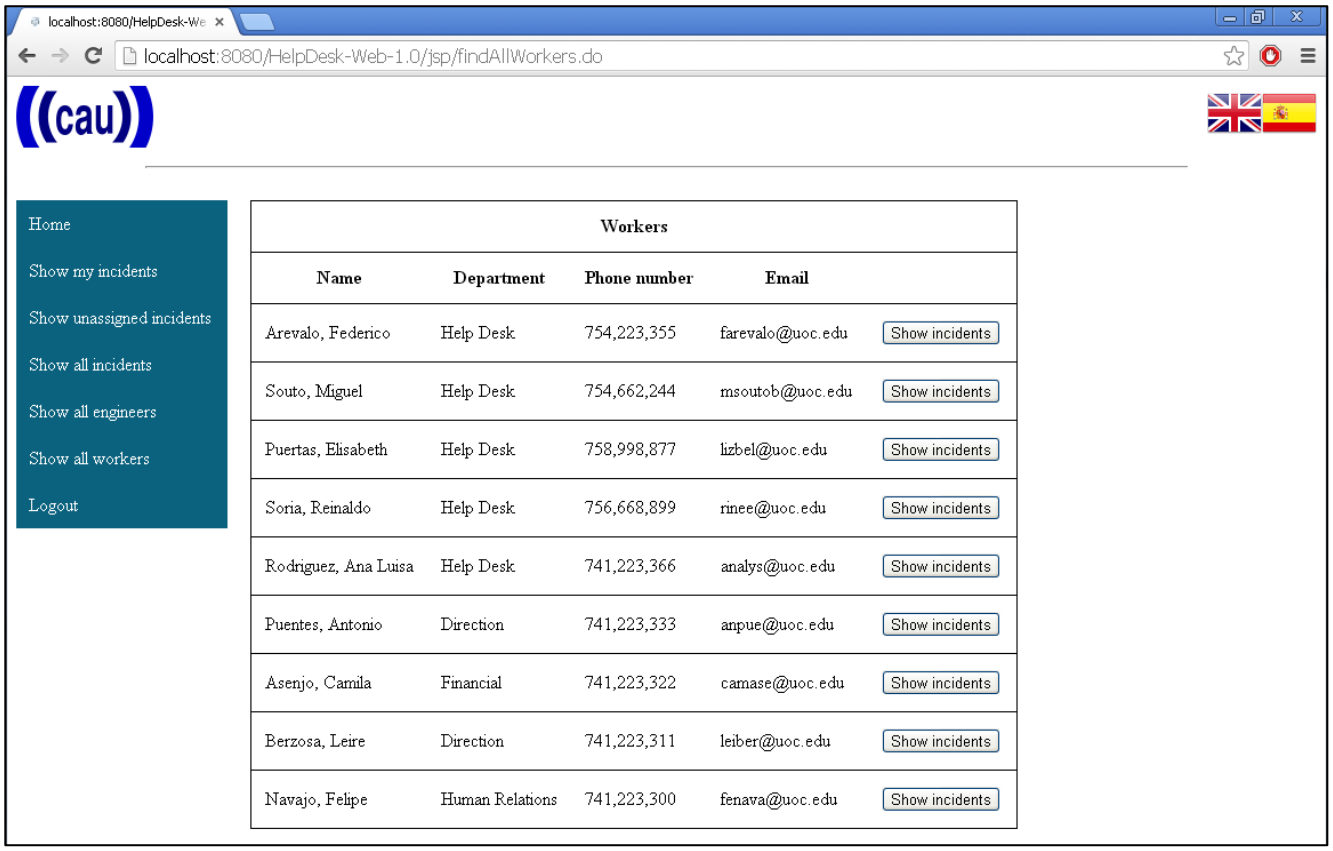


Ilustración 83: HelpDesk - pantalla con todos los trabajadores

A continuación elegimos mostrar las incidencias (“show incidents”) de la trabajadora que comunicó la incidencia, Camila Asenjo.

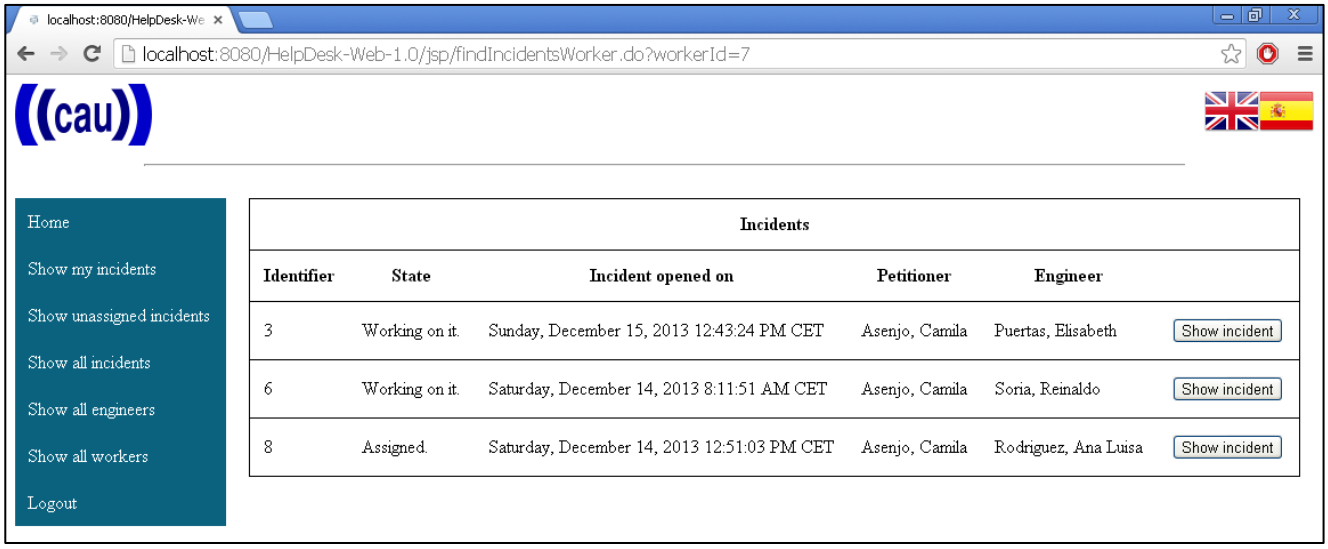


Ilustración 84: HelpDesk - pantalla con las incidencias de una trabajadora

A continuación seleccionamos la incidencia con identificador 3.

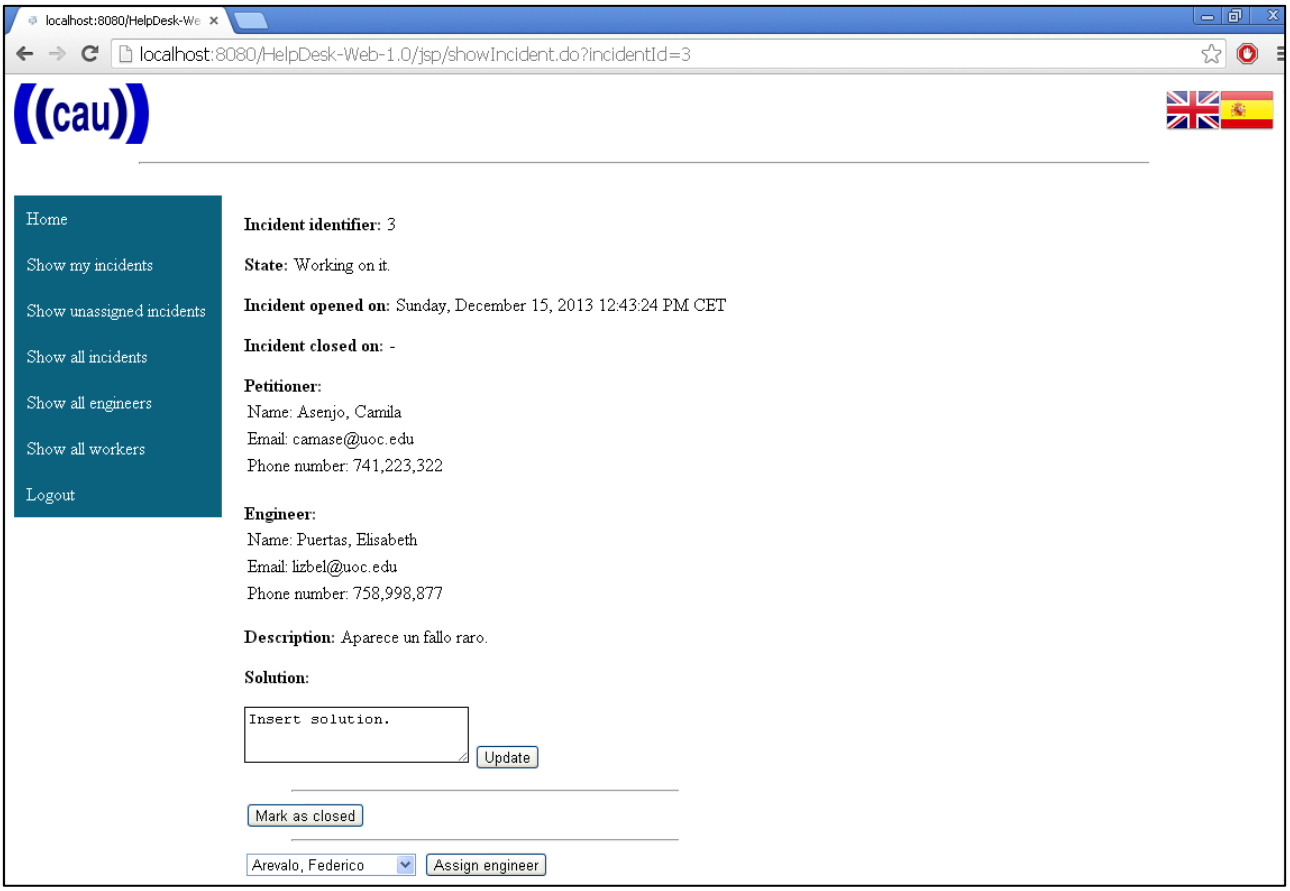


Ilustración 85: HelpDesk - pantalla de incidencia para un administrador

Como se puede comprobar, ahora se muestran también los datos de la ingeniera asignada para resolver la incidencia, así como las operaciones adicionales permitidas al administrador (asignar la incidencia a otro ingeniero). Las fechas y los números también están internacionalizados al estándar inglés, gracias al filtro internationalFilter, y los botones y nombres de los elementos también a través del bundle de la aplicación y el uso de la librería de tags del framework Myst.

El resto de la aplicación es bastante auto explicativo; en caso de duda se invita a consultar el diagrama de estados de la fase de análisis, para saber qué operaciones están permitidas según el estado de la incidencia. También se recomienda consultar la completa documentación del código fuente (o el Javadoc) y los archivos JSP.

6. Conclusiones

6.1. Sobre los frameworks

El estudio ha mostrado que los frameworks se pueden entender como una gran biblioteca, en la cual se recoge el conocimiento de miles de expertos. Este conocimiento se ha reunido y estandarizado en la forma de patrones de diseño, y los frameworks los utilizan para definir su núcleo central. De esta forma, a través de los frameworks, los patrones nos ofrecen un conjunto de soluciones y técnicas contrastadas, ideales para solventar unos problemas que aparecen de forma muy frecuente, como es el caso de crear una aplicación JEE.

Pero un framework no es solo un conjunto de buenas prácticas aplicadas, sino una solución global que abstrae al programador de muchos aspectos complejos y repetitivos, y le ofrece un conjunto de herramientas que le ayuden en la creación de aplicaciones. De esta forma, los diversos frameworks existentes ofrecen unas soluciones básicas comunes sin embargo, cada uno ofrece también unas soluciones particulares, producto de unos puntos de vista y opiniones diferentes sobre cómo solventar la problemática.

¿Hay un objetivo claro para los frameworks? ¿Existe alguna meta definida, y una tecnología definitiva que marque el camino a seguir? En cierta manera, se puede decir que el objetivo final de un framework es abstraer completamente al programador de aplicaciones de todo aquello que no sea la lógica de su propia aplicación. Sin embargo, la arquitectura de las aplicaciones JEE, el modelo de aplicaciones por capas, son conceptos que hacen muy difícil la total abstracción. Hoy por hoy, ninguna tecnología permite que el desarrollo de aplicaciones web se acerque a las más tradicionales y sencillas aplicaciones de escritorio.

Pero aunque no haya ninguna tecnología definitiva, no se puede acusar a los frameworks de ser inmovilistas. Justo al contrario, el mundo de los frameworks se encuentra en constante evolución, y aunque tiene una base de tecnologías fuertemente asentadas, como la inversión de control y la inyección de dependencias, la innovación es constante.

Desde la primera versión de Struts, en la cual el programador solo podía usar objetos complejos, hasta Tapestry, con el cual los objetos del programador son simples y portables, ha habido una tremenda evolución, en la cual los frameworks se han ido volviendo progresivamente mucho más complejos, mientras que el desarrollo de aplicaciones es cada vez más sencillo.

6.2. Trabajo personal

En el terreno personal, debo decir que antes de comenzar este proyecto conocía los patrones de diseño, pero el mundo de los frameworks era prácticamente desconocido para mí. Mis únicas experiencias con aplicaciones empresariales Java habían sido la programación directa de Servlets, con sus comandos y páginas JSP asociadas, y la creación de una aplicación web asíncrona con Google Web Toolkit. Por ello, la realización de este proyecto me ha mostrado un mundo prácticamente nuevo, pero que se ha revelado como fuertemente consolidado.

Desarrollar un framework de presentación no solamente me ha hecho aprender en profundidad las funcionalidades que debe ofrecer, y cómo desarrollarlas, sino que ha repercutido en un mayor conocimiento de las técnicas de programación de Java y las posibilidades que ofrecen. El concepto de la reflexión me ha resultado especialmente novedoso, también complejo, pero fundamental para permitir el uso de objetos simples en el desarrollo de una aplicación. Esta ha sido la herramienta que me ha posibilitado crear el motor de inyección de dependencias.

En un plano más personal, he aprovechado para redescubrir la tecnología EJB. Hasta este momento solo había utilizado la compleja y tediosa versión 2.0, por lo que el cambio a la versión 3.1 ha resultado ser espectacular. Aunque no sea parte directa del estudio que he realizado, la evolución de la API EJB me parece un magnífico ejemplo ilustrativo del camino de simplificación que deben seguir los propios frameworks. Y es que opino que la existencia de los frameworks tiene un propósito único: facilitar la labor del programador de aplicaciones empresariales.

Todo proyecto encuentra dificultades a lo largo de su desarrollo, y en mi caso se pueden resumir en tres. La primera ha sido realizar el propio estudio de los frameworks, pues al principio me encontraba un tanto desorientado. Hay mucha información, pero poco estructurada, por lo que resultó una tarea ardua.

El segundo problema fue aprender las técnicas de programación adecuadas para crear el framework. Varias tecnologías desconocidas para mí resultaron complejas, difíciles de utilizar, o encontraba documentación contradictoria. La reflexión, comprensiblemente, ha resultado un escollo, pero crear una biblioteca de etiquetas, curiosamente, me llevó muchas horas hasta que funcionó correctamente.

La última dificultad -un clásico para mí- es la configuración e interconexión de herramientas de desarrollo. El plugin de Maven para Eclipse se convirtió en la principal fuente de problemas en un momento del desarrollo demasiado avanzado para intentar usar otro conjunto de tecnologías. Básicamente todo lo que podía salir mal, salió mal, aunque finalmente los problemas fueron solventados.

6.3. Mejoras futuras

Con respecto a mi framework, aunque la creación de Myst me ha dejado plenamente satisfecho con mi trabajo, también es cierto que, en caso de haber tenido más tiempo, hay ciertos aspectos en los que me hubiera gustado ahondar.

- He quedado convencido de la necesidad de primar la convención sobre la configuración en este tipo de herramientas, por lo que en una hipotética versión futura intentaría reducir a su mínima expresión, o eliminar, los archivos de configuración. El uso de anotaciones y modificación del bytecode podrían ser funcionalidades interesantes para el framework.

- También me hubiera gustado crear más filtros de uso general que acompañaran al framework. Tareas genéricas como login, logout, debug, subida y descarga de archivos, etc., son suficientemente interesantes y genéricas como para facilitarlas con el propio framework. Afortunadamente, estos filtros pueden ser desarrollados por cualquier tercera parte sin necesidad de modificar el framework. Este es un buen ejemplo del carácter modular y escalable de Myst.

- Probablemente, una futura versión de este framework también se acercara a la orientación a eventos y componentes. A pesar de utilizar una buena capa de abstracción, considero que la orientación a acciones y URLs sigue exigiendo al programador de aplicaciones un conocimiento más que superficial sobre todo lo que rodea a un Servlet, y es un aspecto que hay que tratar de mejorar.

- Por último, hay un aspecto que me parece muy interesante y que no se ha visto en este proyecto, el soporte para peticiones asíncronas, posiblemente basado en el estándar AJAX. Esto facilitaría en gran manera la creación de aplicaciones asíncronas, como aquellas que comunican a usuarios remotos, o las que permiten descargar un archivo grande sin interrumpir la aplicación. Seguramente, esta capacidad implicaría cambios sensibles en el framework, como la necesidad de crear etiquetas para facilitar el uso de Javascript.

- Para dar más consistencia al framework, también sería conveniente la realización de más juegos de pruebas. Utilizando JUnit y un framework de creación de objetos mock, como por ejemplo Mockito, habría que crear un completo juego de pruebas automatizadas que probaran el framework de la forma más completa posible. También sería muy útil la creación de un test de usuarios tanto de la aplicación de pruebas, que podría hacerse más compleja, como de la aplicación de muestra.

7. Tecnologías

Para la programación del framework, la creación y análisis de pruebas, y la creación de la aplicación de muestra, se ha utilizado el siguiente software:

- Java EE 5 para el framework Myst y la aplicación para pruebas.
- Java EE 6 para la aplicación de muestra HelpDesk.
- Eclipse Kepler.
- Maven a través del plugin m2e de Eclipse.
- JAXB Eclipse plugin.
- Tomcat v7.0.
- JBoss AS 7.
- MySQL 5.6.15.
- MySQL connector Java 5.1.27.
- Windows 7, Windows XP.

8. Glosario

AJAX: Asynchronous JavaScript And XML. Técnica de desarrollo web para crear aplicaciones interactivas.

API: Interfaz de Programación de Aplicaciones. Conjunto de métodos proporcionados por una biblioteca a modo de capa de abstracción que pueden ser accedidos por otros programas.

BBDD: siglas para base de datos.

Bean: componentes reutilizables Java que sirven para encapsular otros objetos, a los que permiten acceder mediante getters y setters.

Bundle: paquete de archivos que contiene un conjunto de expresiones en varios idiomas, de forma que puede alternarse de un idioma a otro en una programa.

Bytecode: código intermedio entre el código máquina y el binario, interpretable por una máquina virtual, como la de Java.

EAR: Enterprise ARchive. Formato de empaquetamiento de Java EE para distribuir y desplegar aplicaciones empresariales, formadas por una aplicación web, beans de sesión, clases POJO, etc.

Eclipse: entorno de programación compuesto por una amplia variedad de herramientas que facilita la programación de aplicaciones Java.

EJB: Enterprise JavaBean. API para el desarrollo de aplicaciones empresariales de servidor. Provee comunicación remota, publicación de servicios, control de concurrencia, seguridad, etc.

EL: Expression Language. Lenguaje de programación para introducir expresiones en páginas JSP.

Facelets: sistema de plantillas web basada en componentes, de alta reusabilidad.

Framework: conjunto de programas y bibliotecas que ofrece ayuda y funcionalidad para crear un tipo de software determinado.

Getter: método para obtener un objeto encapsulado en una clase. El nombre del método comienza por "get" y termina por el nombre del objeto.

Hibernate: popular implementación de JPA.

HTML: HyperText Markup Language. Lenguaje de marcado predominante para la elaboración de páginas web. Es usado para describir la estructura y el contenido en forma de texto, así como para complementar el texto con objetos tales como imágenes.

IoC: Inversion of Control. Técnica de programación por la cual las bibliotecas o frameworks que utiliza una aplicación controlan el flujo de la propia aplicación.

JAR: Java ARchive. Formato de empaquetamiento Java para distribuir varias clases, recursos y metadatos relacionados. Las bibliotecas Java se distribuyen en este formato.

Java: lenguaje de programación orientado a objetos que compila el código a un bytecode ejecutable en una máquina virtual.

Javadoc: aplicación para la generación automática de documentación a partir de clases Java que utilizan anotaciones específicas. Por extensión, la documentación generada por esta aplicación muchas veces también es referida por el mismo nombre.

Javascript: lenguaje de programación interpretado orientado a objetos, muy utilizado en el lado cliente de aplicaciones web.

JAXB: Java Architecture for XML Binding. Permite mapear clases Java a archivos XML. Tiene dos funciones: una que serializa los objetos Java a XML; y otra que deserializa XML en objetos Java.

JBoss: servidor de aplicaciones de código abierto. Implementa toda la pila de servicios de JEE.

JEE: Java Enterprise Edition. Plataforma de programación para el desarrollo de aplicaciones Java empresariales.

JNDI: Java Naming Directory Interface. API de servicios de directorio. Permite asignar un nombre a un recurso, de forma que los clientes de ese recurso pueden localizarlo de forma sencilla.

JPA: Java Persistence API. Framework del lenguaje de programación Java que facilita el acceso a una base de datos relacional, mapeando los objetos de la lógica de negocio y manteniendo la persistencia de los datos.

JSP: JavaServer Pages. Tecnología que permite crear páginas web dinámicas basadas en HTML.

JUnit: bibliotecas utilizadas en programación para hacer pruebas unitarias de aplicaciones Java.

Login: identificación y conexión de un usuario a un sistema informático.

Logout: desconexión de un usuario de un sistema informático.

Maven: aplicación para la automatización en la creación de proyectos Java. Permite definir las dependencias en XML, y automáticamente se encarga de importarlas al proyecto.

Mock: objeto artificial que simula el comportamiento de un objeto complejo en un entorno controlado.

MySQL: popular sistema de gestión de bases de datos relaciones.

Plugin: componente software que extiende la funcionalidad de una aplicación.

POJO: Plain Old Java Object. Objeto Java sencillo, en contraposición a los objetos especiales.

Servlet: clase Java que atiende a peticiones HTTP y genera respuestas. Un servlet es instanciado una única vez con la carga del servidor.

Setter: método para asignar un objeto encapsulado en una clase. El nombre del método comienza por “set” y termina por el nombre del objeto.

Tag: etiqueta que delimita un elemento en los lenguajes basados en XML. En JSP los tags realizan cierta funcionalidad, definida en su correspondiente clase Java.

TI: Tecnologías de la Información. Dicho de otra manera, servicio de consulta y resolución de incidencias informáticas de una empresa.

Tomcat: servidor web con soporte para Servlets y JSP.

URI: Uniform Resource Identifier. Cadena de caracteres que identifica un recurso.

URL: Uniform Resource Locator. Cadena de caracteres que identifica un recurso y permite su localización en Internet.

Velocity: motor de plantillas que sirve, entre otra cosas, para producir páginas web dinámicas. Para ello sustituye automáticamente anotaciones en el código HTML por objetos del negocio.

WAR: Web application ARchive. Formato de empaquetamiento de Java EE para distribuir y desplegar aplicaciones web.

XML: eXtensible Markup Language. Metalenguaje de etiquetas que permite definir lenguajes para diferentes necesidades.

XML Schema: descripción de la sintaxis de un determinado subconjunto de XML.

XSLT: Extensible Stylesheet Language Transformations. Lenguaje para transformar documentos XML en otros, o incluso en documentos que no son XML.

9. Bibliografía

- "Core J2EE Pattern" [en línea]. 29/01/2006 [02/11/2013]. Disponible en la web: http://www.corej2eepatterns.com/index.htm
- "Modelo Vista Controlador" [en línea]. 03/10/2013 [20/10/2013]. Disponible en la web: http://es.wikipedia.org/wiki/Modelo_Vista_Controlador
- "Software and Design Patterns" [en línea]. 04/10/2013 [20/10/2013]. Disponible en la web: http://best-practice-software-engineering.ifs.tuwien.ac.at/patterns.html
- "Software framework" [en línea]. 04/11/2013 [07/11/2013]. Disponible en la web: http://en.wikipedia.org/wiki/Software_framework
- "Struts architecture" [en línea]. [20/10/2013]. Disponible en la web: http://dev.anyframejava.org/docs.en/anyframe/plugin/optional/struts/1.0.1/reference/html/ch01.html
- "Struts framework and model-view-controller design pattern" [en línea]. [20/10/2013]. Disponible en la web: http://publib.boulder.ibm.com/infocenter/rtnlhelp/v6r0m0/index.jsp?topic=%2Fcom.ibm.etools.struts.doc%2Ftopics%2Fcstrdoc001.html

- Garnier, Jean-Michel. “Struts 1.1 Controller UML diagrams” [en línea]. 19/12/2003 [22/10/2013]. Disponible en la web: http://rollerjm.free.fr/pro/Struts11.html

- “Struts” [en línea]. [20/10/2013]. Disponible en la web: http://struts.apache.org/

- “Detail information on Struts 2 Architecture” [en línea]. 21/05/2007 [22/10/2013]. Disponible en la web: http://www.roseindia.net/struts/struts2/struts-2-architecture.shtml

- Patel, Viral. “Introduction To Struts 2 Framework” [en línea]. 22/12/2009 [24/10/2013]. Disponible en la web: http://viralpatel.net/blogs/introduction-to-struts-2-framework/

- Newton, Dave. “Apache Struts 2 Web Application Development” [en línea]. Birmingham, Reino Unido: Packt Publishing, 2009 [24/10/2013]. Disponible en la web: http://packtlib.packtpub.com/library/9781847193391/ch15

- “Spring Framework” [en línea]. 28/10/2013 [01/11/2013]. Disponible en la web: http://es.wikipedia.org/wiki/Spring_Framework

- “Spring Framework” [en línea]. 05/10/2013 [01/11/2013]. Disponible en la web: http://en.wikipedia.org/wiki/Spring_Framework

- “Spring Framework Tutorial” [en línea]. [02/11/2013]. Disponible en la web: http://www.tutorialspoint.com/spring/index.htm

- “Convention over configuration” [en línea]. 06/10/2013 [02/11/2013]. Disponible en la web: http://en.wikipedia.org/wiki/Convention_over_configuration

- “Beans, BeanFactory and the ApplicationContext” [en línea]. [02/11/2013]. Disponible en la web: http://docs.spring.io/spring/docs/1.2.x/reference/beans.html

- Walsh, Brian. “AJAX and the Spring Framework with TIBCO General Interface” [en línea]. [02/11/2013]. Disponible en la web: http://brianwalsh.ulitzer.com/node/253549/mobile

- “Web MVC framework” [en línea]. [02/11/2013]. Disponible en la web: http://docs.spring.io/spring/docs/3.0.x/reference/mvc.html

- “Apache Tapestry” [en línea]. 24/09/2013 [03/11/2013]. Disponible en la web: http://en.wikipedia.org/wiki/Apache_Tapestry

- Díaz Sánchez, Rubén y Calderón Sánchez, Alejandro. “Apache Tapestry” [en línea]. 18/05/2012 [03/11/2013]. Disponible en la web: http://osl2.uca.es/wikiIW/index.php/Apache_Tapestry

- “Tapestry 5 Documentation” [en línea]. [03/11/2013]. Disponible en la web: http://tapestry.apache.org/documentation.html

- Jenkov, Jakob. “Java reflection tutorial” [en línea]. [06/12/2013]. Disponible en la web: http://tutorials.jenkov.com/java-reflection/index.html

- “Trail: Internationalization” [en línea]. [06/12/2013]. Disponible en la web: http://docs.oracle.com/javase/tutorial/i18n/index.html

- “Lesson: Regular expressions” [en línea]. [06/12/2013]. Disponible en la web: http://docs.oracle.com/javase/tutorial/essential/regex/

- Miller, Andrig y Husar, Radoslav. “Data Source Configuration in AS 7” [en línea]. 30/09/2013 [21/12/2013] Disponible en la web: https://community.jboss.org/wiki/DataSourceConfigurationInAS7

- Macherla, Praveen. “How to create EJB3 JPA project in Eclipse (JBoss AS 7.1)” [en línea]. 15/02/2012 [07/01/2014]. Disponible en la web: http://theopentutorials.com/examples/java-ee/ejb3/how-to-create-ejb3-jpa-project-in-eclipse-jboss-as-7-1