Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

# Diseño y desarrollo de una plataforma multiportal para dar soporte al proyecto gvSIG

Trabajo Fin de Grado

**Grado en Ingeniería Informática**

**Autor**: Lluís Marqués Castelló

**Tutor**: Mª Carmen Penadés Gramaje

4º Curso de Grado en Ingeniería Informática

03-12-2014

A mi padre,

y

A mi madre

# Resumen

El trabajo de fin de grado "*Diseño y desarrollo de una plataforma multiportal para dar soporte al proyecto gvSIG"* es el resultado de aplicar el proceso establecido por la Ingeniería del Software. En este proyecto introduce el contexto en el que se desarrolla el problema que se expone, la necesidad de un portal web renovado que de soporte a la asociación gvSIG. Tras el análisis de problema se propone como solución Liferay Portal. Este proyecto explica cómo esta plataforma puede solucionar el problema planteado realizando un diseño y desarrollo de la solución siguiendo la metodología RUP y usando las funcionalidades que la plataforma nos ofrece, tales como, estructuras, plantillas, *portlets*…

**Palabras clave:** gvSIG, Liferay, portlet, UML, RUP, CMS, Ingeniería Software, SIG

# Abstract

The project *"Diseño y desarrollo de una plataforma multiportal para dar soporte al proyecto gvSIG"* is the result of applying the software engineer process. This project introduces the context in which the exposed problem is developed, the needed to create a new portal that supports gvSIG association. After analyzing the problem is suggested as solution Liferay Portal. In addition to, this project explains how this platform can resolve the problem presented by designing and developing the solution following RUP and using the features that Liferay platform offer us, such us, structures, templates, portlets…

**Keywords:** gvSIG, Liferay, portlet, UML, RUP, CMS, Sotfware Engineering, SIG

# Índice de contenidos

# Índice de figuras

# Índice de tablas

# PARTE I. INTRODUCCION, CONTEXTO Y SOLUCIÓN

# Capítulo 1. Introducción

Este trabajo de fin de grado tiene como objetivo principal desarrollar una plataforma multiportal para dar soporte a la asociación gvSIG. Para cumplir este objetivo, en este documento se reflejará el proceso seguido y aplicado para abordar la problemática. Este trabajo de fin de grado se sitúa dentro del marco de prácticas en empresa, concretamente en la empresa DISID Technologies la cual es socia de pleno derecho en la asociación gvSIG. DISID comparte el modelo de negocio de software libre de la asociación y participa activamente en el desarrollo y arquitectura software de gvSIG haciendo especial hincapié en la versión 2.0. Además también participa en proyectos como gvSIG Educa/Batoví.

El trabajo incorpora una primera fase de inicio donde se elaborarán un modelo de casos de uso, especificación de requisitos y un diagrama de casos de uso. La siguiente fase que se describe es la de elaboración donde de detallan los casos de uso, diagramas de clases y finalmente diagramas de secuencia. También se detalla una fase de construcción donde se detallan las iteraciones de implementación del sistema y finalmente una fase de transición la cual se detalla el mantenimiento realizado.

El resto del capítulo se compone del siguiente modo: la sección 1.1 se describe la motivación que ha llevado a la realización de este trabajo de fin de grado, en la sección 1.2 se representan los objetivos de este trabajo, y finalmente en la sección 1.3 se describe la estructura del documento, presentando los contenidos de cada capítulo.

## 1.1 Motivación

El proyecto nace en 2004 en el seno de un proyecto de migración de sistemas informáticos de la Consellería de Infraestructuras y Transporte (CIT). Los principales objetivos del proyecto eran cubrir las necesidades de la CIT, pero rápidamente estos objetivos se ven ampliados debido, por un lado, a la naturaleza del software libre y a la facilidad de ser ampliado por una comunidad establecida alrededor del proyecto y por otro, una visión de unas líneas de demarcación y un plan para llevarlas a cabo.

Para entender la interpretación del software libre del proyecto gvSIG hay que situarse en el año 2004 donde gran parte del campo de la geomática está presidido por grandes compañías y su software privativo. Un sector donde se acostumbraba ver proyectos de software libre que finalizaban antes de empezar prácticamente y los que conseguían continuar sin naufragar no conseguían tener un impacto significativo en el campo. Entonces ¿qué hacía diferente este proyecto a los demás? gvSIG da un interpretación que va más allá de la puramente técnica, estableciendo relaciones económicas e ideológicas. El software libre debe ser algo más que compartir archivos fuente o binarios. Compartir el conocimiento no debe ser un fin en si mismo, sino un medio. Un medio para conseguir la misión del proyecto.

“gvSIG tiene la misión de cambiar la concepción de la forma de avanzar en el desarrollo del conocimiento. Consiste en convertir el conocimiento adquirido en conocimiento compartido, de

forma que se puedan sumar cuantos más grupos mejor para conseguir la solución a un problema o el desarrollo de una temática en cuestión" [gmvv14].

La principal motivación de este trabajo es contribuir en el proyecto, proporcionando a la organización un nuevo portal web atractivo que mejore la comunicación entre los usuarios de la comunidad, la visualización de la información y la imagen del proyecto en un conjunto para ayudar conseguir el fin que persiguen y conseguir así que el interés general prevalezca sobre el individual.

## 1.2 Objetivos

El objetivo principal de este trabajo es desarrollar una plataforma multiportal para la asociación gvSIG aplicando métodos de la Ingeniería del Software aprendidos durante el estudio del grado, obteniendo así una solución fruto del proceso aplicado.

Este objetivo general se puede descomponer en subobjetivos relacionados con la solución propuesta:

- Describir el proceso llevado a cabo para la obtención de la solución al problema planteado.
- Analizar, especificar y validar los requisitos proporcionados por la asociación para el nuevo portal.
- Diseñar, en base a esos requisitos, una plataforma multiportal basada en Liferay 6.2 empleando las técnicas de modelado UML.
- Implementar el diseño especificado usando las tecnologías disponibles.
- Ofrecer una solución fruto del proceso aplicado en base a los requerimientos.

Además, otros subobjetivos relacionados con la titulación de grado:

- Ampliar el conocimiento sobre desarrollo y gestión web.
- Involucrarse dentro de un proyecto real y obtener experiencia laboral.
- Demostrar los conocimientos obtenidos a lo largo del estudio del grado.

## 1.3 Estructura del documento

El documento está dividido en tres partes, cada una de ella contiene una serie de capítulos.

La **Parte I** es una introducción al trabajo para establecer las bases. Además de la introducción está compuesta por el capítulo 2 y 3.

En el **capítulo 2** trata sobre el problema que se plantea. Se describe cómo funciona la actual web, que estructura y apartados tiene, como los usuarios interactúan con dicha web y que carencias presenta. También se plantean posibles soluciones alternativas al problema.

En el **capítulo 3** se describe la solución planteada al problema propuesto de forma resumida y el método a seguir para llegar a la solución.

La **Parte II** contiene los capítulos de desarrollo de la solución. Está compuesta por los capítulos 4, 5, 6 y 7.

En el **capítulo 4** se expone la fase de inicio del desarrollo. Se explica la visión del proyecto, el modelo de casos de uso del sistema y la especificación de requisitos que debe cumplir.

En el **capítulo 5** se explica la fase de elaboración del desarrollo. Se define con más detalle cada uno de los casos de uso identificados, la lógica de negocio y finalmente la arquitectura del sistema

En el **capítulo 6** desarrolla la fase de construcción. Esta divida en dos iteraciones, la primera se explica el proceso de desarrollo de la solución del proyecto y la segunda se expone la implementación de un *portlet.*

En el **capítulo 7** expone la fase de transición. Explica la configuración de la arquitectura del proyecto para la puesta a puesta a punto.

La **Parte III** contiene las conclusiones generales del trabajo realizado. Se compone del **capítulo 8** donde se repasan y se resumirán las ideas y metodologías aplicadas y desarrolladas. También se describe brevemente posibles trabajos futuros en base a este.

Finalmente se muestran las referencias bibliográficas utilizadas para la elaboración de este proyecto.

# Capítulo 2. Portal web gvSIG

En este capítulo pretende introducir el contexto en el que se encuentra el problema, para así diseñar una solución que se adapte mejor al problema. Este capítulo está compuesto por una introducción, una descripción del problema y finalmente un planteamiento de soluciones alternativas.

En la sección 2.1 se describe el contexto donde se encuentra el actual portal, en que tecnología se basa, que información ofrece y como se estructura. En la sección 2.2 se describen las carencias y problemas del actual portal y que cambios generales necesita para adaptarse a los nuevos requerimientos. En la sección 2.3 se explican términos claves para nuestra solución y se realiza una comparativa en base a cinco criterios entre las posibles soluciones planteadas.

## 2.1 Introducción

El actual portal web está compuesto por dos dominios: www.gvsig.org y www.gvsig.com. Los dos portales contienen información acerca de los productos que desarrollan, información acerca de la asociación y soporte a la comunidad.

### 2.1.1 Estructura

El portal **gvsig.org** contiene un *home* donde encontramos una barra de navegación que permite al usuario navegar entre las diferentes secciones del portal (ver Figura 2.1). Las secciones son:

- **Inicio**: Es el *home* del portal. Contiene enlaces a las diferentes secciones del portal tales como: los diferentes productos que desarrollan en la asociación, documentación de los productos, información sobre la organización, portales de la comunidad.
- **Organización**: Contiene información sobre la organización como la misión, valores y visión de la organización. También hay información a los servicios que proporcionan, noticias sobre la organización y finalmente información sobre grupos de trabajo.
- **Documentación**: Contiene tres secciones: divulgación, formación, manuales y guías sobre los productos. En la sección divulgación podemos encontrar artículos, ponencias, revistas, libros, blog y otros materiales. En la sección formación nos lleva a otra página web donde tenemos cursos para todo tipo de usuarios, desde los usuarios de los diferentes productos hasta cursos para el desarrollo y extensión de los productos. Y finalmente en la última sección disponemos de manuales y guías para los usuarios, desarrolladores y *testers* de los productos.
- **Descargas**: Contiene los enlaces para descargar los diferentes productos.
- **Noticias**: Contiene todas las noticias que han sido publicadas ordenadas por fecha.

- **Producción**: En el área de producción encontramos un mapa sobre el área de producción, un listado de los proyectos, las diferentes actividades y grupos de trabajos que existen y por último, información sobre la gestión de las traducciones.

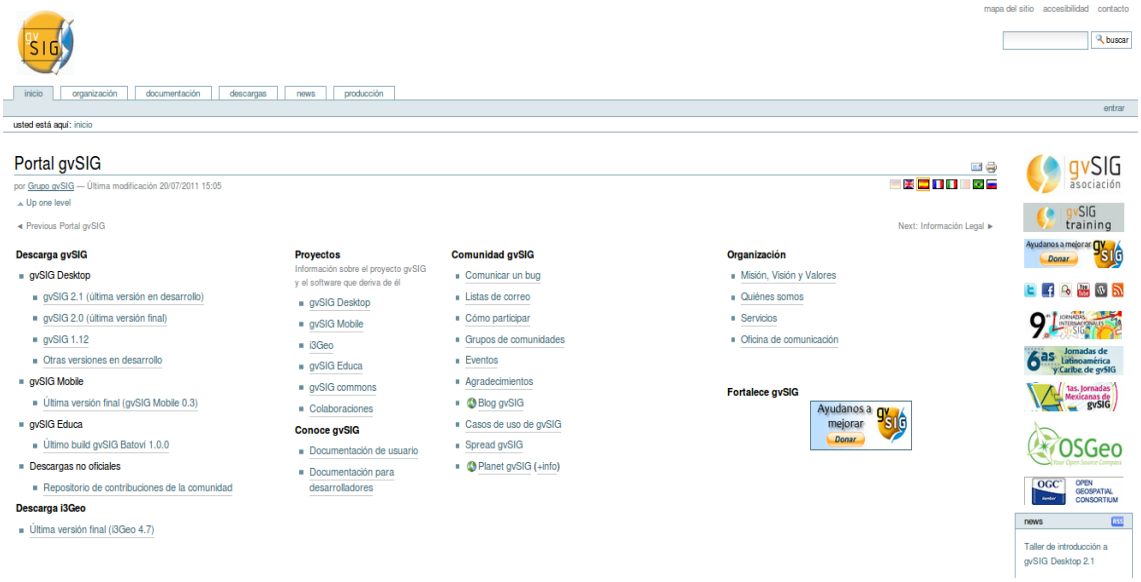



**Figura 2.1** Portal gvSIG.org

Una vez presentada la estructura del portal gvsig.org presentamos la estructura del portal **gvsig.com** (ver Figura 2.2). Se compone de las siguientes secciones:

- **Inicio:** Presenta una descripción breve sobre la asociación y un enlace al otro portal.
- **Productos:** Un listado de enlaces a las páginas de los diferentes productos donde hay una descripción, un lista de funcionalidades y un enlace de descarga.
- **Servicios:** Muestra una breve descripción de los servicios que ofrecen y unos enlaces que llevan al usuario a una descripción más detallada de los servicios mencionados.
- **Sectores:** Contiene información sobre los diferentes sectores que abarca la asociación que van desde la administración local hasta seguridad ciudadana pasando por agricultura, ganadería y pesca.
- **Asociación:** Contiene información sobre el perfil de la asociación, principios, objetivos y estatutos en diferentes idiomas
- **Noticias:** Contiene un lista de las noticias de la organización.
- **Contacto:** Contiene información de como contactar con la organización, así como un listado de todos los correos electrónicos de los principales colaboradores de los productos.


**Figura 2.2** Portal gvSIG.com

### 2.1.2 Tecnología

Los dos portales fueron creados en 2007 usando el CMS[1] llamado **Plone** sobre un servidor de aplicaciones llamado Zope pero no fue hasta Marzo del 2010 cuando pasaron a ser los portales principales de la asociación. De acuerdo con Wikipedia: "*Plone es un sistema de gestión de contenidos que puede utilizarse para construir cualquier tipo de sitio web como portales, sitios webs corporativos, sitios web externos o internos, sitios de publicación de noticias, incluyendo blogs, tiendas en línea (E-commerce), como repositorio de documentos y herramienta colaborativa. Plone es un desarrollo basado en código abierto publicado bajo la GNU General Public License (GPL), basado en Zope y programado en Python. [...] Adicionalmente está diseñado para extender sus funcionalidades por defecto por medio de módulos adicionales llamados Products"* [wplo14].

Los portales usan las tecnologías y lenguajes necesarios para la correcta visualización, navegación y comportamiento como son xhtml, hojas de estilo CSS y JavaScript.

1 **C**ontent **M**anagement **S**ystem, en español, sistema de gestión de contenidos. En el apartado 3.1 se explica con más profundidad.

## 2.2 Descripción del problema

El principal problema es la existencia de dos dominios (gvsig.com y gvsig.org). Por un lado, gran parte de la información esta repetida en los dos portales, y por otro lado hay información que solo se encuentra en uno de los portales. La idea es unificar tanto la información como los portales en un único portal que contenga toda la información. Para ello se pretende substituir por completo el portal gvsig.com y progresivamente el portal gvsig.org, pudiendo enlazar contenidos del nuevo portal con contenidos antiguos en gvsig.org.

Otro problema a destacar es el diseño de la interfaz gráfica de usuario. Se pretende cambiar el aspecto por otro más atractivo a nivel de usuario. Además del aspecto visual desfasado, el portal no se adapta a la posibilidad de acceder desde dispositivos como móviles o tabletas, es decir, no es *responsive*. Se pretende también que el nuevo portal sea *responsive* permitiendo a los usuarios poder acceder cómodamente desde sus móviles y tabletas.

## 2.3 Posibles soluciones

Existen plataformas tecnológicas que ayudan a crear y gestionar portales web proporcionándonos una base sobre la que crear nuestro portal web además de muchas funcionalidades. Estas plataformas reciben el nombre de CMS. Según Wikipedia: "*Un **sistema gestor de contenido** (o **CMS**, del inglés **C**ontent **M**anagement **S**ystem) es una aplicación informática usada para crear, editar, gestionar y publicar contenido digital multimedia en diversos formatos. El gestor de contenidos genera páginas web dinámicas interactuando con el servidor web para generar la página web bajo petición del usuario, con el formato predefinido y el contenido extraído de la base de datos del servidor. Esto permite gestionar, bajo un formato estandarizado, la información del servidor, reduciendo el tamaño de las páginas para descarga y reduciendo el coste de gestión del portal con respecto a un sitio web estático, en el que cada cambio de diseño debe ser realizado en todas las páginas web, de la misma forma que cada vez que se agrega contenido tiene que maquetarse una nueva página HTML y subirla al servidor web.*" [wcms14].

El uso de un CMS en este caso nos facilitará la gestión de todo el contenido web, permitiendo que cualquier usuario sin conocimientos de programación o maquetación pueda subir contenido web, la gestión de los posibles usuarios y sus roles dentro del portal, el establecimiento de flujos de trabajo del contenido, el uso de las funcionalidades que trae consigo el CMS o incluso personalizar, modificar y desarrollar nuevas funcionalidades. En definitiva, este tipo de herramientas nos proporciona un marco muy interesante sobre el que trabajar.

Una vez sabemos que es un CMS y que nos aporta, es importante saber que es un contenedor de *servlets* ya que el CMS que vamos a usar tiene incluido un contenedor de *servlets*. Pero para ello, primero debemos saber que es un contenedor web y que es un *servlet*. De nuevo me remito a Wikipedia:

- "Un **contenedor web** es la implementación que hace cumplimiento del contrato de componentes web de la arquitectura J2EE. Este contrato especifica un entorno de ejecución para componentes web que incluye seguridad, concurrencia, gestión del ciclo de vida, procesamiento de transacciones, despliegue y otros servicios." [wcw14].

- "El ***servlet*** es una clase en el lenguaje de programación Java, utilizada para ampliar las capacidades de un servidor. Aunque los *servlets* pueden responder a cualquier tipo de solicitudes, éstos son utilizados comúnmente para extender las aplicaciones alojadas por servidores web, de tal manera que pueden ser vistos como *applets* de Java que se ejecutan en servidores en vez de navegadores web." [wsvlt14].

  Podemos concluir que un contenedor de *servlets* es una implementación que permite alojar *servlets* y ofrecer unos servicios como despliegue, ciclo de vida, seguridad… y nos permite ampliar las capacidades y funciones del servidor. Un ejemplo muy conocido es Apache Tomcat[2] que es el que usaremos para nuestro trabajo.

  Y por último, debemos conocer en qué consiste **Bootstrap**[3] para poder conocer la importancia dentro de nuestro proyecto. Bootstrap es un framework que proporciona al desarrollador elementos gráficos ya diseñados para la construcción de portales web. Bootstrap básicamente está compuesto por hojas de estilo CSS que implementan la variedad de componentes. Estas son dos de las características principales:

- Debido a que no tiene un soporte relativamente grande de HTML5 y CSS3, gran parte que los elementos gráficos son compatibles con la mayoría de los navegadores. Aunque nada te impide, ni te obliga, a usar elementos basados en HTML5 y CSS3 como son las esquinas redondeadas, sombras y gradientes.

- A partir de la versión 2.0, Bootstrap se hace *responsive*, es decir, se ajusta a la resolución de los dispositivos, permitiendo así una diseño de interfaz visible desde cualquier dispositivo ya sea un móvil, tableta o cualquier otro dispositivo con navegador web.

  Gracias a estas dos características que posee Bootstrap podemos crear un tema atractivo, compatible y *responsive*.

Una vez explicados los términos más relevantes dentro de nuestro proyecto se va a introducir los CMS más importantes. Más adelante se realiza una comparación en base a cinco criterios que nos permite obtener conclusiones y decidir que CMS se ajusta más a nuestro problema.

Los CMS candidatos a priori por su relevancia en el mercado son:

- **WordPress**

  La herramienta de creación y de gestión más utilizada del mundo. Este CMS está enfocado a la creación de blogs y sitios webs de forma fácil y sencilla gracias a su interfaz web. Permite crear páginas, usuarios, perfiles… Además cuenta con una gran variedad de extensiones y temas que aportar mayor funcionalidad y gran diversidad de aspectos a este sistema. [wpcabw13]

- **Liferay Portal**:

  "Liferay Portal es una plataforma web empresarial para construir soluciones que ofrecen resultados inmediatos y valor a largo plazo" [lppo14]. Liferay es un gestor de contenidos de código abierto que se creó en un principio para asociaciones sin ánimo de lucro.

2 http://tomcat.apache.org/

3 http://getbootstrap.com/

Funciona con la mayoría de servidores de aplicaciones, totalmente personalizable con más de sesenta *plugins* para instalar.

- **Joomla**

  Joomla permite desarrollar portales webs dinámicos e interactivos. Desarrollado en PHP y con licencia GPL que permite acceder al código fuente y personalizarlo y añadir nuevas funcionalidades. Grandes empresas tiene basados sus portales bajo esta plataforma como por ejemplo Ebay o Ikea. Es muy usado debido a la facilidad de uso que aporta. Existen miles de extensiones disponibles que se pueden añadir con un par de *clicks*, proporcionando un abanico de funcionalidades muy amplio. Igual de sencillo es añadir y editar y organizar contenido web en el portal mediante la interfaz web que proporciona. [jcwdi13]

- **Drupal**

  Drupal es un CMS de código abierto y escrito en PHP y distribuido bajo los derechos de la lincencia GNU (*General Public License).* Drupal es un proyecto de un estudiante de una universidad alemana, Dries Buytaert. El primer objetico era que Dries Buytaert y sus amigos pudieran compartir noticias y eventos. En 2001 Dries Buytaert transformó el proyecto en código abierto que rápidamente fue aceptado por la comunidad, convirtiéndose en uno de los CMS más potentes y sofisticas de la Web. Drupal permite ejecutarse en cualquier sistema que soporte la PHP y un sistema gesto de base de datos como MySQL. [mid710]

A continuación se comparan los CMS brevemente descritos en base a cinco criterios: si el CMS es *Open Source*, el coste, si permite o lleva integrado Bootstrap y finalmente la disponibilidad de idiomas.

| ***Open Source*** | |
|---|---|
| CMS | |
| Liferay Portal | Si |
| WordPress | Si |
| Joomla | Si |
| Drupal | Si |

**Taba 2.1** Comparación CMS *Open Source*

| **Coste** | |
|---|---|
| CMS | |

| | |
|---|---|
| Liferay Portal | La versión *Community Edition* es totalmente gratuita.<br><br>Existen versiones de pago que proporcionan servidores además de una asistencia técnica. |
| WordPress | Versión principiante gratuita. Versiones de pago con más almacenamiento y dominio. |
| Joomla | Gratuito. |
| Drupal | Gratuito. |

**Taba 2.2** Comparación CMS Coste

| **Integración con Bootstrap** | |
|---|---|
| CMS | |
| Liferay Portal | La última versión de Liferay lleva integrado Bootstrap 2.3.2. |
| WordPress | Existen temas para WordPress basados en Bootstrap 3.X. |
| Joomla | Joomla 3.3 lleva integrado Bootstrap 2.X |
| Drupal | Existe un tema para Drupal 7.x-3.0 llamado Bootstrap que integra la versión 3.2 de Bootstrap con el CMS. |

**Taba 2.3** Comparación CMS Facilidad de aprendizaje y uso

| **SGBD compatibles** | |
|---|---|
| CMS | |
| Liferay Portal | Liferay es compatible con Hibernate, BD2, Derby, MySQL, Postgresql... |
| WordPress | Solo es compatible con MySQL |
| Joomla | Compatible con MySQL y PostgreSQL. |

| | |
|---|---|
| Drupal | Preferible el uso de MySQL aunque soporta también PostgreSQL. |

**Taba 2.4** Comparación CMS SGBD compatibles

| **Idiomas** | |
|---|---|
| CMS | |
| Liferay Portal | Liferay soporta gran cantidad de idiomas. Algunos de ellos son: portugués, italiano, inglés, chino, alemán... |
| WordPress | WordPress por defecto no soporta multilingüe. Existen varios plugins, tanto gratuitos como de pago, que incorporan esta funcionalidad. |
| Joomla | Gestor de idiomas instalado por defecto pero necesitamos instalar aquellos idiomas que necesitemos. |
| Drupal | Activando una serie de plugins preinstalados podemos dar soporte multidioma aunque hay que instalar extensiones para incorporar idiomas. |

**Taba 2.5** Comparación CMS Idiomas

De las tablas comparativas se extraen las siguientes conclusiones:

- Todos los CMS son *Open Source* lo cual es muy importante para este proyecto debido a que cumple con la filosofía que intenta promover la asociación gvSIG. Una de las ventajas del código abierto es la posibilidad de que usuarios con conocimientos sobre desarrollo puedan contribuir con el proyecto ya sea corrigiendo errores o aportando nuevas funcionalidades. Como todos los CMS seleccionados son de código abierto de momento no descartamos ninguno.
- Todos los CMS son gratuitos, aunque en el caso de Liferay Portal y Wordpress existen versiones de pago que aportan mayor funcionalidad o soporte. De momento no descartamos ninguna opción, cualquiera nos sirve.
- Todos los CMS comparados son compatibles o tienen temas basados en BootStrap. Las diferentes versiones de Bootstrap que integran no son relevantes debido a que para nuestro proyecto nos sirve cualquiera de las que integran, así que de momento todas las opciones son viables.

- En cuanto al soporte de bases de datos, uno de los requisitos del cliente es el uso PostgreSQL así que WordPress queda descartado. Quedan tres opciones: Liferay, Joomla y Drupal.
- Por último, el idioma es muy importante en nuestro proyecto debido a que la asociación gvSIG tiene miembros y clientes alrededor de todo el mundo. Por lo tanto se decide descartar Joomla y Drupal y optar por **Liferay Portal**. Aunque Joomla y Drupal tengan soporte para incorporar idiomas, Liferay no necesita la instalación de ninguna extensión adicional. Liferay proporciona más de cuarenta idiomas.

# Capítulo 3. Plataforma multiportal propuesta

En este capítulo se pretende introducir la solución planteada al problema anteriormente explicado. Describir la solución en términos generales nos ayudará a nos desviarnos del objetivo que nos hemos propuesto. Este capítulo está compuesto por una introducción y una descripción de la solución que se plantea en este trabajo de fin de grado.

En la sección 3.1 se describe con algo más de detalle el CMS que se va a usar, que lenguajes necesitaremos, que método usaremos y los pasos a seguir para obtener la solución.

## 3.1 Descripción

La solución plantea un portal basado en Liferay, el cual se complementa con otros sistemas y tecnologías. La figura 3.1 muestra un diagrama global:

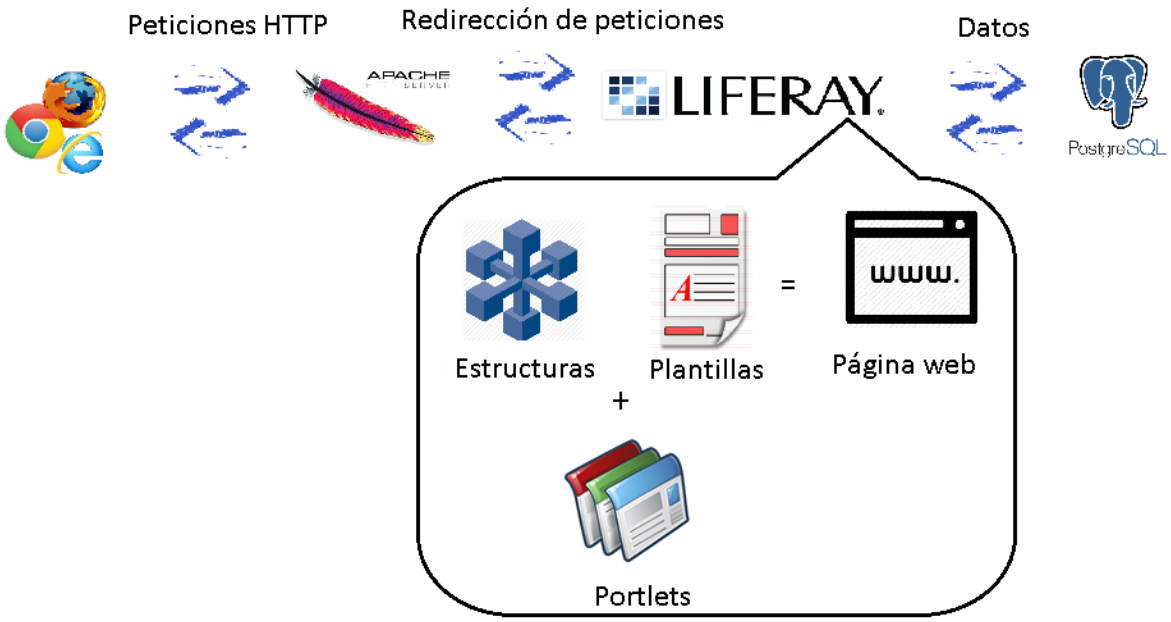



**Figura 3.1** Diagrama global de la solución propuesta

La solución propuesta consta de varios sistemas y tecnologías. Por una parte, el núcleo de la solución es la plataforma Liferay. Liferay aporta un marco de trabajo que ayuda a desarrollar la solución gracias a sus funcionalidades. Dicho servidor ofrece:

- Estructuras: modelos de información definidos en XML que permiten crear contenidos web de forma fácil. Cada estructura está compuesta por campos de diferentes tipos, tales como, fecha, texto, imágenes…
- Plantillas: asociadas a las estructuras permiten mostrar la información que contienen. Las plantillas se definen en el lenguaje **Apache Velocity** que permite la integración con HTML y CSS. Al permitir el uso de CSS y HTML sirve perfectamente para integrar Bootstrap para mostrar la información de las plantillas que cumplan con los requisitos funcionales relacionados con el aspecto y la accesibilidad desde varias plataformas.
- *Portlets*: aplicaciones ejecutadas en el lado del servidor permiten mediante lenguaje Java y JSP generar páginas web en base a una configuración. En nuestro caso nos es muy útil para proporcionar contenido dinámico.

Otro sistema es **Apache.** Apache nos proporcionara la puerta de entrada a nuestro sistema. Ofrece funcionalidades como el modo *proxy* que no es muy útil debido a que proporciona más seguridad a nuestro sistema. Otras funcionalidades que nos pueden servir podrían ser: autentificación mediante https, servidor SSL para la administración del sistema de forma remota, balanceadores de carga…

Otro sistema usado es el gestor de base de datos **PostgreSQL** el cual nos proporciona la persistencia a nuestro sistema y en concreto a nuestro servidor Liferay. Uno de los requisitos del cliente es el uso de este sistema gestor de base de datos. El motivo de esta elección es debido a que a que comparte uno de los objetivos de la asociación. PostgreSQL es de código abierto y totalmente gratuito. Fue creado en la universidad de Californa, Berkeley por un grupo de estudiantes y profesores bajo la dirección del profesor Michael Stonebraker [piac01]. Actualmente, la última versión estable es la 9.3 que sigue estando desarrollada por una comunidad comprometida.

La metodología a seguir es RUP. RUP es un proceso moderno y genérico, está organizado en cinco fases:

- Fase de inicio: el objetivo de la fase de inicio es establecer casos de uso del sistema. Se debe identificar a los actores del sistema y definir sus interacciones.
- Fase de elaboración: los objetivos de la fase de elaboración son desarrollar y entender el dominio del problema, elaborar un diseño de la arquitectura del sistema, desarrollar un plan de desarrollo, detallar las interacciones de los actores e identificar los riegos del proyecto.
- Fase de construcción: la fase de construcción engloba el diseño del sistema, programación y testeo. Al finalizar esta fase se debe obtener un sistema operable.
- Fase de transición: esta fase está relacionada con la migración del sistema desarrollado del entorno de pruebas/desarrollo a un entorno de producción real listo para usar por el cliente.

Además de seguir estas fases también se adoptarán las siguientes buenas prácticas en la medida de lo posible:

- Desarrollo iterativo: establecer iteraciones en la fase de construcción siguiendo el criterio de prioridad establecido por el cliente. Al final de cada fase el cliente validará las nuevas funcionalidades añadidas proporcionándonos el *feedback.*
- Modelado de software gráfico: hacer uso de diagramas UML para el modelado del sistema: diagramas de clases, secuencia, casos de uso…

- Control de requisitos: establecer un control de requisitos frente a los posibles cambios. Analizar el impacto sobre el sistema diseñado al aceptar nuevos requisitos [se911].



# PARTE II. DISEÑO Y DESARROLLO DE LA SOLUCIÓN

# Capítulo 4. Fase de inicio

En este capítulo se pretende introducir y describir la fase de inicio. De la fase de inicio se obtiene los siguientes artefactos: documento de visión, diagrama de casos de uso, especificación de requisitos y diagrama de requisitos. En este proyecto se van a realizar y explicar el documento de visión, el diagrama de casos de uso y la especificación de requisitos.

En la sección 4.1 se hace una introducción al análisis del sistema, explicando en que consiste el proceso en términos generales y que dificultades presenta. En la sección 4.2 se describe con más detalle la fase de identificación del alcance del sistema, los objetivos y el modelo de dominio. En la sección 4.3 se muestra el modelo de casos de uso y finalmente en la sección 4.4 se especifican los requisitos.

## 4.1 Introducción

En 1950 el principal desafío era el desarrollo de hardware para reducir costes en procesamiento y almacenamiento de datos. A medida que la electrónica conseguía avances obteniendo una mayor potencia de cálculo y un abaratamiento en el almacenamiento de datos, el principal desafío se iba desplazando hacia el software. Actualmente el objetivo es obtener software de calidad y abaratar los costes.

El análisis del alcance, objetivos del sistema y la especificación de requisitos es el primer paso para la obtención de software. Es el paso más importante de todo el proceso ya que un mal análisis puede originar problemas en las fases futuras del proceso. Un estudio realizado por **Standish Group** hecho a ocho mil proyectos en trescientas cincuenta empresas de estados unidos muestra que la principal fuente de error proviene de la fase de análisis y especificación de requisitos con un 37%, concretamente por errores producidos por información errónea proveniente del usuario, requisitos incompletos y cambios producidos en los requisitos. Es importante identificar posibles errores en esta fase del proyecto debido a que el coste de reparación en esta fase es mucho más pequeño que la reparación en fases finales como por ejemplo la de mantenimiento. En la siguiente tabla se muestra el coste de reparación de un error en el momento que se detecta.

| ETAPA | COSTE DE REPARACIÓN (horas) |
|---|---|
| Análisis requerimientos | 1-2 |
| Diseño | 5 |
| Codificación | 10 |

| | |
|---|---|
| Pruebas de módulo | 20 |
| Pruebas de sistema | 50 |
| Mantenimiento | 100 - 200 |

**Tabla 4.1** Costes de reparación de un error en el momento en el que se detecta

## 4.2 Visión del proyecto

En este apartado se va a detallar la visión de la plataforma web, es decir los objetivos de la plataforma, el alcance del proyecto y el modelo de dominio.

**Objetivos**

Los objetivos del proyecto son:

- Crear una plataforma web fruto de la unión de los dos portales actuales, consiguiendo así un mejor acceso a la información actual.
- Mejorar el aspecto visual más acorde al diseño web actual, haciéndolo más atractivo de cara a los usuarios.
- Migrar la tecnología sobre la cual se basa los portales actuales para obtener una mayor funcionalidad de cara a los usuarios y una mayor robustez.
- Mejorar la adaptabilidad del portal para mejorar la accesibilidad mediante otros dispositivos como móviles o tabletas.

**Alcance**

El alcance del proyecto incluye las siguientes actividades:

- Diseño y creación de una plataforma web basada en Liferay 6.2 que cumpla con los requerimientos especificados y que permita a los administradores de la plataforma gestionar los contenidos.
- Formar a los administradores en el uso de la plataforma web tanto de forma presencial, resolviendo posibles dudas o dando soporte, como elaborando un manual de administrador.
- Modelar entidades capaces de dar soporte a los contenidos que el cliente quiera mostrar.
- Modelar plantillas de visualización que cumplan con la forma especificada por el cliente.
- Ampliar la funcionalidad del portal para cumplir los requisitos especificados.
- Corregir posibles errores que puedan surgir durante el uso de la plataforma.

La figura 4.1 muestra el modelo de dominio:

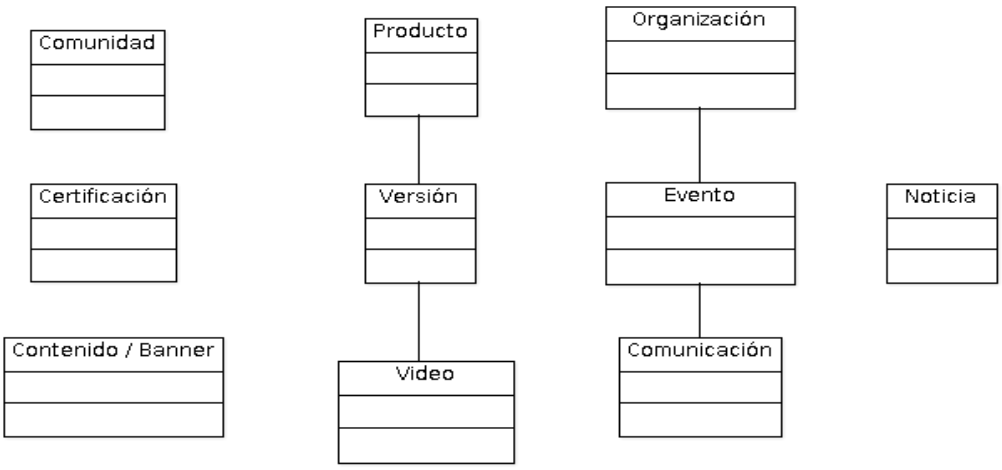



**Figura 4.1** Modelo de dominio

El sistema está compuesto por las entidades: contenido/banner, producto, versión, vídeo, evento, organización, comunicación, noticia, comunidad y certificación.

La entidad **producto** representa cada uno de los productos de la asociación. Está relacionada con la entidad **versión** que representa cada una de las versiones desarrolladas. Una versión está relacionada con la entidad **vídeo** que representa cualquier vídeo sobre una versión de un producto. La entidad **evento** representa un evento organizado por la asociación. Un evento tiene **comunicaciones** que representan cada una de las ponencias del evento. Una comunicación puede estar relacionada o no con un evento debido a que existen comunicaciones como artículos o libros que no están relacionados con ningún evento. Lo mismo ocurre con las **organizaciones**, que pueden estar relacionadas con algún evento ejerciendo el rol de patrocinador o pueden ser colaboradores o miembros de la asociación. La entidad **certificación** representa cada unas de las certificaciones que otorga la asociación en base a unos requisitos. La entidad **contenido/banner** se sitúa en el primer nivel de paginación del portal y representa información a modo de introducción en el portal. La entidad **noticia** representa una información relevante sobre cualquier tema relacionado con la asociación.

## 4.3 Modelo de casos de uso

La figura 4.2 muestra los casos de uso y los actores identificados: Administrador y Usuario. Este proyecto se centra en la funcionalidad del administrador. El administrador debe poder realizar todo tipo de acciones relacionadas con la gestión de contenidos, gestión de páginas, gestión de traducciones, creación de listados de contenidos, creación de formularios web y creación de navegaciones en las páginas. El actor usuario representado en la figura 4.2 únicamente se le ha indicado un caso de uso (“iniciar sesión”), asumiendo que el resto de interacciones con la plataforma son las típicas búsquedas, visualización de información y navegación entre páginas. En

este caso, toda esta funcionalidad queda fuera del alcance de este proyecto.


**Figura 4.2** Diagrama de casos de uso

## 4.4 Especificación de requisitos

Las principales fuentes de requisitos identificadas son:

- ***Stakeholders* relevantes**

  Los *stakeholders* relevantes identificados son: el gerente de la asociación gvSIG y el gerente de comunicación.

- **Sistemas actuales**

  Sistemas actuales identificados los cuales pueden ser relevantes para obtener requisitos son los portales actuales www.gvsig.com y www.gvsig.org.

- **Documentos y formularios de la organización**

  Los formularios interesantes identificados son los formularios de inscripción a jornadas y los formularios de inscripción a talleres.

La siguiente tabla describe los roles que cumplen los *stakeholders* dentro de este proyecto, proporciona una vista de alto nivel sobre los *stakeholders* y el ámbito sobre cada uno de ellos dentro del sistema.

| *Stakeholder* | Rol | Usuario directo | Intereses |
|---|---|---|---|
| Gerente asociación gvSIG | Supervisa la comunicación de la asociación, supervisa el desarrollo y testeo de software, gestiona las jornadas y los talleres, etc. | Si | Mejora de la estructura general del portal, mejora del sistema del portal web, mejora de la imagen de la asociación hacia sus usuarios, mejorar la accesibilidad del portal y proporcionar una mejor calidad de contenido e información a los usuarios. |
| Gerente de comunicación | Encargado de la comunicación de la asociación gvSIG. Administra contenidos de la página web, se encarga de las traducciones, publica noticias, etc. | Si | Mejora en la gestión de contenidos (creación, edición y borrado) y mejora en la gestión de los idiomas de los contenidos. |

**Tabla 4.4** Descripción de las necesidades de los *stakeholders*

Tras haber identificado los *stakeholders*, su rol dentro de la plataforma y los intereses, se listan los requisitos funcionales y no funcionales obtenidos mediante reuniones. Los requisitos funcionales son:

- La plataforma debe ofrecer la funcionalidad de gestionar (añadir, modificar y eliminar) todos tipo de contenidos *webs*.
- La plataforma se debe poder gestionar las páginas del portal.
- En principio deben estar activos los idiomas: español, inglés, portugués, italiano, ruso, portugués de Brasil y ruso. Posibilidad en un futuro la ampliación de idiomas disponibles.
- La plataforma se debe poder crear listados de contenidos que se generen de forma dinámica según unos criterios.

- La plataforma se debe poder añadir navegaciones a las páginas.
- La plataforma debe ofrecer la posibilidad de gestionar formularios web.
- La plataforma debe ofrecer un sistema de almacenamiento de documentos que sean accesibles mediante enlaces de descarga para los usuarios.
- Se debe poder relacionar contenidos *webs*. Por ejemplo una versión se debe poder relacionar por algún método con el producto.
- La plataforma debe de mostrar un listado de enlaces a *builds* que se encuentran alojadas en un servidor externo y que se actualice automáticamente.
- Se debe poder crear conjuntos de entidades con la misma estructura de información, es decir, crear comunidades, noticias, productos, versión... Todas las entidades del mismo tipo contienen los mismos campos de información, es decir, todas las comunidades deben tener un título, una descripción y una lista de colaboradores.

Y los requisitos no funcionales identificados son:

- El diseño del portal debe ser adaptativo para mejorar la accesibilidad desde dispositivos portátiles como móviles y *tablets*.
- Debe poder estar accesible y visualizarse de forma correcta desde cualquier navegador.
- La plataforma debe tener un sistema de respaldo de archivos que se actualice una vez al día.

# Capitulo 5. Fase de elaboración

En este capítulo pretende introducir la solución técnica del proyecto. Los objetivos de esta fase es la elaboración del documento Arquitectura compuesto por la vista de casos de uso, la vista lógica, la vista de implementación y finalmente la vista física.

En la sección 5.1 se va detallar los casos de uso usando una plantilla. Seguidamente en la sección 5.2 se diseñará y explicara el diagrama de clases. En la sección 5.3 se mostrará y explicara los diagramas de secuencia y finalmente en la sección 5.4 se detallará la arquitectura en la cual se basa la plataforma.

## 5.1 Vista de casos de uso

Anteriormente se ha mostrado el diagrama de casos de uso, a continuación se van a explicar los casos de uso siguiendo una plantilla. La plantilla de casos de uso utilizada está basada es la especificada por la MADEJA (Marco de Desarrollo de la Junta de Andalucía)[4].

1. Añadir página

<table>
<tr><td></td><td>Añadir página</td></tr>
<tr><td>Dependencias</td><td>• Iniciar sesión</td></tr>
<tr><td>Precondición</td><td>• La plataforma debe estar configurada y operativa al menos en fase de desarrollo.<br>• El usuario debe haber iniciado sesión<br>• El usuario debe tener permisos para añadir páginas al portal</td></tr>
<tr><td>Descripción</td><td>El sistema deberá comportarse como se describe en la siguiente secuencia:</td></tr>
<tr><td>Secuencia normal</td><td><table><tr><td>Paso</td><td>Acción</td></tr><tr><td>1</td><td>El administrador debe acceder al panel de administrador del portal</td></tr></table></td></tr>
</table>

4 MADEJA http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/409

| | | |
|---|---|---|
| | 2 | Acceder a la gestión de páginas |
| | 3 | Hacer click en Añadir página |
| | 4 | Introducir el nombre de página |
| | 5 | Si no se introduce el nombre de la página, |
| | | 5.1 El sistema vuelve a pedir el nombre de página |
| | 6 | Si se introduce el nombre de página al menos en el idioma por defecto, |
| | | 6.1 El caso de uso termina con éxito. |
| Postcondición | Después de añadir la página se debe mostrar en el árbol de navegación lateral. | |
| Comentarios | Ninguno. | |

2. Editar página

| | | |
|---|---|---|
| | Editar página | |
| Dependencias | • Iniciar sesión<br>• Añadir página | |
| Precondición | • La plataforma debe estar configurada y operativa al menos en fase de desarrollo.<br>• El usuario debe haber iniciado sesión<br>• El usuario debe tener permisos para editar páginas al portal | |
| Descripción | El sistema deberá comportarse como se describe en la siguiente secuencia: | |
| Secuencia normal | Paso | Acción |
| | 1 | El administrador debe acceder al panel de administración. |

| | | |
|---|---|---|
| | 2 | Acceder a la gestión de páginas. |
| | 3 | Hacer click en la página que se desea editar. |
| | 4 | Editar la página. |
| | 5 | Si se hace click en Guardar, |
| | | 5.1 Validar cambios. |
| | | 5.2 Si los cambios son correctos guardar. En caso contrario no guardar y redireccionar a de nuevo a las propiedades de página. |
| | 6 | Si se hace click en Cancelar, |
| | | 6.1 Los cambios no se guardan y se finaliza el caso de uso. |
| Postcondición | Al guardar los cambios, se debe redireccionar de nuevo al panel de edición. Si se cancelan se redirecciona a las propiedades de la página principal. | |
| Comentario | Ninguno. | |

3. Eliminar página

| | |
|---|---|
| | Eliminar página |
| Dependencias | • Iniciar sesión<br>• Añadir página |
| Precondición | • La plataforma debe estar configurada y operativa al menos en fase de desarrollo.<br>• El usuario debe haber iniciado sesión<br>• El usuario debe tener permisos para eliminar páginas al portal |
| Descripción | El sistema deberá comportarse como se describe en la siguiente secuencia: |

<table>
<tr><td>Secuencia normal</td><td>
<table>
<tr><td>Paso</td><td>Acción</td></tr>
<tr><td>1</td><td>El administrador debe acceder al panel de administrador del portal</td></tr>
<tr><td>2</td><td>Acceder a la gestión de páginas</td></tr>
<tr><td>3</td><td>Seleccionar la página a eliminar</td></tr>
<tr><td>4</td><td>Introducir el nombre de página</td></tr>
<tr><td>5</td><td>Se debe mostrar un dialogo para confirmar la eliminación</td></tr>
<tr><td rowspan="2">5</td><td>Si se hace click en aceptar,</td></tr>
<tr><td>5.1 El sistema elimina la página y finalizar el caso de uso.</td></tr>
<tr><td rowspan="2">6</td><td>Si se hace click en Cancelar,</td></tr>
<tr><td>6.1 El sistema no hace nada y redirecciona a la página seleccionada anteriormente.</td></tr>
</table>
</td></tr>
<tr><td>Postcondición</td><td>Después de eliminar la página, debe de desaparecer del árbol de páginas lateral.</td></tr>
<tr><td>Comentarios</td><td>Ninguno.</td></tr>
</table>

4. Añadir contenido web

| | Añadir contenido web |
|---|---|
| Dependencias | • Iniciar sesión<br>• Crear contenido web<br>• Añadir página<br>• Seleccionar plantilla de visualización |
| Precondición | • La plataforma debe estar configurada y operativa al menos en fase de desarrollo. |

<table>
<tr><td></td><td>• El usuario debe haber iniciado sesión<br>• El usuario debe tener permisos para añadir portlets al portal<br>• Que hayan creado contenidos webs</td></tr>
<tr><td>Descripción</td><td>El sistema deberá comportarse como se describe en la siguiente secuencia:</td></tr>
<tr><td>Secuencia normal</td><td><table>
<tr><td>Paso</td><td>Acción</td></tr>
<tr><td>1</td><td>Abrir el panel de aplicaciones</td></tr>
<tr><td>2</td><td>Añadir el portlet visor de contenido a la página</td></tr>
<tr><td>3</td><td>Seleccionar contenido web para mostrar</td></tr>
<tr><td>4</td><td>Hacer click en Guardar</td></tr>
<tr><td>5</td><td>Seleccionar plantilla de visualización</td></tr>
<tr><td>6</td><td>Hacer click en Guardar</td></tr>
<tr><td>7</td><td>Cerrar la ventana</td></tr>
</table></td></tr>
<tr><td>Postcondición</td><td>Después de añadir el contenido web se debe de mostrar en la página.</td></tr>
<tr><td>Comentarios</td><td>Ninguno.</td></tr>
</table>

5. Modificar contenido web

| | Modificar contenido web |
|---|---|
| Dependencias | • Iniciar sesión<br>• Crear contenido web |
| Precondición | • La plataforma debe estar configurada y operativa al menos en fase de desarrollo.<br>• El usuario debe haber iniciado sesión.<br>• El usuario debe tener permisos para modificar |

<table>
<tr><td></td><td>contenidos webs.<br><br>• Deben haber creados contenidos web</td></tr>
<tr><td>Descripción</td><td>El sistema deberá comportarse como se describe en la siguiente secuencia:</td></tr>
<tr><td>Secuencia normal</td><td>
<table>
<tr><td>Paso</td><td>Acción</td></tr>
<tr><td>1</td><td>Acceder al panel de administración</td></tr>
<tr><td>2</td><td>Hacer click en contenido web del panel izquierdo</td></tr>
<tr><td>3</td><td>Desplegar el menú del contenido web que se quiera modificar</td></tr>
<tr><td>4</td><td>Hacer click en Editar</td></tr>
<tr><td>5</td><td>Realizar la modificaciones</td></tr>
<tr><td rowspan="2">6</td><td>Si se hace click en Publicar,</td></tr>
<tr><td>6.1 Validar los cambios, si son correctos guarda y se finaliza el caso de uso. En caso contrario, se redirecciona a la edición del contenido web original indicando las modificaciones no válidas.</td></tr>
<tr><td>7</td><td>Si se hace click en Guardar como borrador,</td></tr>
<tr><td></td><td>7.1 Validar los cambios, si son correctos guarda como borrador y se finaliza el caso de uso. En caso contrario, se redirecciona a la edición del contenido web original.</td></tr>
<tr><td>8</td><td>Si se hace click en Cancelar,</td></tr>
<tr><td></td><td>8.1 Se cancelan los cambios y se redirecciona al usuario a la administración de contenido web.</td></tr>
</table>
</td></tr>
<tr><td>Postcondición</td><td>Si se guarda como borrador y el contenido web se visualiza en alguna página, los cambios no serán visibles hasta que no se publiquen.</td></tr>
<tr><td>Comentarios</td><td>Ninguno.</td></tr>
</table>

6. Eliminar contenido web

<table>
<tr><td></td><td>Eliminar contenido web</td></tr>
<tr><td>Dependencias</td><td>• Iniciar sesión<br>• Crear contenido web</td></tr>
<tr><td>Precondición</td><td>• La plataforma debe estar configurada y operativa al menos en fase de desarrollo.<br>• El usuario debe haber iniciado sesión.<br>• El usuario debe tener permisos para eliminar contenidos webs.<br>• Deben haber creados contenidos web.</td></tr>
<tr><td>Descripción</td><td>El sistema deberá comportarse como se describe en la siguiente secuencia:</td></tr>
<tr><td>Secuencia normal</td><td><table><tr><td>Paso</td><td>Acción</td></tr><tr><td>1</td><td>Acceder al panel de administración</td></tr><tr><td>2</td><td>Hacer click en contenido web del panel izquierdo</td></tr><tr><td>3</td><td>Desplegar el menú del contenido web que se quiera eliminar</td></tr><tr><td>4</td><td>Hacer click en Enviar a la papelera</td></tr></table></td></tr>
<tr><td>Postcondición</td><td>El contenido web se envía a la papelera, si se desea eliminar definitivamente se debe vaciar la papelera.</td></tr>
<tr><td>Comentarios</td><td>Para vaciar la papelera, se debe acceder a la papelera desde el menú situado a la izquierda.</td></tr>
</table>

7. Asociar contenido web

<table>
<tr><td></td><td>Asociar contenido web</td></tr>
<tr><td>Dependencias</td><td>• Iniciar sesión</td></tr>
</table>

<table>
<tr><td></td><td>• Crear contenido web</td></tr>
<tr><td>Precondición</td><td>• La plataforma debe estar configurada y operativa al menos en fase de desarrollo.<br>• El usuario debe haber iniciado sesión.<br>• El usuario debe tener permisos para editar contenidos webs.<br>• El usuario debe tener permisos para crear vocabularios y crear categorías.<br>• Deben haber creados contenidos web.<br>• Deben haber creados al menos un vocabulario y una categoría.</td></tr>
<tr><td>Descripción</td><td>El sistema deberá comportarse como se describe en la siguiente secuencia:</td></tr>
<tr><td>Secuencia normal</td><td>
<table>
<tr><td>Paso</td><td>Acción</td></tr>
<tr><td>1</td><td>Acceder al panel de administración</td></tr>
<tr><td>2</td><td>Hacer click en contenido web del panel izquierdo</td></tr>
<tr><td>3</td><td>Desplegar el menú del contenido web que se quiera asociar.</td></tr>
<tr><td>4</td><td>Hacer click en Editar.</td></tr>
<tr><td>5</td><td>Hacer click en Clasificación</td></tr>
<tr><td>6</td><td>Hacer click en Seleccionar</td></tr>
<tr><td>7</td><td>Seleccionar las categorías.</td></tr>
<tr><td>8</td><td>Si se hace click en Publicar,</td></tr>
<tr><td></td><td>8.1 Validar los cambios, si son correctos guarda y se finaliza el caso de uso. En caso contrario, se redirecciona a la edición del contenido web original indicando las modificaciones no válidas.</td></tr>
<tr><td>9</td><td>Si se hace click en Guardar como borrador,</td></tr>
</table>
</td></tr>
</table>

| | | |
|---|---|---|
| | | 9.1 Validar los cambios, si son correctos guarda como borrador y se finaliza el caso de uso. En caso contrario, se redirecciona a la edición del contenido web original. |
| | 10 | Si se hace click en Cancelar, |
| | | 10.1 Se cancelan los cambios y se redirecciona al usuario a la administración de contenido web. |
| Postcondición | El contenido web queda asociado a una categoría. | |
| Comentarios | Ninguno | |

8. Añadir traducción a un contenido

| | | |
|---|---|---|
| | Añadir traducción a un contenido | |
| Dependencias | • Iniciar sesión<br>• Crear contenido web<br>• Modificar contenido web | |
| Precondición | • La plataforma debe estar configurada y operativa al menos en fase de desarrollo.<br>• El usuario debe haber iniciado sesión.<br>• El usuario debe tener permisos para editar contenidos *webs.*<br>• Deben haber creados contenidos *web.*<br>• Deben haber disponible al menos un idioma más a parte del predefinido en el portal. | |
| Descripción | El sistema deberá comportarse como se describe en la siguiente secuencia: | |
| Secuencia normal | Paso | Acción |
| | 1 | Acceder al panel de administración |

| | | |
|---|---|---|
| | 2 | Hacer click en contenido web del panel izquierdo |
| | 3 | Desplegar el menú del contenido web que se quiera asociar. |
| | 4 | Hacer click en Editar. |
| | 5 | Hacer click en Añadir traducción. |
| | 6 | Seleccionar Idioma |
| | 7 | Rellenar los campos del contenido web con la traducción en la ventana emergente. |
| | 8 | Si el usuario hace click en Guardar, |
| | | 8.1 Se validan los cambios y si son correctos se guarda la traducción, en caso contrario se redirecciona de nuevo a la ventana emergente inicial. Finaliza el caso de uso. |
| | 9 | Si el usuario hace click en Cancelar, |
| | | 9.1 Se cancelan los cambios y se cierra la ventana emergente. Finaliza el caso de uso. |
| Postcondición | La traducción se guarda junto con el contenido web. | |
| Comentarios | La validación de campos consiste en comprobar que los campos requeridos no están vacíos. | |

9. Seleccionar plantilla de visualización por defecto de un contenido web

| | |
|---|---|
| | Seleccionar plantilla de visualización por defecto de un contenido web |
| Dependencias | • Iniciar sesión<br>• Crear contenido web<br>• Modificar contenido web |
| Precondición | • La plataforma debe estar configurada y operativa al |

<table>
<tr><td></td><td>menos en fase de desarrollo.<br>• El usuario debe haber iniciado sesión.<br>• El usuario debe tener permisos para editar contenidos webs.<br>• Deben haber creados contenidos web.<br>• Deben haber creadas plantillas de visualización para el tipo de contenido web.</td></tr>
<tr><td>Descripción</td><td>El sistema deberá comportarse como se describe en la siguiente secuencia:</td></tr>
<tr><td>Secuencia normal</td><td>
<table>
<tr><td>Pasos</td><td>Acción</td></tr>
<tr><td>1</td><td>Acceder al panel de administración</td></tr>
<tr><td>2</td><td>Hacer click en contenido web del panel izquierdo</td></tr>
<tr><td>3</td><td>Desplegar el menú del contenido web que se quiera asociar.</td></tr>
<tr><td>4</td><td>Hacer click en Editar.</td></tr>
<tr><td>5</td><td>Hacer click en Seleccionar justo al lado de donde indica Plantilla.</td></tr>
<tr><td>6</td><td>Seleccionar la plantilla de visualización</td></tr>
<tr><td rowspan="2">7</td><td>Si se hace click en Publicar,</td></tr>
<tr><td>7.1 Se finaliza el caso de uso.</td></tr>
<tr><td>8</td><td>Si se hace click en Guardar como borrador,</td></tr>
<tr><td></td><td>8.1 Guarda como borrador y se finaliza el caso de uso.</td></tr>
<tr><td>9</td><td>Si se hace click en Cancelar,</td></tr>
<tr><td></td><td>9.1 Se cancelan los cambios y se redirecciona al usuario a la administración de contenido web.</td></tr>
</table>
</td></tr>
<tr><td>Postcondición</td><td>Cambiar la plantilla de visualización por defecto solo debe afectar a la asignación automática de plantilla de visualización a</td></tr>
</table>

| | |
|---|---|
| | la hora de seleccionar contenidos webs desde un visor de contenido web. |
| Comentarios | Ninguno. |

10. Añadir navegación

| | |
|---|---|
| | Añadir navegación |
| Dependencias | • Iniciar sesión<br>• Crear página |
| Precondición | • La plataforma debe estar configurada y operativa al menos en fase de desarrollo.<br>• El usuario debe haber iniciado sesión.<br>• El usuario debe tener permisos para añadir *portlets*.<br>• Deben haber creadas páginas *webs*. |
| Descripción | El sistema deberá comportarse como se describe en la siguiente secuencia: |
| Secuencia normal | Paso / Acción<br>1 / Abrir el panel de aplicaciones<br>2 / Añadir el portlet de navegación<br>3 / Configurar portlet ( Ver plantilla Configurar portlet navegación )<br>4 / Finaliza el caso de uso |
| Postcondición | Se debe visualizar el portlet de navegación en la página web. |
| Comentarios | Ninguno |

11. Quitar navegación

<table>
<tr><td></td><td>Quitar navegación</td></tr>
<tr><td>Dependencias</td><td>• Iniciar sesión<br>• Añadir navegación</td></tr>
<tr><td>Precondición</td><td>• La plataforma debe estar configurada y operativa al menos en fase de desarrollo.<br>• El usuario debe haber iniciado sesión.<br>• El usuario debe tener permisos para eliminar portlets.<br>• Deben haber creadas páginas webs.</td></tr>
<tr><td>Descripción</td><td>El sistema deberá comportarse como se describe en la siguiente secuencia:</td></tr>
<tr><td>Secuencia normal</td><td>
<table>
<tr><td>Paso</td><td>Acción</td></tr>
<tr><td>1</td><td>Acceder a la página donde se encuentra el portlet de navegación</td></tr>
<tr><td>2</td><td>Abrir configuración del portlet</td></tr>
<tr><td>3</td><td>Hacer click en Eliminar</td></tr>
<tr><td rowspan="2">4</td><td>Si se hace click sobre Aceptar,</td></tr>
<tr><td>4.1 El portlet se elimina de la página. Finaliza el caso de uso.</td></tr>
<tr><td rowspan="2">5</td><td>Si se hace click en  Cancelar,</td></tr>
<tr><td>5.1 El portlet no se eliminar y finaliza el caso de uso.</td></tr>
</table>
</td></tr>
<tr><td>Postcondición</td><td>El portlet se elimina de la página pero las páginas del portal no se eliminan.</td></tr>
<tr><td>Comentario</td><td>Ninguno.</td></tr>
</table>

12. Crear formulario web

<table>
<tr><td></td><td>Crear formulario web</td></tr>
<tr><td>Dependencias</td><td>• Iniciar sesión<br>• Crear página</td></tr>
<tr><td>Precondición</td><td>• La plataforma debe estar configurada y operativa al menos en fase de desarrollo.<br>• El usuario debe haber iniciado sesión.<br>• El usuario debe tener permisos para añadir portlets.<br>• Deben haber creadas páginas webs.</td></tr>
<tr><td>Descripción</td><td>El sistema deberá comportarse como se describe en la siguiente secuencia:</td></tr>
<tr><td>Secuencia normal</td><td>
<table>
<tr><td>Paso</td><td>Acción</td></tr>
<tr><td>1</td><td>Abrir el panel de aplicaciones</td></tr>
<tr><td>2</td><td>Añadir el portlet Web form</td></tr>
<tr><td>3</td><td>Acceder a la configuración del portlet Web form</td></tr>
<tr><td>4</td><td>Modificar propiedades titulo, descripción, requerir Captcha, correo electrónico al que se envían los formularios según las necesidades.</td></tr>
<tr><td rowspan="3">5</td><td>Configurar campos del formulario. Si se hace clic en Añadir campo</td></tr>
<tr><td>5.1 Se añade un nuevo campo.</td></tr>
<tr><td>5.2 Indicar tipo de campo, texto y opciones.</td></tr>
<tr><td rowspan="2">6</td><td>Si se hace click en Borra campo,</td></tr>
<tr><td>6.2 Se elimina el campo de la lista.</td></tr>
<tr><td>7</td><td>Hacer click en Guardar. Finaliza el caso de uso.</td></tr>
</table>
</td></tr>
<tr><td>Postcondición</td><td>Se debe visualizar el portlet con el formulario configurado.</td></tr>
<tr><td>Comentario</td><td>Si el portlet no se encuentra disponible, hay que activarlo desde</td></tr>
</table>

| | |
|---|---|
| | el panel de administración. |

13. Modificar formulario web

| | |
|---|---|
| | Modificar formulario web |
| Dependencias | • Iniciar sesión<br>• Crear página<br>• Crear formulario web |
| Precondición | • La plataforma debe estar configurada y operativa al menos en fase de desarrollo.<br>• El usuario debe haber iniciado sesión.<br>• El usuario debe tener permisos para modificar *portlets.*<br>• Deben haber formularios web creados con el portlet *Web form.* |
| Descripción | El sistema deberá comportarse como se describe en la siguiente secuencia: |
| Secuencia normal | <table><tr><td>Paso</td><td>Acción</td></tr><tr><td>1</td><td>Acceder a la página donde se encuentra el formulario que se desea editar</td></tr><tr><td>2</td><td>Abrir menú desplegable de configuración de portlet</td></tr><tr><td>3</td><td>Hacer click en editar</td></tr><tr><td>4</td><td>Modificar la configuración del *portlet*.</td></tr><tr><td>5</td><td>Hacer click en Guardar. Finaliza el caso de uso.</td></tr></table> |
| Postcondición | Se debe visualizar las modificaciones sobre el portlet de navegación. |
| Comentario | Ninguno. |

14. Eliminar formulario web

<table>
<tr><td></td><td>Eliminar formulario web</td></tr>
<tr><td>Dependencias</td><td>• Iniciar sesión<br>• Crear formulario web</td></tr>
<tr><td>Precondición</td><td>• La plataforma debe estar configurada y operativa al menos en fase de desarrollo.<br>• El usuario debe haber iniciado sesión.<br>• El usuario debe tener permisos para eliminar portlets.<br>• Deben haber creados formularios webs.</td></tr>
<tr><td>Descripción</td><td>El sistema deberá comportarse como se describe en la siguiente secuencia:</td></tr>
<tr><td>Secuencia normal</td><td>
<table>
<tr><td>Paso</td><td>Acción</td></tr>
<tr><td>1</td><td>Acceder a la página donde se encuentra el portlet de formulario web.</td></tr>
<tr><td>2</td><td>Abrir configuración del portlet</td></tr>
<tr><td>3</td><td>Hacer click en Eliminar</td></tr>
<tr><td>4</td><td>Si se hace click sobre Aceptar,</td></tr>
<tr><td>5</td><td>4.1 El portlet se elimina de la página. Finaliza el caso de uso.</td></tr>
<tr><td>6</td><td>Si se hace click en  Cancelar,</td></tr>
<tr><td></td><td>5.1 El portlet no se eliminar y finaliza el caso de uso.</td></tr>
</table>
</td></tr>
<tr><td>Postcondición</td><td>El portlet se elimina de la página.</td></tr>
<tr><td>Comentario</td><td>Ninguno.</td></tr>
</table>

15. Iniciar sesión

<table>
<tr><td></td><td>Iniciar sesión</td></tr>
<tr><td>Dependencias</td><td>Ninguna</td></tr>
<tr><td>Precondición</td><td>• La plataforma debe estar configurada y operativa al menos en fase de desarrollo.</td></tr>
<tr><td>Descripción</td><td>El sistema deberá comportarse como se describe en la siguiente secuencia:</td></tr>
<tr><td>Secuencia normal</td><td>
<table>
<tr><td>Paso</td><td>Acción</td></tr>
<tr><td>1</td><td>Acceder a la página donde se encuentra el portlet de login.</td></tr>
<tr><td>2</td><td>Introducir usuario y contraseña</td></tr>
<tr><td rowspan="2">3</td><td>Si los datos son correctos,</td></tr>
<tr><td>3.1 El usuario inicia sesión y finaliza el caso de uso.</td></tr>
<tr><td>4</td><td>Si los datos son incorrectos,</td></tr>
<tr><td></td><td>4.1 Se redirecciona de nuevo a la página y se avisa al usuario de que los datos son incorrectos.</td></tr>
</table>
</td></tr>
<tr><td>Postcondición</td><td>El usuario queda identificado en la plataforma.</td></tr>
<tr><td>Comentario</td><td>Ninguno.</td></tr>
</table>

16. Cerrar sesión

| | |
|---|---|
| | Cerrar sesión |
| Dependencias | • Iniciar sesión |
| Precondición | • La plataforma debe estar configurada y operativa al menos en fase de desarrollo.<br>• Debe de haber un usuario identificado. |
| Descripción | El sistema deberá comportarse como se describe en la siguiente |

<table>
<tr><td></td><td>secuencia:</td></tr>
<tr><td>Secuencia normal</td><td><table>
<tr><td>Paso</td><td>Acción</td></tr>
<tr><td>1</td><td>Hacer click en el menú de usuario</td></tr>
<tr><td>2</td><td>Hacer click en Salir</td></tr>
<tr><td>3</td><td>Finaliza el caso de uso.</td></tr>
</table></td></tr>
<tr><td>Postcondición</td><td>La sesión del usuario se finaliza y se redirige a la página principal del portal.</td></tr>
<tr><td>Comentario</td><td>Ninguno.</td></tr>
</table>

17. Crear listado de generación dinámica

<table>
<tr><td></td><td>Crear listado de generación dinámica</td></tr>
<tr><td>Dependencias</td><td>• Iniciar sesión<br>• Crear página<br>• Crear contenido web</td></tr>
<tr><td>Precondición</td><td>• La plataforma debe estar configurada y operativa al menos en fase de desarrollo.<br>• El usuario debe haber iniciado sesión.<br>• El usuario debe tener permisos para añadir portlets.</td></tr>
<tr><td>Descripción</td><td>El sistema deberá comportarse como se describe en la siguiente secuencia:</td></tr>
<tr><td>Secuencia normal</td><td><table>
<tr><td>Paso</td><td>Acción</td></tr>
<tr><td>1</td><td>Abrir el panel de aplicaciones</td></tr>
<tr><td>2</td><td>Añadir el portlet Publicador de contenidos.<br>Finaliza el caso de uso.</td></tr>
</table></td></tr>
<tr><td>Postcondición</td><td>El portlet Publicador de contenidos queda añadido a la página. Por defecto se muestran todos los contenidos webs.</td></tr>
</table>

| | |
|---|---|
| Comentario | Ninguno. |

18. Añadir filtrado a un listado de generación dinámica

| | |
|---|---|
| | Añadir filtrado a un listado de generación dinámica |
| Dependencias | • Iniciar sesión<br>• Crear página<br>• Crear listado de generación dinámica |
| Precondición | • La plataforma debe estar configurada y operativa al menos en fase de desarrollo.<br>• El usuario debe haber iniciado sesión.<br>• El usuario debe tener permisos para configurar *portlets.*<br>• Debe de haber un listado de generación dinámica |
| Descripción | El sistema deberá comportarse como se describe en la siguiente secuencia: |
| Secuencia normal | <table><tr><td>Paso</td><td>Acción</td></tr><tr><td>1</td><td>Acceder a la página donde se encuentra el listado que se desea editar</td></tr><tr><td>2</td><td>Abrir menú desplegable de configuración de portlet</td></tr><tr><td>3</td><td>Hacer click en configuración</td></tr><tr><td>4</td><td>Seleccionar el ámbito y tipo de contenido que se mostrará</td></tr><tr><td>5</td><td>Seleccionar las reglas que deben cumplir los contenidos webs que se mostrarán.</td></tr><tr><td>6</td><td>Seleccionar ordenación y agrupación.</td></tr><tr><td>7</td><td>Hacer click en guardar. Finaliza el caso de uso.</td></tr></table> |

| | |
|---|---|
| Postcondición | Al guardar las preferencias, el listado se genera de nuevo con los contenidos que cumplen con las reglas especificadas. |
| Comentario | Ninguno. |

## 5.2 Vista lógica

La vista de implementación no se va a analizar y modelar toda la plataforma debido a la complejidad interna que posee Liferay. En este apartado se va a mostrar y explicar el diagrama de clases de sobre un *portlet* propuesto para cumplir con un requisito descrito anteriormente:

- La plataforma debe de mostrar un listado de enlaces a *builds* que se encuentran alojadas en un servidor externo y que se actualice automáticamente.

El *portlet* deberá conectarse a una página web que posee el listado de las distribuciones en desarrollo del producto gvSIG Desktop y mediante *web scraping* mostrar la información en un listado de tablas, la primera tabla siempre será los archivos de la última distribución y a continuación se mostrará un listado con el resto de tablas que contendrán todas las demás *builds* con sus respectivos archivos. Las tablas, al igual que el resto del portal, deben ser *responsive y* tener un estilo acorde con el resto del portal. Además el *portlet* debe ser configurable para poder indicar que versión y distribución se quiere mostrar. La página web es la siguiente: http://downloads.gvsig.org/download/gvsig-desktop/ El listado de versiones del producto se encuentra dentro de la carpeta dists/ que a su vez dentro de cada versión en la carpeta builds/ se encuentra el listado de todas las distribuciones de la versión.

La técnica de *web scraping* consiste en extraer información de páginas web mediante algún tipo de software. Está enfocado a la transformación de datos en formato HTML para su posterior almacenamiento, análisis o visualización. Existen diversas técnicas de *web scraping*, en este portlet se va a utilizar un *parser* llamado Jsoup [jsoup]. Esta técnica puede vulnerar los términos y condiciones de sitios webs, por lo que se recomienda revisar estos términos y solicitar al autor de la página web el permiso correspondiente. El autor de este proyecto ha sido autorizado por la asociación gvSIG para usar la técnica de *web scraping* sobre la página http://downloads.gvsig.org/download/gvsig-desktop/.

La figura 5.2 muestra el diagrama de clases:

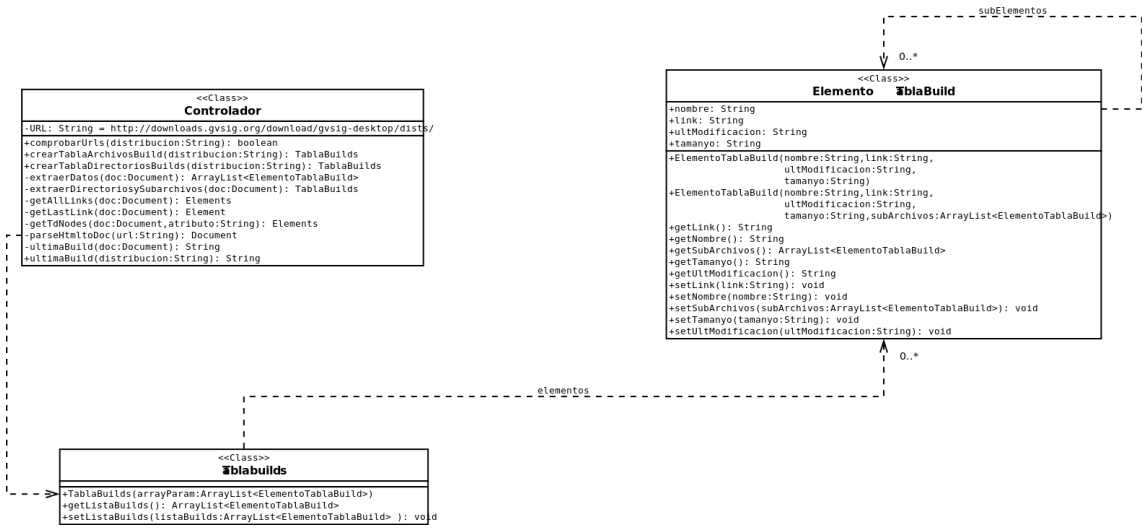



**Figura 5.2** Diagrama de clases del *portlet* descargasBuilds

El *portlet* está formado por tres clases:

- Controlador: clase situada entre la vista y la lógica de negocio. La capa de visualización interactúa con el controlador y el controlador con la capa de lógico. Tiene las siguientes operaciones:

  - public boolean comprobrarUrls(String)

  Devuelve *true* si la distribución se encuentra en el listado de distribuciones, en caso contrario devuelve *false.*

  - public TablaBuilds crearTablaArchivosBuilds(String)

  Devuelve un objeto de tipo *TablaBuilds* que contiene la información de los archivos de la última *build.*

  - public TablaBuilds creatTablaDirectoriosBuilds(String)

  Devuelve un objeto de tipo *TablaBuilds* que contiene la información de todas las *builds* de la distribución indicada.

  - private ArrayLisy<ElementoTablaBuild> extraerDatos(Document)

  A partir de un objeto Document crea una colección con toda la información interesante.

  - private TablaBuilds extraerDirectoriosySubarchivos(Document)

  A partir de un objeto *Document* se obtiene los datos de los directorios como nombre, link, última modificación, tamaño y archivos que contiene.

  - private Elements getAllLinks(Document)

  A partir de un objeto Document obtiene todos los links.

  - private Element getLastLink(Document)

  Obtiene el último link a partir de un objeto Document.

  - private Elements getTdNodes(Document, String)

A partir de un objeto Document y un propiedad obtiene todos los nodos con la propiedad especificada.

- private Document parseHtmltoDoc(String) throws IOException

A partir de una URL que recibe como parámetro, se obtiene un objeto Document con toda la información de la página.

- private String ultimaBuild(Document)

A partir de un objeto Document de una distribución se obtiene la última *build* de la misma.

- public String ultimaBuild(String)

A partir de una URL se obtiene el nombre de la última *build* o dirección.

- TablaBuilds: esta clase representa una tabla del *portlet* ya sea una tabla de archivos o directorios. Está formada por *ArrayList* de ElementoTablaBuild que representa cada uno de los elementos de la tabla. Contiene operaciones de *set y get* para acceder a los atributos privados.
- ElementoTablaBuilds: esta clase representa cada elemento de las tablas del portlet. Cada elemento está formado por un nombre, *link*, fecha de última modificación, tamaño y una colección en el caso de que el elemento se trate de un directorio. Contiene operaciones de *set* y *get* para acceder a los atributos privados.

## 5.3 Vista de implementación

En este apartado se ha modelo el paso de mensajes del *portlet* descargasBuilds entre la vista, el controlador y el modelo.

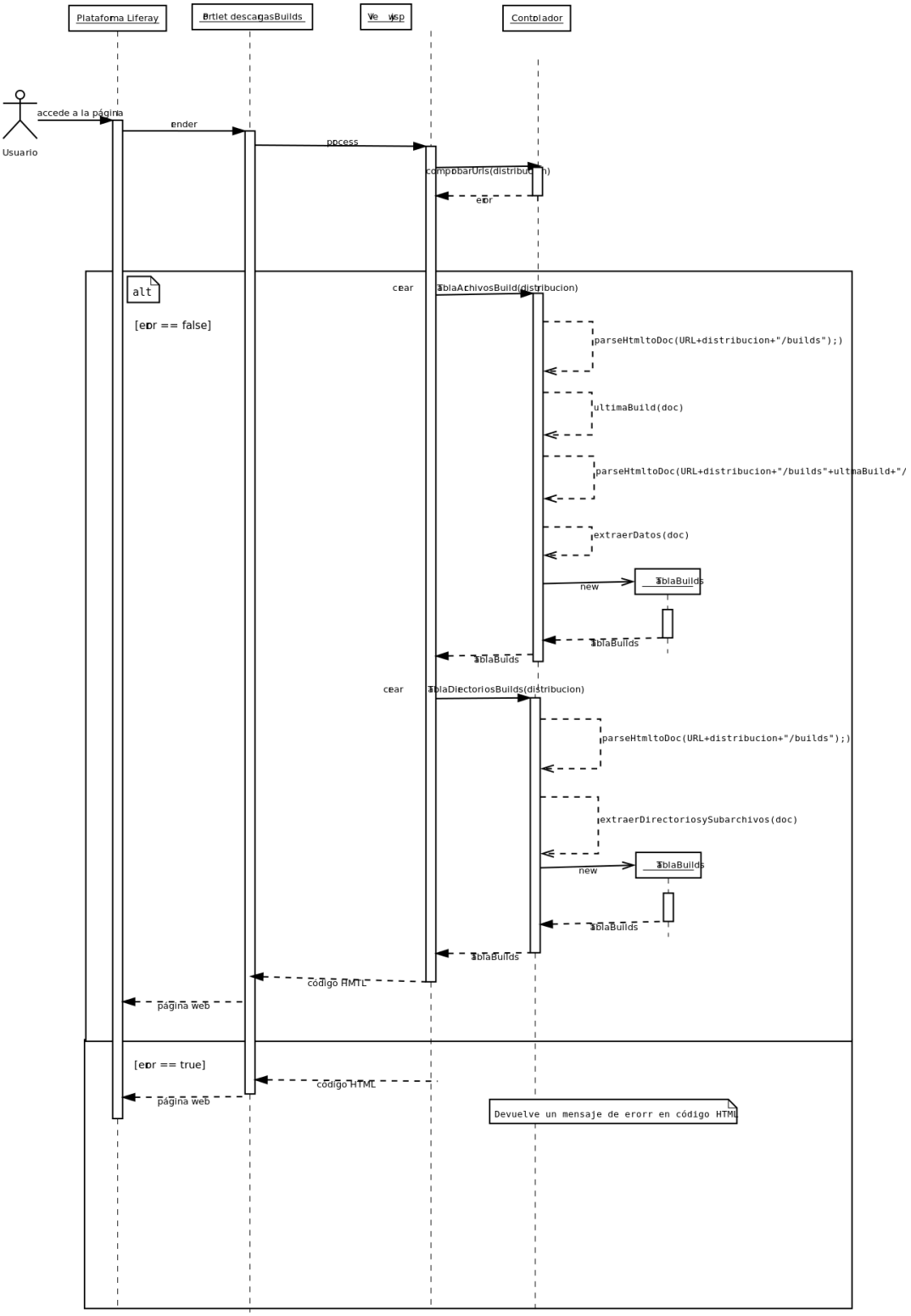



**Figura 5.3** Diagrama de secuencia del *portlet descargasBuilds*.

Cuando un usuario accede a la página donde se encuentra el *portlet*, la plataforma ejecuta la acción *render* de todos los *portlets* que se encuentran en la página. En este caso nuestro portlet descargasBuilds ejecuta el archivo v*iew.jsp* que actúa como vista y es el encargado de generar la parte de código HTML que se muestra en la página. El archivo instancia un controlador y comprueba que la distribución que recibe es correcta. Si la distribución es incorrecta genera código HTML indicando el error obtenido, en caso contrario llama a las operaciones *crearTablaArchivosBuilds* y *crearTablaDirectoriosBuilds*. El controlador ejecuta una serie de métodos para extraer y devolver dos objetos *TablaBuilds* que contienen la información de los archivos de la última distribución y la lista de todas las distribuciones con sus respectivos archivos. Una vez que la vista obtiene los objetos con la información relevante, los recorre y genera el código HTML con la información.

## 5.4 Vista física

Para entender el funcionamiento del sistema debemos saber los términos esenciales. Anteriormente, en la propuesta de solución, hemos visto que la arquitectura está compuesta por un servidor web Apache instalado en la máquina física, dos máquinas virtuales, una con Liferay junto con Tomcat y la otra con el gestor de base de datos PostgreSQL, y finalmente un sistema de almacenamiento en red. En términos generales, podemos decir que este sistema está compuesto por 3 tipos de componentes: servidores web, máquina virtual y almacenamiento en red (NAS). A continuación definiremos cada término y explicaremos brevemente el funcionamiento.

### 5.4.1 Servidor web

Según Wikipedia: "*Un **servidor web** o **servidor HTTP** es un programa informático que procesa una aplicación del lado del servidor, realizando conexiones bidireccionales y/o unidireccionales y síncronas o asíncronas con el cliente y generando o cediendo una respuesta en cualquier lenguaje o Aplicación del lado del cliente. El código recibido por el cliente suele ser compilado y ejecutado por un navegador web. Para la transmisión de todos estos datos suele utilizarse algún protocolo. Generalmente se usa el protocolo HTTP para estas comunicaciones, perteneciente a la capa de aplicación del modelo OSI."* [wsw14].

El servidor web nos proporciona una interfaz la cual es accesible desde cualquier navegador web (cliente) desde cualquier parte. El servidor web se queda en modo espera hasta que reciba alguna petición de algún navegador web. Una vez procesada la petición, el servidor le contestará enviando la página web solicitada que el navegador interpretará y mostrará al usuario. En la sección 4.2 de este capítulo se explica con más detalle que rol cumple exactamente el servidor web en nuestro sistema.

### 5.4.2 Máquina virtual

*"En informática una* ***máquina virtual*** *es un software que simula a una computadora y puede ejecutar programas como si fuese una computadora real. Este software en un principio fue definido como "un duplicado eficiente y aislado de una máquina física".* [wmv14].

Una máquina virtual, como indica la definición, permite la simulación de la existencia de otra máquina totalmente aislada pero que se ejecuta sobre el mismo hardware. Las máquinas virtuales tienen muchas utilidades, una de ellas es la posibilidad de ejecutar procesos en una máquina virtual para obtener un entorno de ejecución aislado del sistema. Esta funcionalidad en concreto es la usada por Java y su máquina virtual JVM para la ejecución de programas escritos en Java sin importar el sistema operativo de la máquina física. Otra funcionalidad bastante importante, es el ahorro en hardware que supone el alojar una máquina virtual y poder ejecutar procesos de forma aislada que en principio son incompatibles con otros que se ejecutan en la máquina física sin la necesidad de comprar nuevo hardware o contratarlo. En la sección 4.2 de este capítulo se explica con más detalle que rol cumple exactamente las máquinas virtuales en nuestro sistema.

### 5.4.3 Almacenamiento en red (NAS)

"***NAS*** *(del inglés* ***Network Attached Storage****) es el nombre dado a una tecnología de almacenamiento dedicada a compartir la capacidad de almacenamiento de un computador (Servidor) con computadoras personales o servidores clientes a través de una red (normalmente TCP/IP), haciendo uso de un Sistema Operativo optimizado para dar acceso con los protocolos CIFS, NFS, FTP o TFTP.*

*Generalmente, los sistemas NAS son dispositivos de almacenamiento específicos a los que se accede desde los equipos a través de protocolos de red (normalmente TCP/IP). También se podría considerar un sistema NAS a un servidor (Microsoft Windows, Linux,...) que comparte sus unidades por red, pero la definición suele aplicarse a sistemas específicos.*

*[…] Muchos sistemas NAS cuentan con uno o más dispositivos de almacenamiento para incrementar su capacidad total. Frecuentemente, estos dispositivos están dispuestos en* ***RAID*** *(****R****edundant* ***A****rrays of* ***I****ndependent* ***D****isks) o contenedores de almacenamiento redundante.*" [wnas14].

En nuestro sistema utilizaremos un almacenamiento en red (NAS) para el almacenamiento de todos los datos de nuestro portal. La principal ventaja que ofrece estos sistemas en el balanceo de carga y la tolerancia a fallos debido a que normalmente, como indica la definición, tienen varios discos organizados en sistemas RAID. Adelanto que en nuestro sistema disponemos un NAS con dos discos con RAID 1. En la sección 4.2 de este capítulo se explica con más detalle qué rol cumple exactamente el almacenamiento en red en nuestro sistema, qué nos ofrece y qué es RAID.

A continuación se muestra la figura 5.4, que simboliza la arquitectura del sistema.

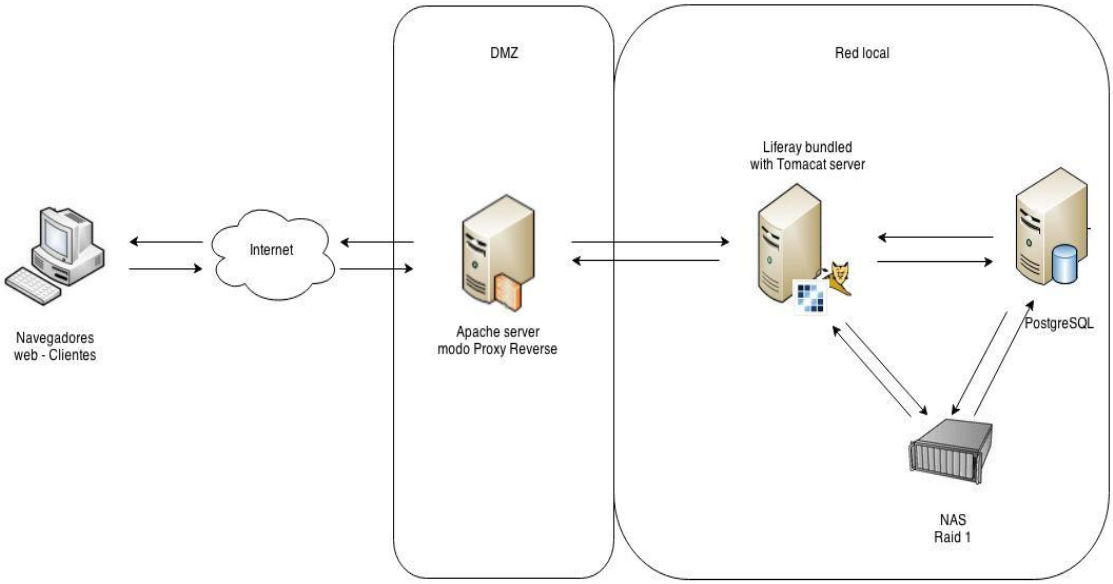



**Figura 5.4**. Arquitectura del sistema del portal gvSIG

Como podemos observa en la figura 4.1, el sistema está formado por 4 elementos: Apache web server, máquina virtual con Liferay con Tomcat, máquina virtual con el sistema gestor de base de datos PostgreSQL y finalmente un NAS con RAID 1.

### 5.4.4 Apache web server

El servidor web Apache se encuentra alojado en la máquina física. Pertenece a la zona desmilitarizada (DMZ en adelante) de nuestra arquitectura y es la puerta de entrada a nuestro sistema. La DMZ es una zona intermedia entre una red externa, en nuestro caso Internet, y la red local donde se encuentran nuestras máquinas virtuales. Entonces, ¿por qué nos es interesante esta zona? La DMZ nos permite establecer el servidor que debe ser accesible desde el exterior sin exponer la red local a posibles ataques o intrusos.

La función principal del servidor en la de **Proxy Reverse** redirigiendo el tráfico HTTP a nuestra máquina virtual Liferay. Esto nos aporta varias ventajas. Añade un nuevo nivel de seguridad protegiendo así los demás componentes. También cachea el contenido estático del portal web, mejorando el rendimiento general del portal descargando trabajo a la máquina virtual Liferay.

### 5.4.5 Máquinas virtuales

En el apartado 4.1.2 se ha introducido un poco en qué consiste una máquina virtual y que ventajas ofrece en término generales. El uso de máquinas virtuales nos aporta unas ventajas a nuestro sistema que son importantes detallarlas:

- Posibilidad de convivir con otros sistemas

  Existe la posibilidad de que en un mismo servidor puedan estar alojados más sistemas conviviendo con nuestros servidores o máquinas virtuales. Gracias al entorno aislado que

ofrecen las máquinas virtuales es posible que dos sistemas o tecnologías incompatibles puedan convivir dentro de una misma máquina.

- Fácil migración

  Otra de las ventajas más importantes es la posibilidad de migrar los sistemas a otra máquina de forma sencilla y rápida. Esto se debe a que se pueden crear imágenes de las máquinas virtuales e instalarlas en otro sistema. Supongamos que nuestro servidor se queda corto o ha sufrido un problema crítico y necesita ser reemplazado, gracias a la máquina virtuales solo habría que instalar la imagen de cada una de las máquinas instaladas en el servidor antiguo. En caso de que nuestro sistema no estuviera planteado con máquinas virtuales, habría que instalar y configurar cada servidor con todo el trabajo que eso supone.

**Liferay con Tomcat**

Máquina virtual con sistema operativo Debian 7. Tiene 2GB de memoria RAM y 25 GB de almacenamiento en disco duro. La máquina virtual tiene una instalación de Liferay 6.2 junto con Tomcat 7.0. La máquina responde al puerto HTTP 8080 y al AJP 8009.

**SGBD PostgreSQL**

Máquina virtual con el sistema de gestión de bases de datos PostgreSQL 9.3. PostgreSQL es un sistema de gestor de base de datos publicado bajo licencia BSD[5]. Una de las características más importantes de PostgreSQL es la alta concurrencia, mientras una conexión puede escribir otras conexiones pueden leer sin bloqueos. Cada usuario obtiene la versión del último *commit*. Otras características son: amplia variedad de tipos nativos, claves ajenas, disipadores...

### 5.4.5 NAS

Del inglés ***N**etwork **A**ttached **S**torage*, da nombre a la tecnología de almacenamiento en red dedicada a compartir recursos de almacenamiento entre los componentes de una red, haciendo uso de un sistema operativo optimizado para dar acceso por los protocolos de fichero más utilizados como por ejemplo CIFS, NFS, FTP, TFTP.

El papel principal del NAS es proporcionar almacenamiento de contenidos con copias de seguridad programadas en el mismo *datacenter*. Tanto los contenidos de la máquina virtual de Liferay como la máquina virtual con la base de datos tienen una copia de seguridad en el NAS.

Está formado por dos discos con RAID 1. RAID, del inglés ***R**edundant **A**rray of **I**ndependent **D**isks*, hace referencia a un sistema de almacenamiento de datos que usa múltiple unidades de almacenamiento entre los que se distribuye o replican datos. Según la configuración RAID, los beneficios varían entre: mayor integridad, mayor tolerancia a fallos, mayor rendimiento y mayor capacidad. Existen diferentes niveles de RAID. Nuestro sistema tiene un nivel de RAID 1: espejo. Un RAID 1 crea una copia exacta de un conjunto de datos, en dos o más discos. Con esto se consigue que la fiabilidad del sistema aumente de forma exponencial, ya que para que el sistema falle, lo deben hacer todos sus discos. De forma indirecta la velocidad de lectura aumenta linealmente al número de discos y la velocidad de escritura no se ve afectada, ya que los discos escriben la información de forma paralela.

5 Más información en Licencia BSD - Wikipedia

# Capítulo 6. Fase de construcción

En este capítulo pretende introducir los detalles de implementación del proyecto. Los objetivos de esta fase es especificar los campos de cada entidad de información y detallar las iteraciones para llevar a cabo la creación de las estructuras de datos, la creación de plantillas de visualización, implementación del *portlet* descargasBuilds y el entorno local provisional.

En la sección 6.1 se va especificar los campos de las entidades de información y su visualización en el portal, seguidamente en la sección 6.2, primera iteración, se explicará la creación de la entidades de información y la creación de la plantillas de visualización. En la sección 6.3, segunda iteración, se explicará de forma simplificada la implementación del *portlet* descargasBuilds y finalmente en la última iteración se explicará el proceso llevado a cabo para montar el entorno local provisional.

## 6.1 Especificación de entidades de información y su visualización

Antes de exponer las iteraciones dentro de la fase de construcción se debe especificar las entidades de información del portal así como su visualización dentro de este. Son necesarias las siguientes entidades de información: comunicación, versión, vídeo, certificación, evento, organización, comunidad, producto y noticia. Todos los bocetos que se muestran a continuación han sido realizado con la herramienta Balsamiq Mockups [bsmqmus]

- **Comunicación**: debe tener un campo para indicar un resumen/autores, una fecha y archivos asociados a la comunicación. Los archivos, deben tener una categoría, nombre, enlace al archivo, un campo que permita especificar un archivo alojado en la galería de la plataforma para permitir el enlace a un archivo externo o hacer uso de un archivo de la galería, icono y una descripción. La visualización sigue el siguiente boceto:

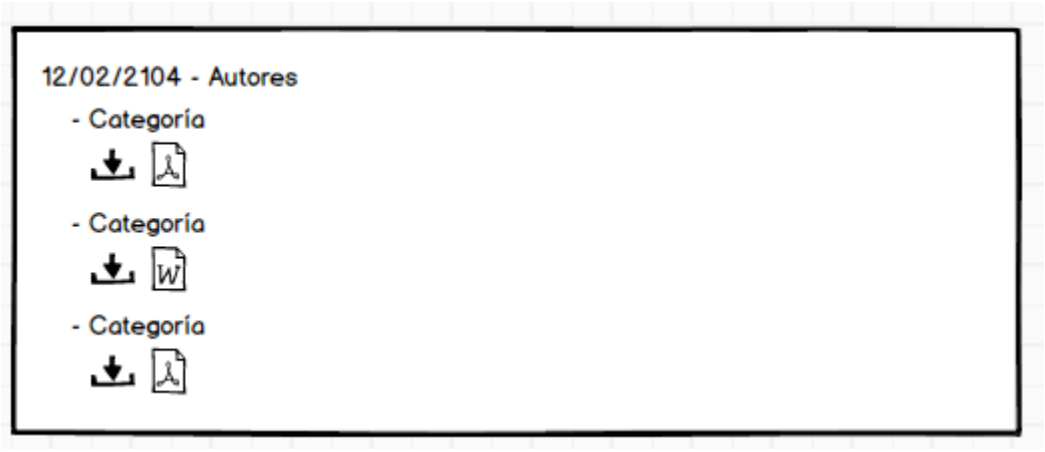
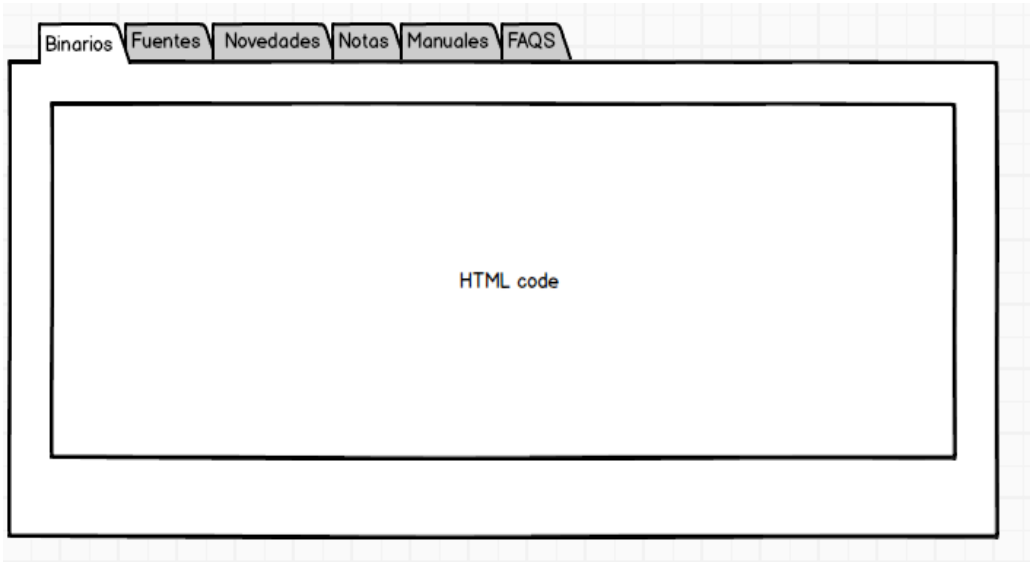
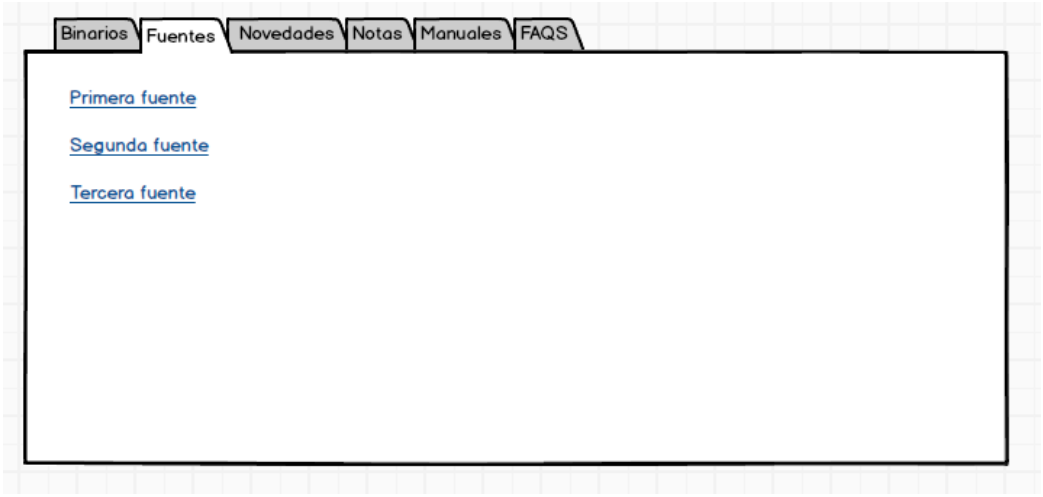


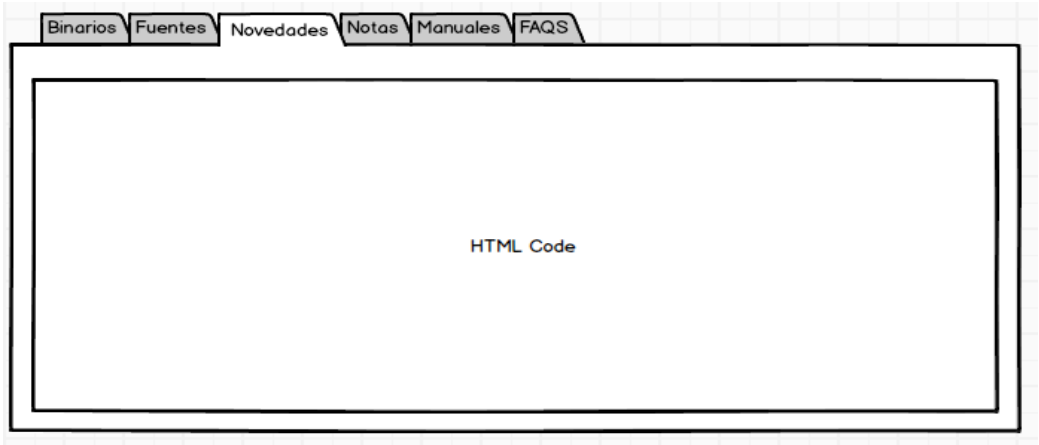

**Figura 6.1** Boceto de visualización de una comunicación

- **Versión**: debe tener una fecha, un campo para indicar los binarios de la versión, un conjunto de campos formados por un titulo, enlace y un acceso a la galería de la plataforma para definir las fuentes de la versión. También debe tener un campo para especificar las novedades de la versión, otro campo para especificar las notas, un conjunto de campos

formados por título, enlace, y acceso a la galería para indicar manuales y un conjunto de campos formado por pregunta y respuesta para indicar las FAQS de la versión. La visualización sigue el siguiente boceto:



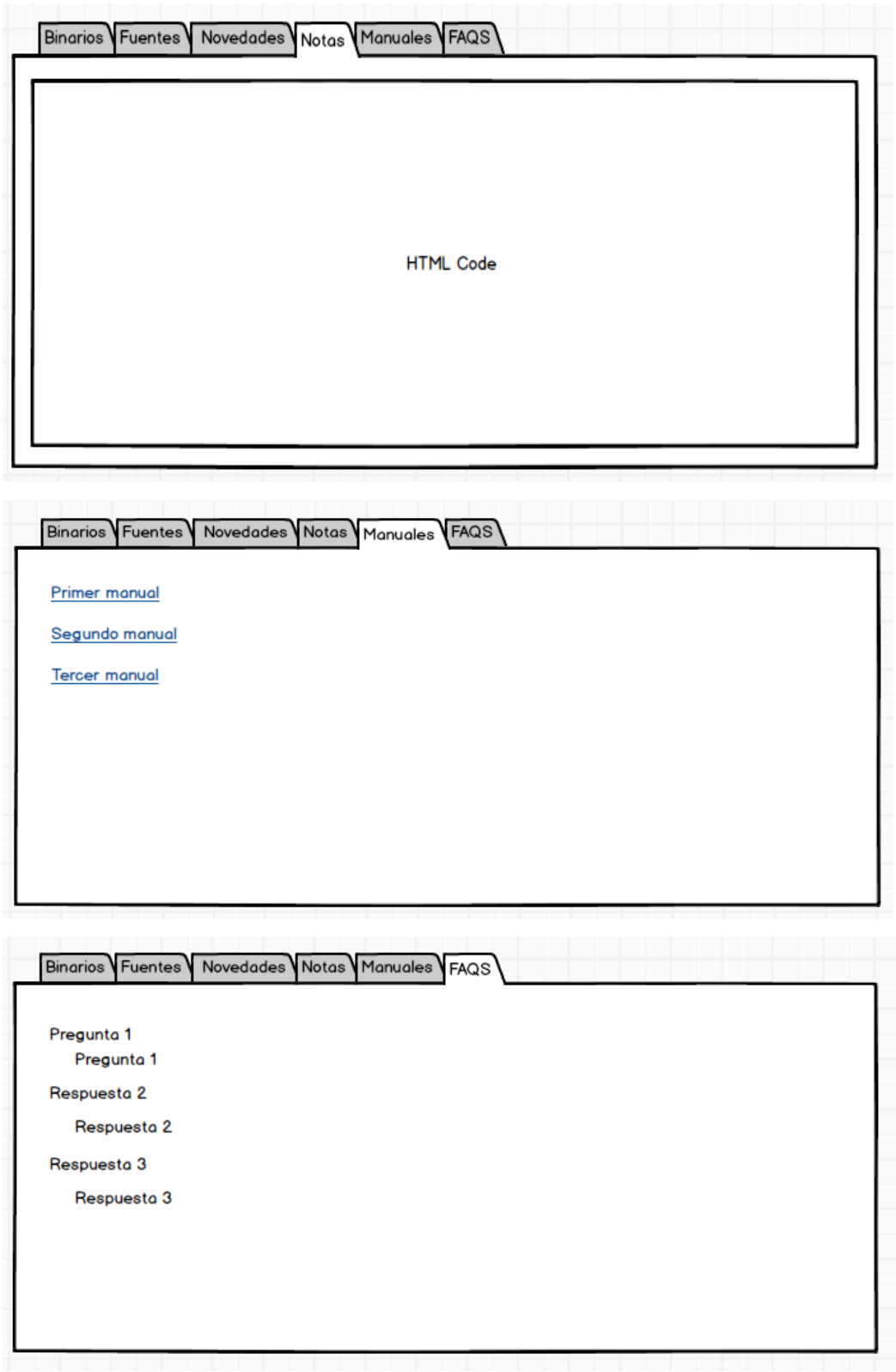



**Figura 6.2** Boceto de visualización de una versión

- **Certificación**: debe tener un campo para indicar los créditos del curso y los objetivos. También debe tener un conjunto de campos formado por créditos del módulo, el tipo, descripción del módulo y notas para indicar los todos los módulos de la certificación. La visualización sigue el siguiente boceto:

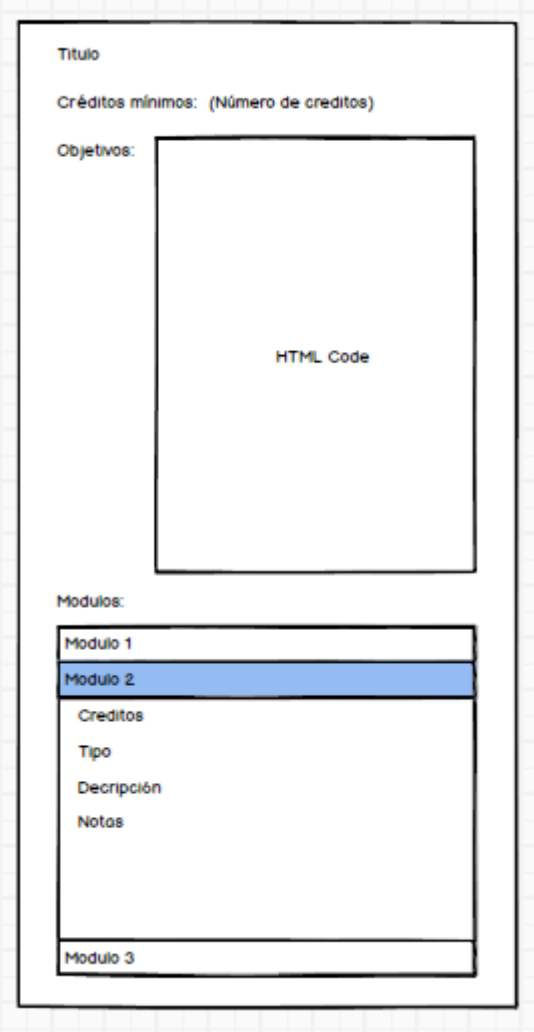



**Figura 6.3** Boceto de visualización de una certificación

- **Evento**: debe tener un acceso a la galería para indicar el cartel del evento, distintos editores de texto para indicar el autor, los objetivos, los comités, un acceso a la galería para la imagen de la sede, un editor de texto para indicar información sobre la sede, un editor de texto para indicar información acerca de como llegar, un editor de texto para indicar información acerca del alojamiento, un editor de texto para indicar información sobre qué ver y qué hacer, un conjunto de campos repetible formados por una fecha y una descripción para indicar las fechas significativas del evento, un editor de texto para indicar información acerca de la inscripción general al evento, dos campos de texto para indicar dos URLs para el formulario de inscripción y la zona de usuario. Además, un editor de texto para la descripción a la inscripción a talleres, un editor de texto para indicar el programa, las normas, como colaborar, *codesprints*, *bugsprints*, cena de gala y contacto. Finalmente un campo para especificar una página del portal donde se encuentran los colaboradores. La visualización de la información se debe dividir en diferentes páginas. La primera página se mostrará la imagen del evento y el autor en la parte inferior.

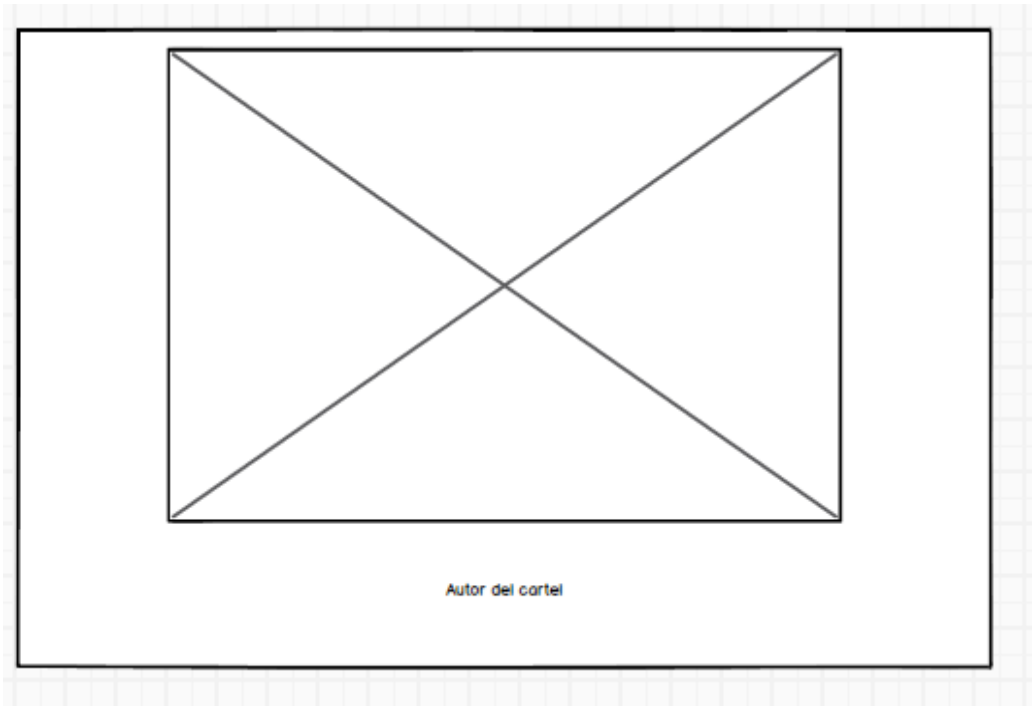



**Figura 6.4** Primer parte del boceto de visualización de un evento

La siguiente página mostrará los objetivos y comités, ambos mostrará el código HTML de la estructura:

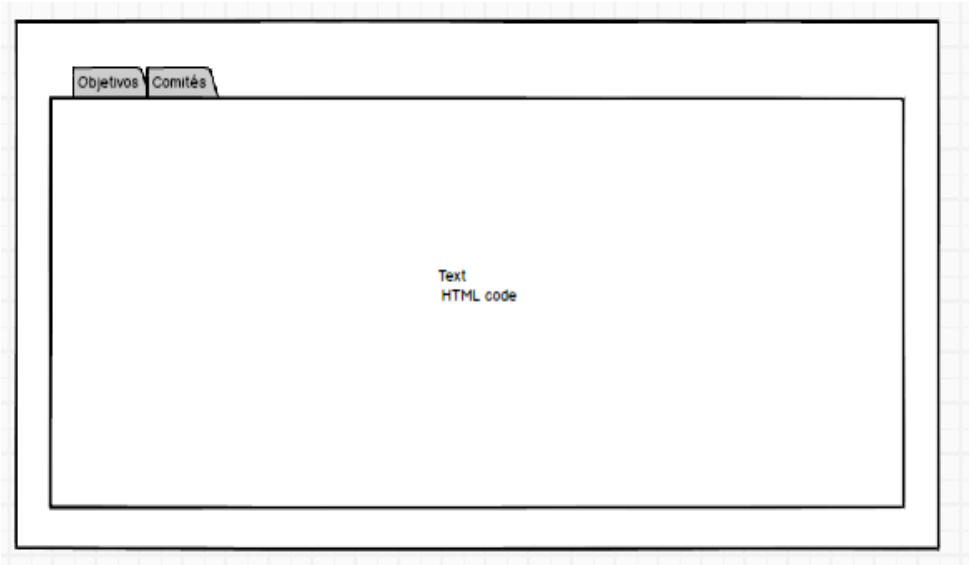



**Figura 6.5** Segunda parte del boceto de visualización de un evento

La siguiente página mostrará la sede, cómo llegar, alojamiento y qué ver y qué hacer del mismo modo que se la página anterior, en forma de pestañas horizontales mostrando el código HTML de la estructura directamente.

La siguiente página muestra las fechas significativas siguiendo el siguiente boceto:

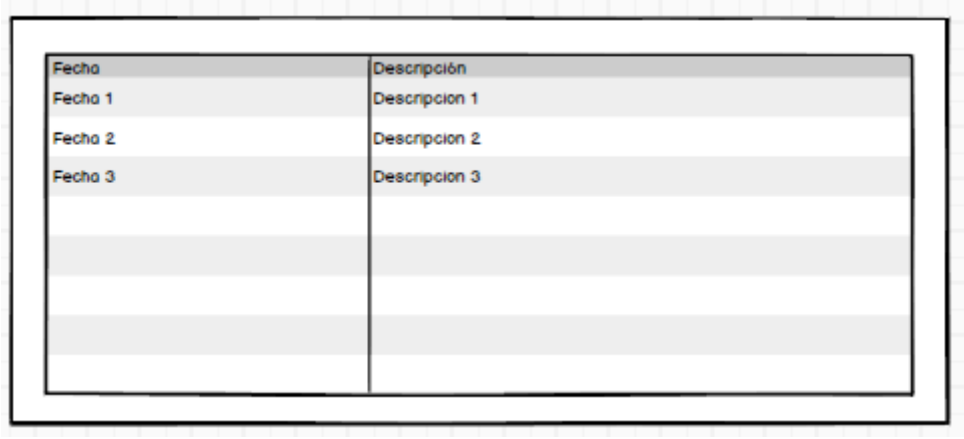
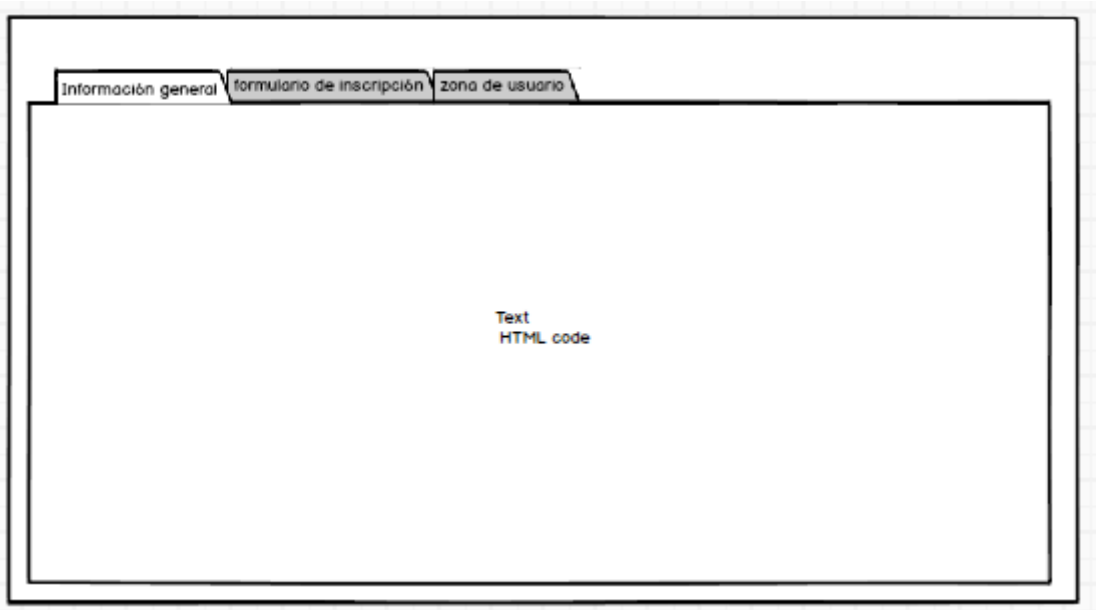
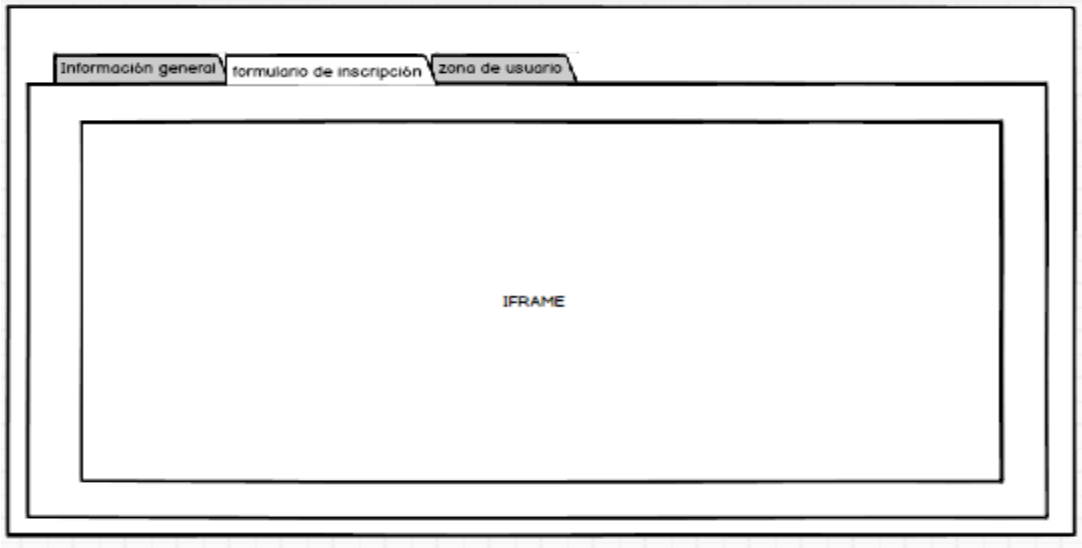



**Figura 6.6** Tercera parte del boceto de visualización de un evento

La siguiente página muestra información sobre la inscripción, zona de usuario y el formulario de inscripción en forma de pestañas horizontales siguiendo los siguientes bocetos:



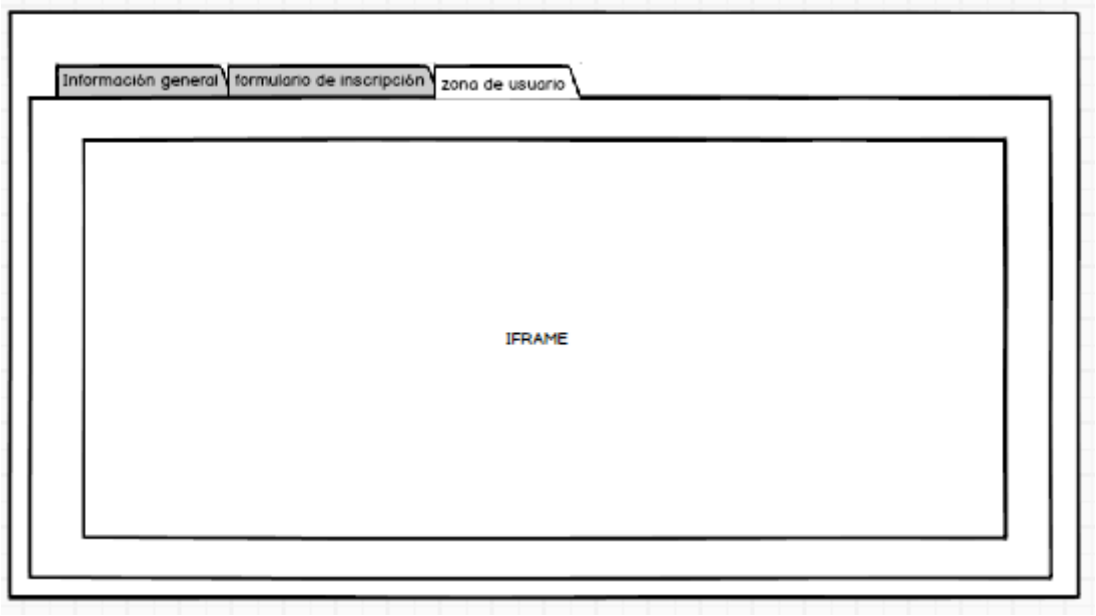



**Figura 6.7** Quinta parte del boceto de visualización de un evento

La siguientes páginas muestran el programa, patrocinadores y colaboradores, comunicaciones, cómo colaborar y contacto del mismo modo que se inserta en la estructura.

- **Organización**: debe tener acceso a la galería para indicar la imagen de la organización. Además un campo de texto para la indicar la dirección de la organización, otro campo de texto para indicar la página web y finalmente un editor de texto para escribir una descripción. La visualización sigue el siguiente boceto:

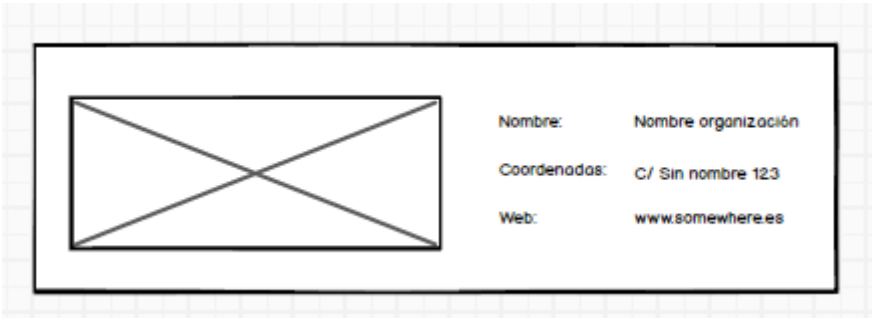



**Figura 6.8** Boceto de la visualización de una organización

- **Comunidad**: cada comunidad debe tener una imagen por lo que es necesario un campo que permita el acceso a la galería. Además, un editor para indicar descripción y objetivos de la comunidad, enlaces de interés, documentos. Finalmente, debe tener campos de texto para indicar los coordinadores de la comunidad. Cada coordinador tiene un nombre, una zona, correo electrónico, organización a la que pertenece y una responsabilidad/cargo. La visualización sigue el siguiente boceto:

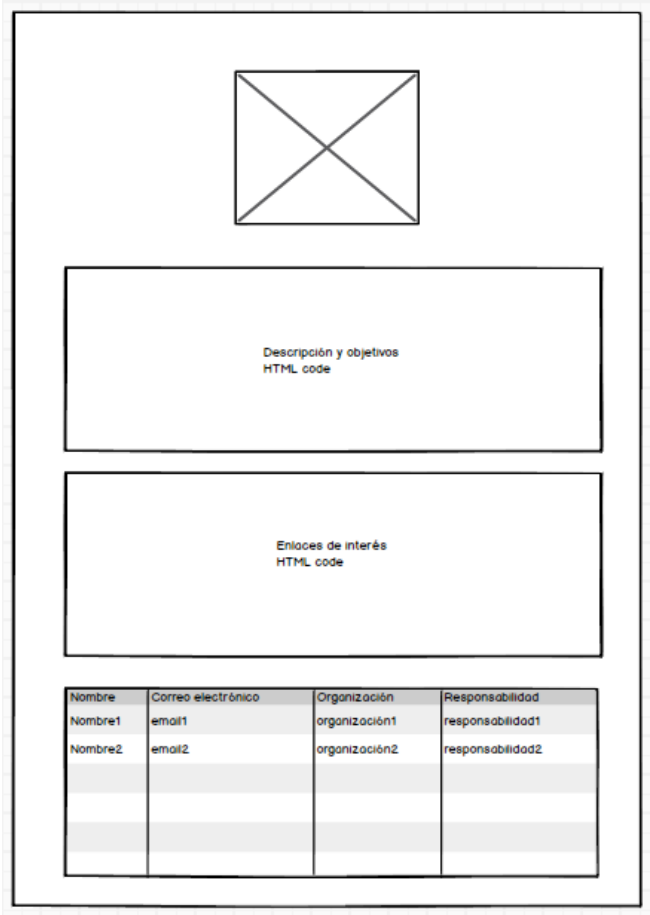



**Figura 6.9** Boceto de la visualización de una comunidad

- **Producto**: debe tener acceso a la galería para seleccionar la imagen del producto, editores para indicar la descripción, documentación, desarrollo y casos de uso. La visualización del producto se divide en varias páginas. La primera página debe seguir el siguiente boceto:

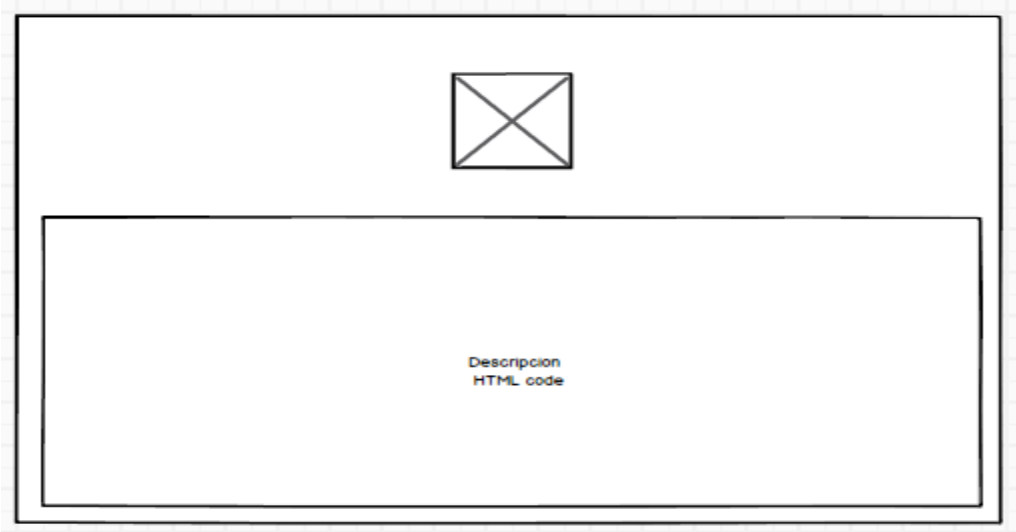



**Figura 6.10** Primera parte del boceto de visualización de un producto

Las siguientes páginas muestran toda la información de forma directa del editor de texto.

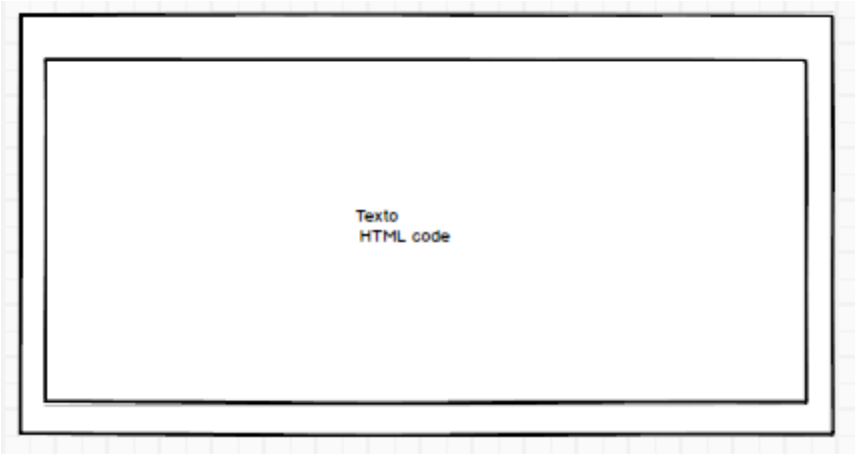



**Figura 6.11** Segunda parte del boceto de visualización de un producto

- **Noticia**: debe tener un editor para el texto de la noticia y un campo para indicar la fecha de la noticia.

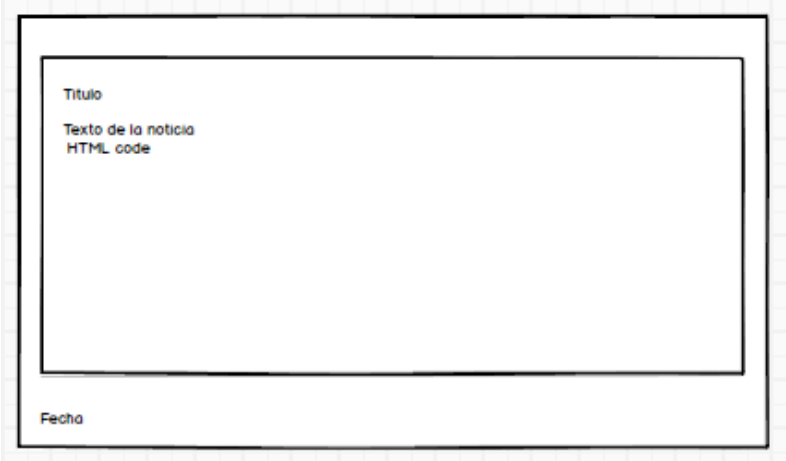



**Figura 6.12** Boceto de visualización de una noticia

## 6.2 1ª Iteración: creación de las entidades de información e implementación de las plantillas de visualización

Liferay establece una separación entre la información y como se visualiza dicha información. Por una parte, podemos definir **estructuras** para guardar información en diferentes tipos de campos y por otra parte podemos crear un numero indefinido de **plantillas de visualización** asociadas a una estructura que definen como se representa la información de la estructura asociada.

La creación de estructuras se realiza mediante el método de *drag and drop*. Este método permite al administrador del portal crear entidades de información de forma rápida y sencilla, simplemente con coger y arrastrar podemos crear nuevos campos dentro de la estructura. La figura 6.13 y 6.14 muestran la interfaz que se nos presenta para la creación de una estructura:


**Figura 6.13:** Interfaz para la creación de una nueva estructura

La creación de una nueva estructura precisa de un nombre (obligatorio), descripción y estructura padre (ambos opcionales). El nombre de la estructura y la descripción podemos especificarlo en diferentes idiomas.

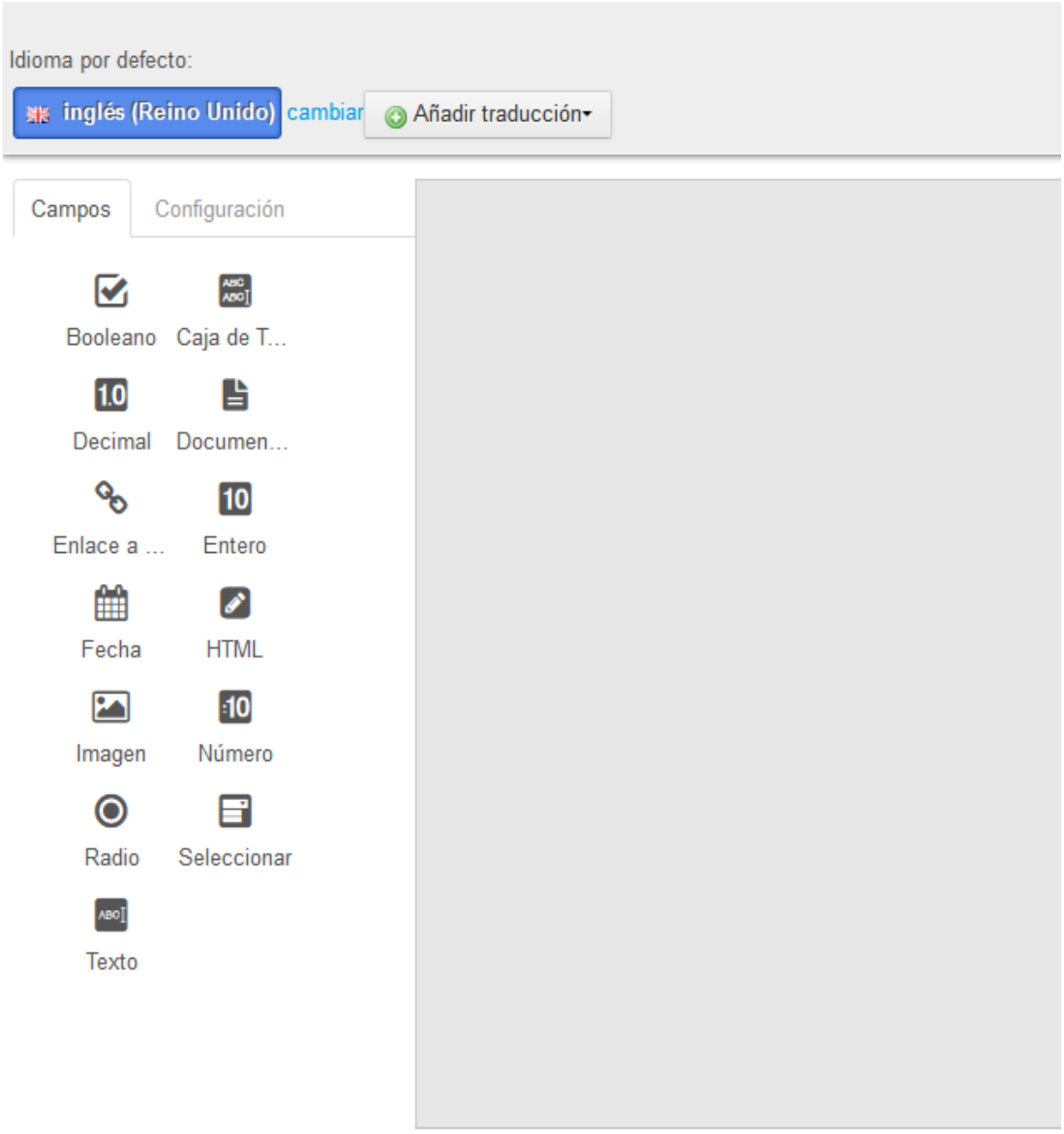



**Figura 6.14** Interfaz de creación de campos

En la figura 6.14 nos muestra el método de creación de campos dentro de una estructura. En la parte izquierda de la figura se visualizan los diferentes tipos de campos soportados por defecto en Liferay. Para crear un nuevo campo dentro de nuestra nueva estructura simplemente hay que seleccionar el tipo que más se ajuste y arrastrarlo hacia la parte derecha de la figura. Por ejemplo, si nuestra estructura necesita una fecha, tendremos que pinchar sobre el botón "Fecha" y arrastrarlo hacia la parte derecha. La figura 6.15 muestra el resultado de la operación:

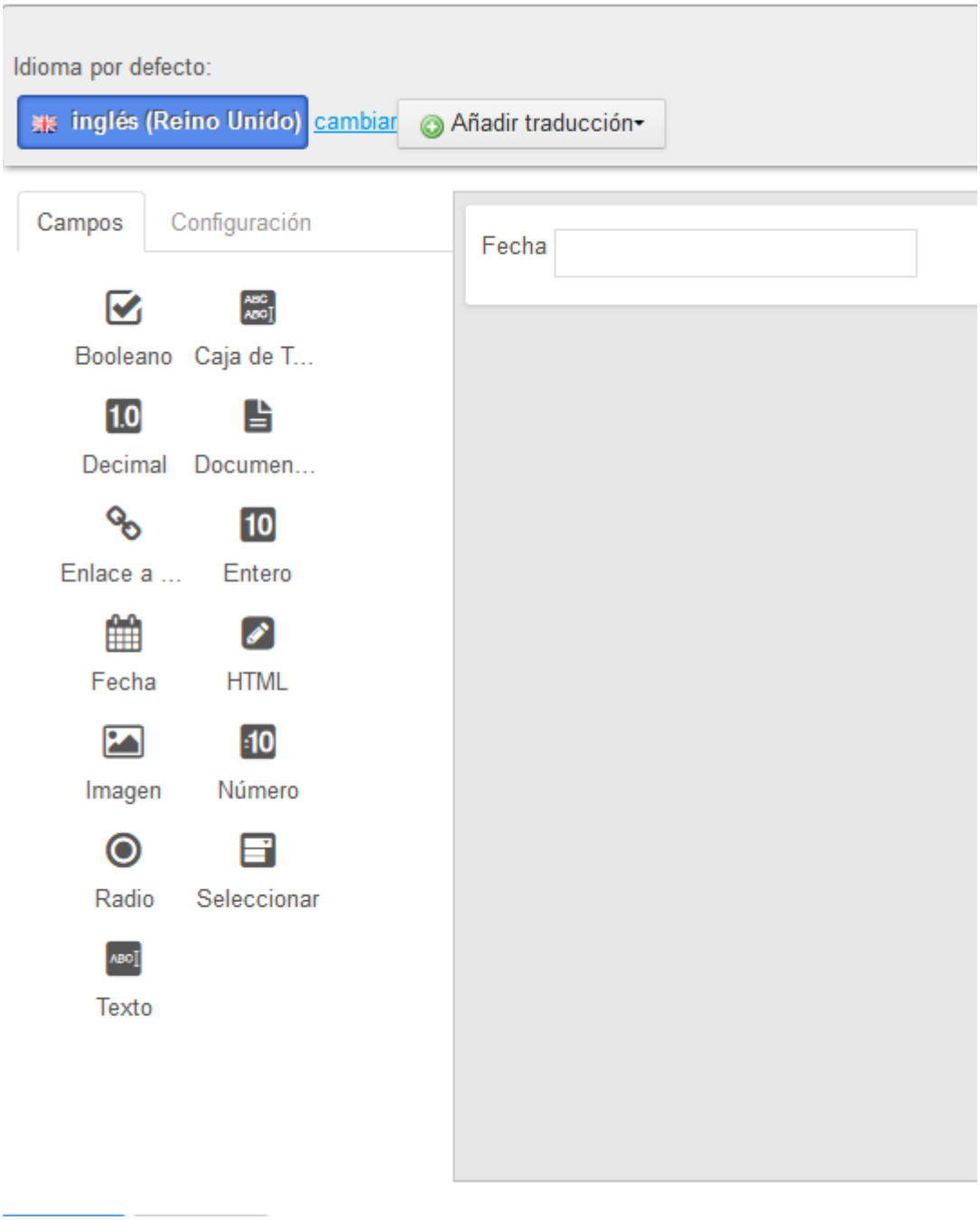



**Figura 6.15** Muestra el resultado de añadir un campo de tipo fecha a una estructura

Una vez añadido un campo, la interfaz de creación de estructuras nos permite añadir campos dentro de otros para mejorar el modelado de entidades de información. Por otra parte, la interfaz nos permite configurar distintas opciones sobre el campo según nuestros requerimientos. Para mostrar la configuración de un campo basta con hacer click con el botón primario sobre el campo. La figura 6.16 muestra las posibles opciones de configuración sobre el campo de tipo fecha:

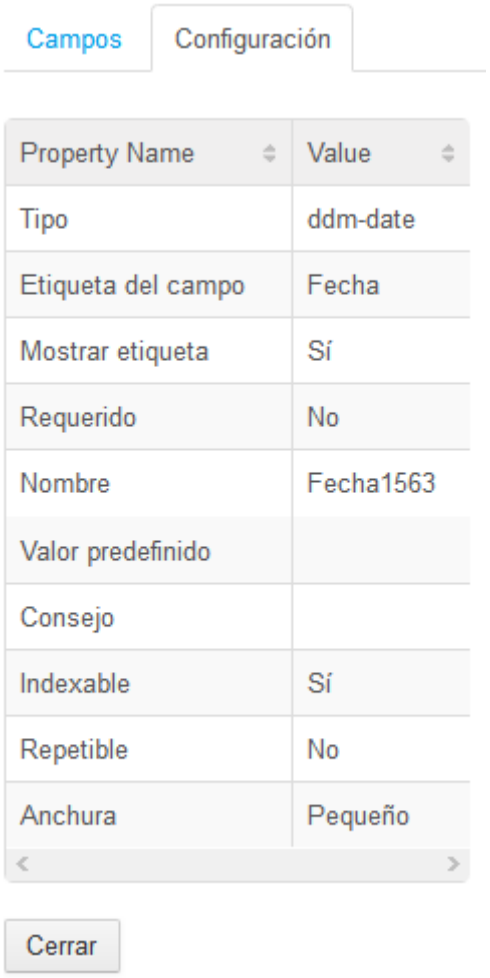



**Figura 6.16** Posibles configuraciones sobre un campo

Las diferentes opciones de configuración son las siguientes:

- Tipo: muestra el tipo de campo seleccionado. Este campo solo tiene valor informativo hacia el usuario que esté configurando los campos.
- Etiqueta del campo: permite cambiar el texto que aparece en la parte izquierda de cuadrado de texto.
- Mostrar etiqueta: ofrece la posibilidad de ocultar o no la etiqueta en la estructura.
- Requerido: permite configurar el campo seleccionado como requerido cuando se crean nuevos contenidos webs.
- Nombre: permite definir un nombre para el campo. Este nombre es usado en las plantillas de visualización para obtener los datos del campo o realizar comprobaciones. El nombre del campo debe ser único con respecto al resto de campos de la estructura.
- Valor predefinido: permite configurar el valor predefinido del campo.
- Consejo: permite mostrar un mensaje junto con el campo para indicar aspectos sobre el valor que se ha de introducir en el campo.

- Indexable: permite configurar si el contenido del campo se indexado por los buscadores ofrecidos por la plataforma.
- Repetible: permite indicar si el campo es repetible o no, es decir, si indicamos el campo repetible, cuando un administrador cree un contenido web con esta estructura podrá repetir este campo el número de veces que desee. Por ejemplo si dentro de una comunidad necesitamos definir campos para los socios, deberemos marcar los campos como repetibles para poder añadir tantos socios como sea necesario.
- Anchura: permite definir la anchura del campo.

A continuación se explica en detalle los pasos seguidos para la creación de una estructura en base a su especificación para comprender mejor el proceso. Se ha elegido la entidad de información **Comunidad** para el ejemplo. Recordemos lo especificado en la sección 6.1 Especificación de entidades de información:

- **Comunidad**: cada comunidad debe tener una imagen por lo que es necesario un campo que permita el acceso a la galería. Además, un editor para indicar descripción y objetivos de la comunidad, enlaces de interés, documentos. Finalmente, debe tener campos de texto para indicar los coordinadores de la comunidad. Cada coordinador tiene un nombre, una zona, correo electrónico, organización a la que pertenece y una responsabilidad/cargo.

En base a la especificación, debemos primero asignar un titulo y una descripción a nuestra estructura. El título será: "Comunidad" y la descripción "Representa una comunidad y sus coordinadores dentro de la asociación gvSIG.". La figura 6.17 muestra el resultado:


**Figura 6.17** Ejemplo de creación de la estructura Comunidad

Una vez indicado el título y la descripción debemos añadir los campos necesarios en base a la especificación. Los campos necesarios son: un campo de tipo "Documentos y multimedia" que nos proporciona el acceso a la galería de la plataforma para indicar la imagenes de la comunidad, tres edtiores HTML para la descripción y objetivos, enlaces de interés y documentos de la comunidadad. Además un campo de texto repetible para el nombre del coordinador que englobará el resto de campos de texto que definen un coordinador como son zona, correo electrónico, organización y cargo. La figura 6.18 muestra el resultado:

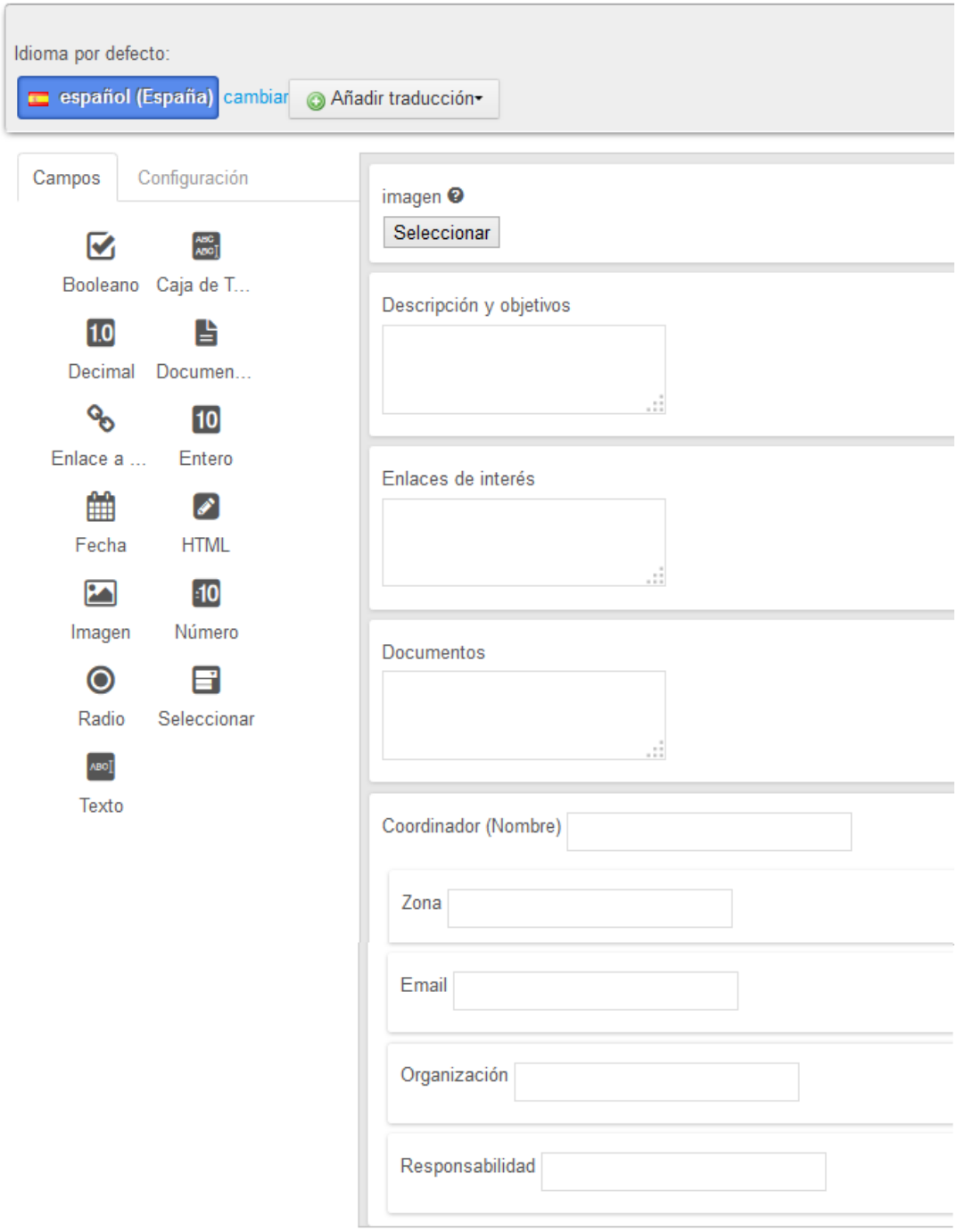



**Figura 6.18** Resultado de la estructura Comunidad

Una vez añadido todos lo campos necesarios debemos configurarlos uno a uno. En este caso las opciones de configuración más importantes serán el nombre de la etiqueta, el nombre identificativo y si es repetible o no. La figura 6.19 muestra la configuración del campo “Imagen”.

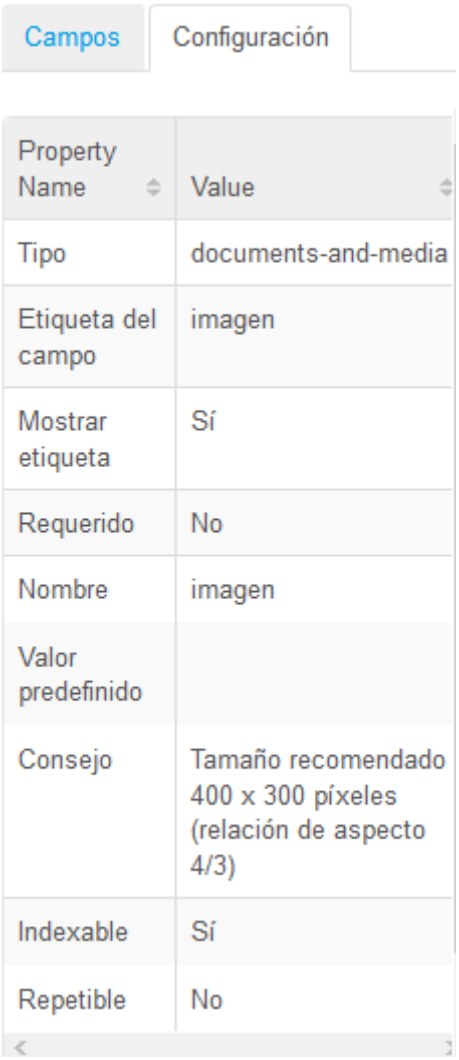



**Figura 6.19** Opciones de configuración del campo “Imagen”.

Se ha definido la etiqueta del campo “Imagen” para indicar al gestor de contenidos web del portal que tipo de archivo debe seleccionar. También se ha definido el nombre del campo para más adelante poder indentificarlo y acceder a el desde las plantillas de visualización y finalmente se ha indicado que el campo no sea repetible debido a las comunidades solo deben tener una imagen. [lugacst14]

Una vez creada nuestra esctructura, se debe crear tantas estructuras como entidades de información tengamos. Debido a que el proceso a seguir es el mismo para todas las estructuras no se va a detallar el proceso de creación del resto de estructuras.

A continuación, se va a detallar el proceso de creación de una plantilla de visualización. La figura 6.20 muestra la interfaz de creación de plantillas de visualización.

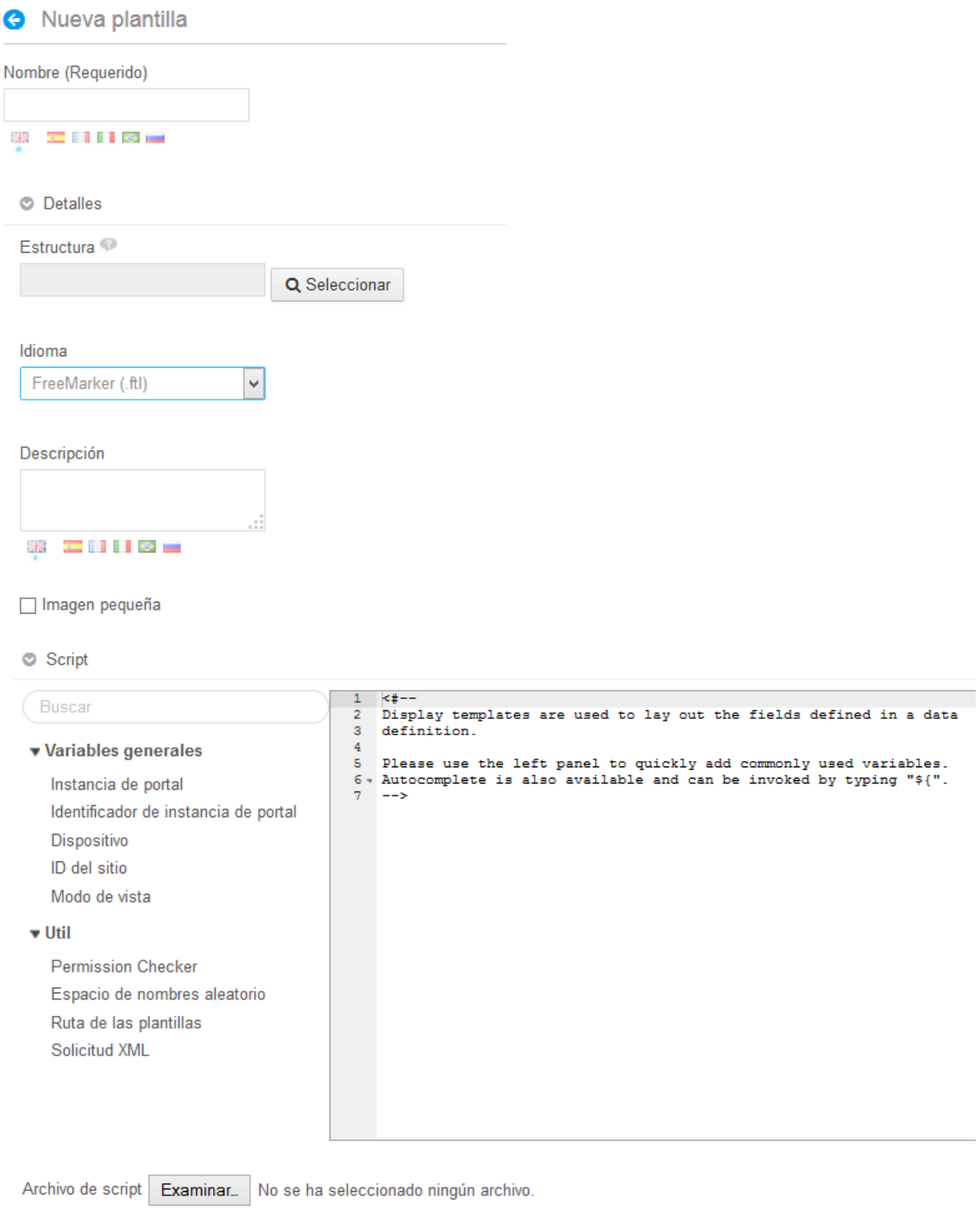



**Figura 6.20** Interfaz de creción de plantillas de visualización

Para crear una nueva plantilla de visualización debemos epsecificar el nombre de la plantilla, la estructura a la que va asociada, el lenguaje de la plantilla, una descripción (opcional), si tiene una imagen pequeña o no (opcional) y finalmente el código fuente. La plantilla permite definirse en dos idiomas: **Velocity** y **FreeMarker**. En este proyecto se ha decidido usar Velocity por su sencillez y su facilidad de mantener en el futuro con respecto a FreeMarker.

Siguiendo con el ejemplo anterior, se va a detallar el proceso de creación de la plantilla de visualización para la estructura Comunidad. El primer paso es indicar el titulo de la plantilla de

visualización: Comunidad. Seleccionamos la estructura anteriormente creada y seleccionamos Velocity en el campo idioma. Una vez indicado la información principal de la plantilla se procede a escribir el código Velocity para mostrar la información de la estructura.

**Velocity** permite generar páginas web dinámicamente sin mucho esfuerzo. Esta basado en directivas y en referencias. Las directivas empiezan con el carácter "#" mientras que las referencias empiezan por el carácter "$".

Las referencias en Velocity pueden ser de tres tipos: variables, propiedades de las variables y métodos. Las variables estan compuestas por el carácter "$" seguido de un identificador. Un ejemplo de variable sería *$pagina* . Las propiedades están formadas por el carácter "$" más un identificador seguido de un punto y otro identificador. Un ejemplo de propiedad *$pagina.titulo* . Y por último un método esta formado por el carácter "$" seguido de un identificador, seguido por un punto, otro identeificador y finalmente los parámetros necesarios del método entre paréntesis. Un par de ejemplos de métodos: *$cliente.getDireccion()* y *$pagina.setTitulo(" Nuevo titulo")*.

Las directivas permiten definir el flujo de ejecución de una plantilla y asignar valores a las referencias. Entre las directivas encontramos *set* usado para asignar valores a las referencias, bloques condicionales como *if-else*, *bucles* como *foreach* y *macro* que permiten definir segementos de código repetivo.

Una vez hecha una introducción básica, se va a implementar nuestra primera plantilla. Recordemos que el una Comunidad debe seguir el siguiente boceto especificado anteiormente:

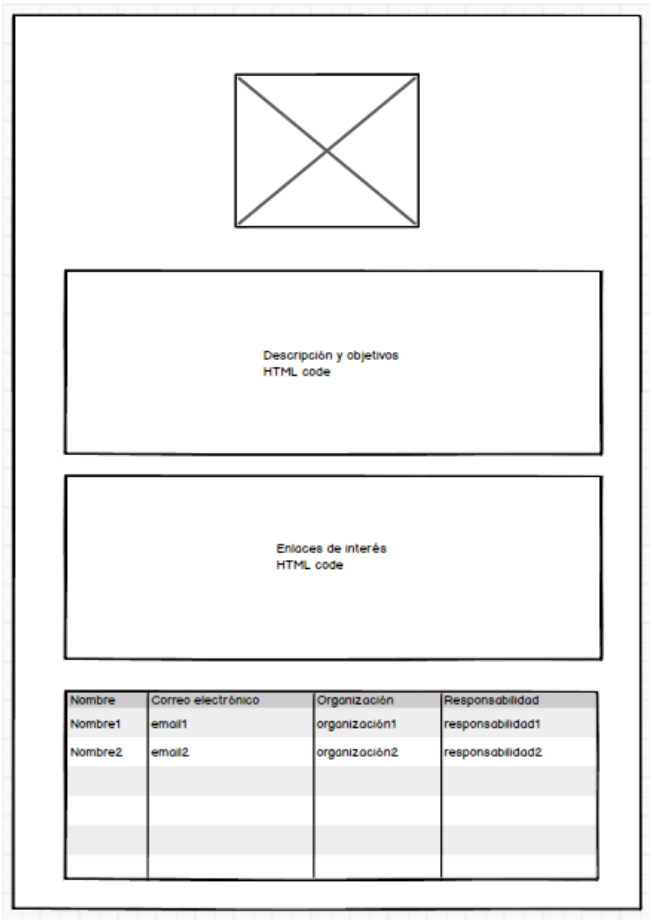



**Figura 6.21** Boceto de visualización de una comunidad

El primer paso será acceder a la imagen de la estrucutra y posteriormente escribir las etiquetas necesarias para que el navegador web muestre la imagen correctamente. La etiqueta necesaria para mostrar una imagen en HTML es: **img** que tiene un atributo **src** para indicar la URL de la imagen y un atributo opcional **alt** que muestra un mensaje de ayuda al dejar el puntero sobre la imagen.

Para acceder a la URL de la imagen debemos saber el nombre indicado anteriormente en la configuración del campo. El nombre asignado es: **imagen.** Para acceder a la URL, siemplemente debemos invocar al método ***getData()*** que devuelve la URL en forma de String. Para el atributo **alt** de la imagen debemos saber el titulo de la coumidad. El nombre de la comunidad se encuentra en el campo obligatiorio título de la estructura. El valor de este campo se encentra en la variable ***$reserved-artivle-title*.** Solamente queda escribir las etiquetas HTML para poder mostrar una imagen en una página web. La figura 6.22 muestra el resultado:

```
1  #if($validator.isNotNull($imagen.getData()))
2   <div class="image">
3      <img src="$imagen.getData()" alt="$reserved-article-title.getData()">
4  </div>
5  #end
```

**Figura 6.22** Primera parte plantilla comunidad

La variable ***$validator*** nos proporciona métodos de utilidad para realizar ciertas comprobaciones como por ejemplo si una variable es distinta de ***null***. Una vez insertada la imagen, el siguiente paso es añadir la descripción, objetivos y enlaces de interés. Recordemos que los campos descripción y objetivos y enlaces de interés son editores HTML, por lo que directamente el contenido de estos editores se debe mostrar en la página del portal. Simplemente deberemos acceder al contenido y añadir dos títulos. Los nombres de los campos son ***descripcionComunidad*** y ***enlacesComunidad***, accedemos al contenido del mismo modo que el anterior, con el método ***getData()*** y lo englobamos con las etiquetas correspondientes. Para los títulos, asignamos el tipo de titulo mediante etiquetas HTML e invocamos la directiva ***#language*** que nos busca en un registro de claves i18n para traducir la cadena que recibe como parámetro. El método de claves i18n se explicará en la fase de transición. La figura 6.23 muestra el resultado:

```
7  #if($validator.isNotNull($descripcionComunidad.getData()))
8      <h3> #language("descripcion_objetivos")</h3>
9      <p>$descripcionComunidad.getData()</p>
10 #end
11 #if($validator.isNotNull($enlacesComunidad.getData()))
12     <h3> #language("enlacesInteres") </h3>
13     <p>$enlacesComunidad.getData()</p>
14 #end
```

**Figura 6.23** Segunda parte plantilla comunidad

Finalmente solo queda representar todos los coordinadores de la comunidad en forma de tabla. Siguiendo el proceso anterior debemos acceder a la información de los campos de la estructura y añadir las etiquetas HTML **table, thead, tbody, th** y **td**. Es importante destacar que los campos que conforman un coordinador dentro de una comunidad están englobados por un campo que es repetible. Debido a esto deberemos recorrer todos los elementos repetidos que se corresponderán con cada una de las filas de la tabla. Para ello disponemos de la función **getSiblings()** que nos devuelve todos los elementos sobre un campo. Una vez obtenidos todos los coordinadores con el

método anterior, se deberá iterar sobre ellos y obtener la información de cada uno mediante la directiva *foreach*. Además se debe tener en cuenta que la tabla solo se debe mostrar en el caso de que exista un coordinador dentro de la comunidad. Para ello deberemos primero obtener todos los coordinadores, acceder al primero de la colección y comprobar que no es ***null.*** Finalmente, si algún campo de la estructura se encuentra vacío, no se debe mostrar nada. Esto se consigue añadiendo "!" al acceder a una variable. La figura 6.24 muestra el resultado:

```
#if(!$coordinador.getSiblings().isEmpty() && $validator.isNotNull($coordinador.getSiblings().get(0).getData()))
<h3>#language("coordinadores")</h3>
<table class="table table-striped">
    <thead>
        <tr>
            <th> #language("nombre") </th>
            <th> #language("email") </th>
            <th> #language("organizacion") </th>
            <th> #language("responsabilidad") </th>
            <th> #language("zona") </th>
        </tr>
    </thead>
    <tbody>
            #foreach ($item in $coordinador.getSiblings())
                <tr>
                    <td> $!item.getData() </td>
                    <td> $!item.email.getData() </td>
                    <td> $!item.organizacion.getData() </td>
                    <td> $!item.responsabilidad.getData() </td>
                    <td> $!item.zona.getData()</td>
                </tr>
            #end
    </tbody>
</table>
#end
```

**Figura 6.24** Tercera parte de la plantilla comunidad

El resto de plantillas del portal siguen el mismo proceso. Obtener la información para añadirle las etiquetas HTML necesarias para mostrarla como se requiera. Por motivos de simplificación de este trabajo no se va a mostrar el proceso de creación de todas las plantillas, ya que es muy similar al ejemplo mostrado. [vug14]

## 6.3 2ª Iteración: implementación del *portlet* descargaBuilds.

La implementación y despliegue de un portlet en Liferay necesita varias herramientas configuradas. Las herramientas necesarias son:

- Entorno de desarrollo: el entorno de desarrollo elegido para el desarrollo ha sido **Eclipse** debido a que es de código abierto y es compatible con otra de las herramientas necesarias.[eclipse]
- *Framework* de desarrollo: Liferay permite Struts, Spring MVC, JSF, entre otros. Se ha elegido **Liferay MVCPortlet** debido a que es fácil de entender, ligero y simple.
- Compilador: se ha usado **Ant** para compilar el código fuente y desplegar el *portlet* en el entorno Liferay [ant].
- Liferay *plugins SDK*. Liferay proporciona su propio **S***ofware* ***D****evelopment* ***K****it* para el desarrollo de *plugins*. [lsdk]
- Entorno Liferay. Necesario para realizar pruebas de funcionamiento.

Liferay nos proporciona otra herramienta que nos proporciona un interfaz gráfico para la creación de *plugins* dentro de Eclipse. Podemos instalarla desde el Eclipse *marketplace* y se llama **Liferay**

**IDE.** Esta herramienta es opcional ya que se pueden crear los *plugins* desde la consola con el compilador **Ant**.

Una vez instaladas y configuradas todas las herramientas, se ha de crear un *plugin* de tipo *portlet.* En Eclipse:

1. Ir a *New -> Other...*
2. Seleccionar Liferay *Plugin Project*

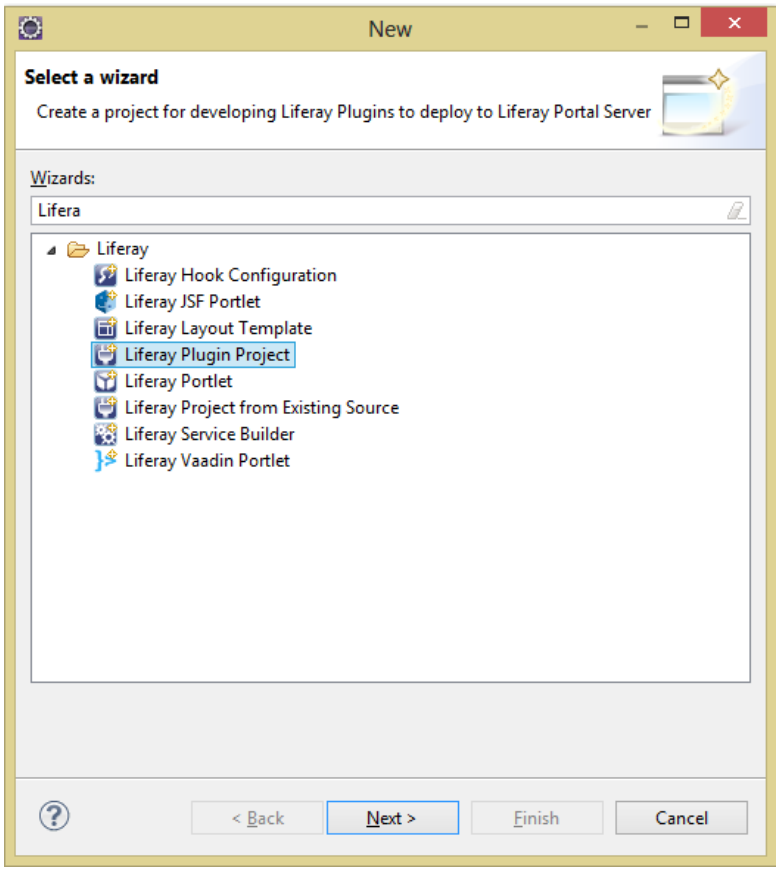



**Figura 6.25** Nuevo Liferay *Plugin Project*

3. Escribir el nombre del proyecto que en este caso será: *portletDescargasBuilds*.
4. Escribir el nombre con el que se mostrará dentro del entorno Liferay. En este caso será Descargas builds.
5. Seleccionar en *Build type* la opción Ant (liferay-plugins-sdk).
6. Seleccione la configuración de *Plugins* SDK.
7. Seleccione el Liferay *runtime* configurado.

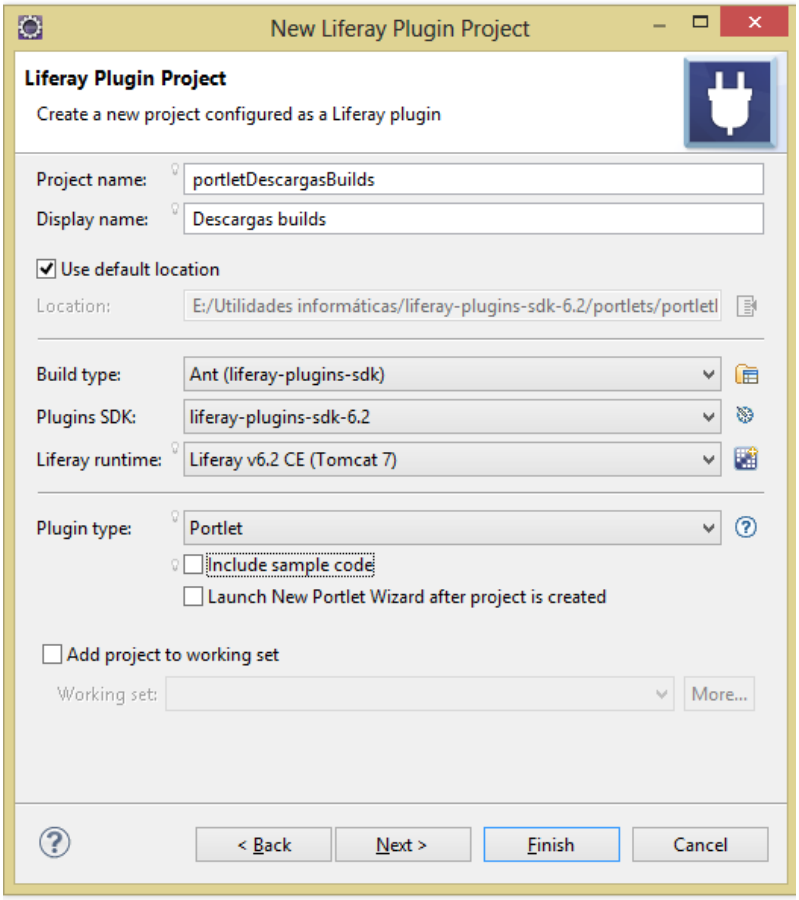



**Figura 6.26** Configuración del nuevo Liferay *plugin*

8. Haga click en *Finish*.

Una vez finalizada la creación del *portlet*, se va a explicar la estructura que ayudará a entender mejor como se ha implementado.

El proyecto está compuesto por tres componentes:

- Código Java
- Archivos de configuración
- Archivos del lado del cliente (archivos JavaScript, archivos CSS...)

La estructura creada a partir de asistente de creación de *portlet* es la siguiente:

- Nombre del *portlet*/
  - build.xml
  - docroot/
    - css
    - js
    - META-INF/

- WEB-INF/
  - lib/
  - src/ (No creada por defecto)
  - tld/
  - liferay-display.xml
  - liferay-plugin-package.properties
  - portlet.xml
  - web.xml
- view.jsp

El componente de código java debe alojarse en el directorio docroot/META-INF/src/. Los archivos de configuración deben alojarse en docroot/WEB-INF. Dentro de este directorio encontramos el archivo de configuración *portlet.xml*, además de otros archivos de configuración específicos de Liferay. Para nuestro *portlet* estos archivos son importantes ya que vamos a desplegar nuestro portlet sobre un entorno Liferay. Los archivos consisten en:

- *liferay-display.xml*: Describe la categoría del portlet dentro del entorno de Liferay. Las categorías sirven para agrupar *portlets* y facilitar al usuario la búsqueda e inclusión en el portal.
- *liferay-plugin-package.properties:* Describe varias opciones sobre el despliegue en caliente. El termino desplegar en caliente significar desplegar el *portlet* en el entorno de Liferay mientras este está en funcionamiento.
- *liferay-porlet.xml:* Describe varias opciones sobre el portlet tales como el icono del portlet, directorios de archivos del lado del cliente, posibles roles dentro del portlet, etc.

Finalmente los archivos por parte del cliente son archivos JavaScript, CSS y JSP que son interpretados por los navegadores de los usuarios. Estos archivos deben alojarse dentro de sus respectivas carpetas exceptuando los archivos JSP, por lo tanto los archivos JavaScript dentro de *docroot/WEB-INF/js*, los archivos CSS dentro de *docroot/WEB-INF/css* y finalmente los archivos JSP en *docroot.* [ldgapp14]

Después de esta breve introducción a la estructura de un *portlet*, se va a explicar como se va a integrar nuestro modelo de entidades con la estructura del portlet. Vamos a seguir el patrón **modelo-vista-controlador (MVC** en adelante**)**. El patrón MVC separa la lógica de los datos y de la visualización. En nuestro caso, por un lado el modelo se corresponde con los datos, la lógica la proporciona nuestro controlador y finalmente la vista los archivos JSP del *portlet* que nos permiten unir código java con código HTML para generar páginas web dinámicamente.

Una vez explicado brevemente el patrón a seguir, vamos a crear nuestras clases *ElementoTablaBuild y TablaBuilds* para definir el modelo, seguidamente crearemos el *ControladorDescargasBuilds* que nos proporcionará la lógica de negocio usada por la vista para representar el modelo. Y por último la vista, los archivos JSP necesarios para mostrar los datos.

La figura 6.27 muestra la interfaz *TablaBuilds*:

```java
/*
 * Copyright 2014 DiSiD Technologies S.L.L. All rights reserved.
 */
package com.liferay.descargasbuild.api;

import java.util.ArrayList;

public interface TablaBuilds {

	/**
	 * Obtiene el ArrayList de la clase con los elementos de la tabla.
	 *
	 * @return ArrayList con los elementos de la tabla.
	 */
	public ArrayList<ElementoTablaBuild> getListaBuilds();

	/**
	 * Se asigna un ArrayList con los elementos de la tabla al objeto que recibe
	 * como parametro.
	 *
	 * @param listaBuilds ArrayList que se quiere asignar a este objeto.
	 */
	public void setListaBuilds(ArrayList<ElementoTablaBuild> listaBuilds);

}
```

**Figura 6.27** Interfaz *TablaBuilds*

La figura 6.28 muestra la interfaz *ElementoTablaBuild*:

```java
/*
 * Copyright 2014 DiSiD Technologies S.L.L. All rights reserved.
 */
package com.liferay.descargasbuild.api;

import java.util.ArrayList;

public interface ElementoTablaBuild {

	/**
	 * Obtiene el link del elemento
	 *
	 * @return Link del elemento
	 */
	public String getLink();

	/**
	 * Obtiene el nombre del elemento
	 *
	 * @return Nombre del elemento.
	 */
	public String getNombre();

	/**
	 * Obtiene la coleccion de archivos dentro del directorio
	 *
	 * @return Colleccion de archivos que se encuentran dentro del directorio.
	 */

	public ArrayList<ElementoTablaBuild> getSubArchivos();
```

```java
/**
 * Obtiene el tamaño del directorio
 *
 */
public String getTamanyo();

/**
 * Obtiene la ultima modificacion del directorio
 *
 * @return Ultima modificacion del directorio
 */

public String getUltModificacion();

/**
 * Asigna un link a un elemento
 *
 * @param link Link que se quiere asignar
 */
public void setLink(String link);

/**
 * Asigna el nombre a un elemento
 *
 * @param nombre Nombre que se quiere asignar.
 */
public void setNombre(String nombre);

/**
 * Asgina la coleccion de archivos que se encuentran en el directorio.
 *
 * @param subArchivos
 */

public void setSubArchivos(ArrayList<ElementoTablaBuild> subArchivos);

/**
 * Asigna el tamaño a un elemento.
 *
 * @param tamanyo Tamaño que se quiere asignar.
 */
public void setTamanyo(String tamanyo);

/**
 * Se asigna la ultima modificacion al elemento.
 *
 * @param ultModificacion Fecha que se quiere asginar como ultima modificacion
 */

public void setUltModificacion(String ultModificacion);
```

**Figura 6.28** Interfaz *ElementoTablaBuild*

Las dos interfaces ofrecen métodos para obtener los atributos de los objetos y asignar valores u objetos a los atributos. La implementación de estas dos interfaces resulta bastante sencilla ya que cuando el método sea de tipo *get* simplemente debemos devolver el valor y en caso de que el método sea de tipo *set* debemos asignar el valor que recibe como parámetro al atributo de la clase. La figura 6.29 muestra un ejemplo de implementación:

```java
@Override
public String getNombre() {
    return nombre;
}

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

**Figura 6.29** Ejemplo de implementación

Una vez descrito el modelo, vamos a crear la interfaz *ControladorDescargasBuilds*. La figura 6.30 muestra la interfaz *ControladorDescargasBuild:*

```
package com.liferay.descargasbuild.api;


public interface ControladorDescargasBuilds {

    /**
     * A partir de una distribucion permite la comprobacion de si existe y es
     * correcta.
     *
     * @param version Version que se quiere comprobar.
     * @return True en caso de que sean incorrecta, false en caso contrario.
     */
    public boolean comprobarUrls(String version);

    /**
     * A partir de una version crear la tabla con los archivos con su link,
     * nombre, ultima modificacion y tamaÃ±o de la ultima compilacion.
     *
     * @param version Version de la cual se quiere obtener la tabla de
     *          su ultima compilacion.
     * @return Objeto TablaBuilds con la informacion de cada elemento.
     */
    public TablaBuilds crearTablaArchivosBuild(String version);

    /**
     * A partir de una version crear la tabla con todos los directorios con
     * su link, nombre, ultima modificacion, tamaÃ±o y archivos dentro del
     * directorio.
     *
     * @param version Version de la cual se quiere obtener la tabla de
     *          todos los directorios.
     * @return Objeto TablaBuilds con la informacion de todos los directorios y
     *         sus archivos.
     */
    public TablaBuilds crearTablaDirectoriosBuilds(String version);

    /**
     * A partir de una url se obtiene el nombre de la ultima build o direccion.
     *
     * @param version Version que se quiere obtener el
     *          nombre de la ultima compilacion
     * @return Nombre de la ultima compilacion.
     */
    public String ultimaBuild(String version);

}
```

**Figura 6.30** Interfaz *ControladorDescargasBuilds.*

Definidas las operaciones disponibles vamos a implementarlas. Las operaciones que ofrecen son las siguientes:

- *comprobarUrls( String version )*

  A partir de una versión comprueba si existe y es correcta. Esta operación es útil para comprobar que el usuario, al configurar el *portlet*, ha introducido una versión correcta. La figura 6.31 y 6.32 muestra la implementación de la operación:

```
    @Override
    public boolean comprobarUrls(String version) {
        try {
            parseHtmltoDoc(URL + version + "/builds/");
        } catch (IOException e) {
            return true;
        }
        return false;
    }
```

**Figura 6.31** Primera parte de la implementación de la operación *comprobarUrls.*

```
/**
 * A partir de un URL se obtiene el document DOM
 *
 * @param url
 *            Url de la pagina web fuente.
 * @return Document DOM de la pagina fuente.
 * @throws IOException
 */
private Document parseHtmltoDoc(String url) throws IOException {
    return Jsoup.connect(url).get();
}
```

**Figura 6.32** Segunda parte de la implementación de la operación *comprobarUrls.*

El primer paso será construir la URL completa a partir de una versión. El controlador posee como constante la URL base de donde sacar la información de las versiones, *builds* y archivos. La versión que recibe como parámetro la concatena a la URL base y le añade al final "/builds/" para comprobar que además de que la versión es correcta posee distribuciones para extraer los datos. El método delega la comprobación en un método en la librería JSoup. Si al ejecutar la sentencia *Jsoup.connect(url).get()* no lanza ninguna excepción significa que la versión introducida es correcta, en caso de que se lance una excepción la URL será incorrecta.

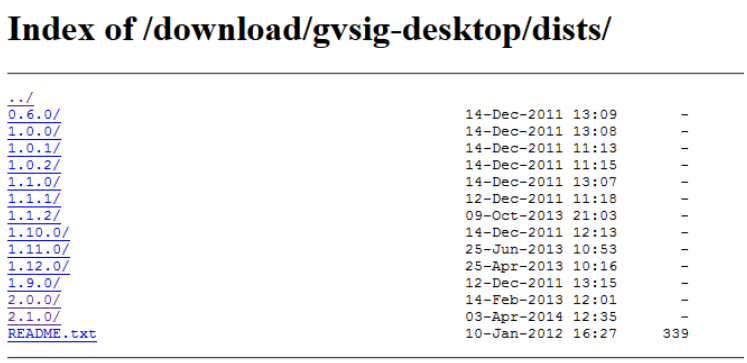
- *cearTablaArchivosBuild ( String version )*

  A partir de una versión, obtiene la última distribución y se extrae los datos para crear un nuevo objeto *TablaBuilds*. Antes de implementar el método deberemos inspeccionar la estructura de la fuente de datos para tener claro como abordar la implementación. La estructura es la siguiente:

  - Página web – URL base http://downloads.gvsig.org/download/gvsig-desktop/dists/
    - versión 1.11/
    - versión 1.12/
    - …
    - versión 2.1.0/
      - builds/
        - 2200/
        - 2201/
        - …
        - 2254/
      - misc/
      - web/
      - descargable 1
      - docs/
      - …

La página web base contiene el listado de enlaces (nodos de tipo <*a*>) de todas las versiones que existen actualmente. Dentro de cada versión se encuentran diferentes directorios y archivos, solo

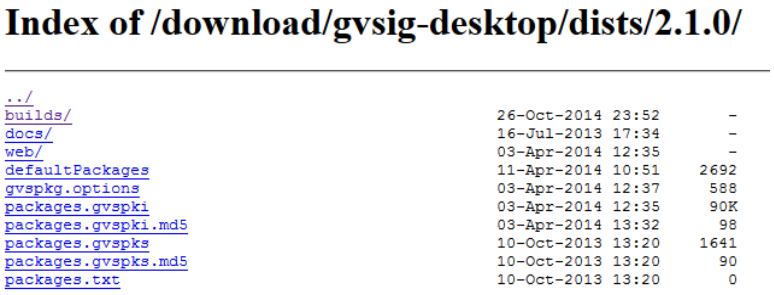
nos interesa el directorio *builds/* que es el que contiene todas las *builds* de la versión. Cada *build* se representa con un número y contiene archivos descargables y directorios. Las figuras 6.33, 6.34, 6.35 y 6.36 representan la estructura de la página web:



**Index of /download/gvsig-desktop/dists/**

| | | |
|---|---|---|
| ../ | | |
| 0.6.0/ | 14-Dec-2011 13:09 | - |
| 1.0.0/ | 14-Dec-2011 13:08 | - |
| 1.0.1/ | 14-Dec-2011 11:13 | - |
| 1.0.2/ | 14-Dec-2011 11:15 | - |
| 1.1.0/ | 14-Dec-2011 13:07 | - |
| 1.1.1/ | 12-Dec-2011 11:18 | - |
| 1.1.2/ | 09-Oct-2013 21:03 | - |
| 1.10.0/ | 14-Dec-2011 12:13 | - |
| 1.11.0/ | 25-Jun-2013 10:53 | - |
| 1.12.0/ | 25-Apr-2013 10:16 | - |
| 1.9.0/ | 12-Dec-2011 13:15 | - |
| 2.0.0/ | 14-Feb-2013 12:01 | - |
| 2.1.0/ | 03-Apr-2014 12:35 | - |
| README.txt | 10-Jan-2012 16:27 | 339 |

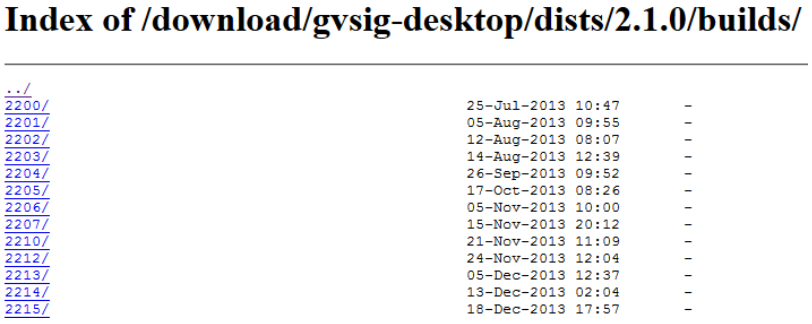
**Figura 6.33** Lista de versiones – URL Base



**Index of /download/gvsig-desktop/dists/2.1.0/**

| | | |
|---|---|---|
| ../ | | |
| builds/ | 26-Oct-2014 23:52 | - |
| docs/ | 16-Jul-2013 17:34 | - |
| web/ | 03-Apr-2014 12:35 | - |
| defaultPackages | 11-Apr-2014 10:51 | 2692 |
| gvspkg.options | 03-Apr-2014 12:37 | 588 |
| packages.gvspki | 03-Apr-2014 12:35 | 90K |
| packages.gvspki.md5 | 03-Apr-2014 13:32 | 98 |
| packages.gvspks | 10-Oct-2013 13:20 | 1641 |
| packages.gvspks.md5 | 10-Oct-2013 13:20 | 90 |
| packages.txt | 10-Oct-2013 13:20 | 0 |

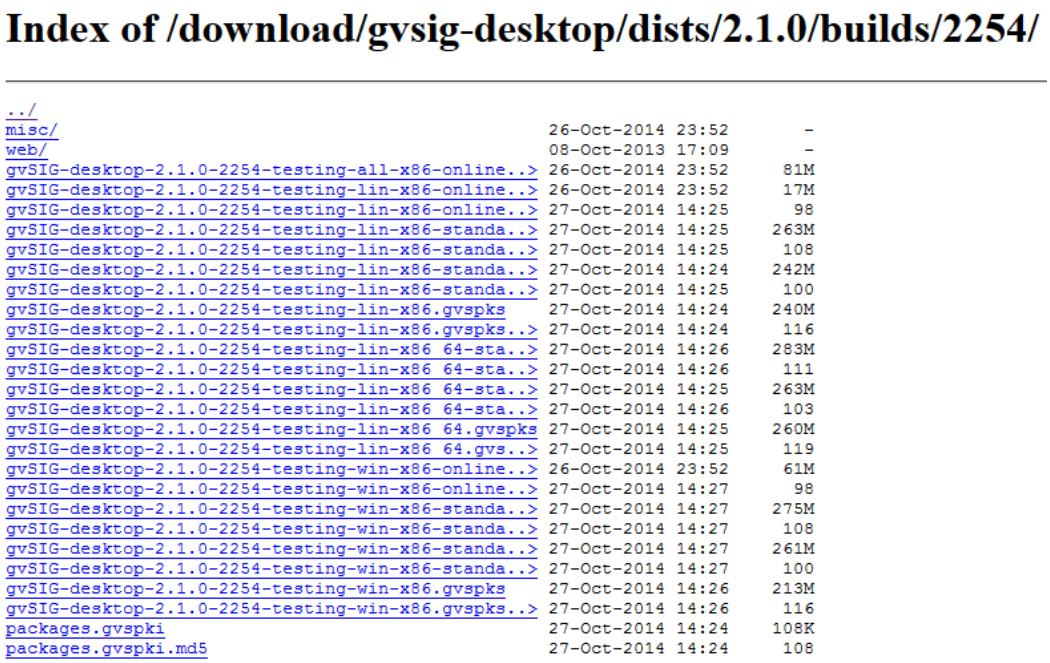
**Figura 6.34** Listado de directorios y archivos dentro de una versión



**Index of /download/gvsig-desktop/dists/2.1.0/builds/**

| | | |
|---|---|---|
| ../ | | |
| 2200/ | 25-Jul-2013 10:47 | - |
| 2201/ | 05-Aug-2013 09:55 | - |
| 2202/ | 12-Aug-2013 08:07 | - |
| 2203/ | 14-Aug-2013 12:39 | - |
| 2204/ | 26-Sep-2013 09:52 | - |
| 2205/ | 17-Oct-2013 08:26 | - |
| 2206/ | 05-Nov-2013 10:00 | - |
| 2207/ | 15-Nov-2013 20:12 | - |
| 2210/ | 21-Nov-2013 11:09 | - |
| 2212/ | 24-Nov-2013 12:04 | - |
| 2213/ | 05-Dec-2013 12:37 | - |
| 2214/ | 13-Dec-2013 02:04 | - |
| 2215/ | 18-Dec-2013 17:57 | - |

**Figura 6.35** Lista de *builds* dentro del directorio *builds/*



**Index of /download/gvsig-desktop/dists/2.1.0/builds/2254/**

| | | |
|---|---|---|
| ../ | | |
| misc/ | 26-Oct-2014 23:52 | - |
| web/ | 08-Oct-2013 17:09 | - |
| gvSIG-desktop-2.1.0-2254-testing-all-x86-online..> | 26-Oct-2014 23:52 | 81M |
| gvSIG-desktop-2.1.0-2254-testing-lin-x86-online..> | 26-Oct-2014 23:52 | 17M |
| gvSIG-desktop-2.1.0-2254-testing-lin-x86-online..> | 27-Oct-2014 14:25 | 98 |
| gvSIG-desktop-2.1.0-2254-testing-lin-x86-standa..> | 27-Oct-2014 14:25 | 263M |
| gvSIG-desktop-2.1.0-2254-testing-lin-x86-standa..> | 27-Oct-2014 14:25 | 108 |
| gvSIG-desktop-2.1.0-2254-testing-lin-x86-standa..> | 27-Oct-2014 14:24 | 242M |
| gvSIG-desktop-2.1.0-2254-testing-lin-x86-standa..> | 27-Oct-2014 14:25 | 100 |
| gvSIG-desktop-2.1.0-2254-testing-lin-x86.gvspks | 27-Oct-2014 14:24 | 240M |
| gvSIG-desktop-2.1.0-2254-testing-lin-x86.gvspks..> | 27-Oct-2014 14:24 | 116 |
| gvSIG-desktop-2.1.0-2254-testing-lin-x86_64-sta..> | 27-Oct-2014 14:26 | 283M |
| gvSIG-desktop-2.1.0-2254-testing-lin-x86_64-sta..> | 27-Oct-2014 14:26 | 111 |
| gvSIG-desktop-2.1.0-2254-testing-lin-x86_64-sta..> | 27-Oct-2014 14:25 | 263M |
| gvSIG-desktop-2.1.0-2254-testing-lin-x86_64-sta..> | 27-Oct-2014 14:26 | 103 |
| gvSIG-desktop-2.1.0-2254-testing-lin-x86_64.gvspks | 27-Oct-2014 14:25 | 260M |
| gvSIG-desktop-2.1.0-2254-testing-lin-x86_64.gvs..> | 27-Oct-2014 14:25 | 119 |
| gvSIG-desktop-2.1.0-2254-testing-win-x86-online..> | 26-Oct-2014 23:52 | 61M |
| gvSIG-desktop-2.1.0-2254-testing-win-x86-online..> | 27-Oct-2014 14:27 | 98 |
| gvSIG-desktop-2.1.0-2254-testing-win-x86-standa..> | 27-Oct-2014 14:27 | 275M |
| gvSIG-desktop-2.1.0-2254-testing-win-x86-standa..> | 27-Oct-2014 14:27 | 108 |
| gvSIG-desktop-2.1.0-2254-testing-win-x86-standa..> | 27-Oct-2014 14:27 | 261M |
| gvSIG-desktop-2.1.0-2254-testing-win-x86-standa..> | 27-Oct-2014 14:27 | 100 |
| gvSIG-desktop-2.1.0-2254-testing-win-x86.gvspks | 27-Oct-2014 14:26 | 213M |
| gvSIG-desktop-2.1.0-2254-testing-win-x86.gvspks..> | 27-Oct-2014 14:26 | 116 |
| packages.gvspki | 27-Oct-2014 14:24 | 108K |
| packages.gvspki.md5 | 27-Oct-2014 14:24 | 108 |

**Figura 6.36** Listado de descargables y directorios dentro de una *build*

Una vez sabemos que estructura tiene la fuente de datos procedemos a implementar la lógica del controlador. La figura 6.37 muestra la implementación del método *crearTablaArchivosBuilds*:

```
@Override
public TablaBuilds crearTablaArchivosBuild(String version) {
    Document doc = null;
    try {
        doc = parseHtmltoDoc(URL + version + "/builds/");
        String compilacion = ultimaBuild(doc);
        doc = parseHtmltoDoc(URL + version + "/builds/" + compilacion
                + "/");
        return new DefaultTablaBuilds(extraerDatos(doc));
    } catch (IOException e) {
        return null;
    }
}
```

**Figura 6.37** Implementación del método *crearTablaArchivosBuild*

El primer paso es extraer la última *build* de la versión, para ello obtendremos el objeto de tipo *Document*, propio de la librería JSoup y que representa una página HTML completa, de la página web de distribuciones de la versión. Para ello llamamos al método *parseHtmlToDoc(String URL)* , visto en la figura 6.31, que nos devuelve un objeto *Document* a partir de una URL.

Una vez obtenido el objeto *Document* solo nos queda obtener el nombre del último *build* de la versión. Para ello llamamos al método *ultimaBuild(Document doc).* La figura 6.38 muestra la implementación del método:

```
/**
 * A partir de un document DOM de una distribucion se obtiene el ultimo link
 * o ultima compilacion/directorio.
 *
 * @param doc
 *            Document DOM con las compilaciones o directorios
 * @return El nombre de la ultima build
 */
private String ultimaBuild(Document doc) {
    return getLastLink(doc).attr("href");
}
```

**Figura 6.38** Implementación del método *ultimaBuild*

El método ejecuta la sentencia *getLasLink(doc).attr("href")* devolviendo el contenido del atributo *href*. La figura 6.39 muestra la implementación del método *getLastLink(Document doc):*

```
    /**
     * Obtiene el ultimo link a partir de un Document DOM
     *
     * @param doc
     *            Document DOM de un pagina web
     * @return Ultimo link del documento como instancia de Element.
     */
    private Element getLastLink(Document doc) {
        return doc.select("a[href]").last();
    }
```

**Figura 6.39** Implementación del método *getLastLink.*

El método invoca al método *select(String cssQuery)* que permite obtener cualquier nodo del documento mediante una consulta *CSS*. En nuestro caso se ha obtenido todos los nodos *a* con atributo *href*, es decir todo el listado de *builds*, para luego obtener el último que se corresponde con la última *build* de la versión. Este método nos devuelve un objeto *Element* al cual podemos se debe obtener el valor del atributo *href* el cual se corresponde con la última *build*.

Una vez obtenido el nombre de la última distribución debemos extraer de nuevo un objeto *Document* de la página web y recorrer los nodos del documento y extraer las información que sea interesante. Gracias a que todas las páginas web siguen la misma estructura se ha implementado un método genérico capaz de extraer la información relevante. La figura 6.40 muestra la implementación del método *extraerDatos(Document doc):*

```
94  /**
95   * A partir de un Document DOM se obtiene los datos como nombre, link, ultima
96   * modificacion o tamaño.
97   *
98   * @param doc Document DOM con la informacion.
99   * @return ArrayList de instancias de ElementoTablaBuild con la informacion
100  */
101 private ArrayList<ElementoTablaBuild> extraerDatos(Document doc) {
102   ArrayList<ElementoTablaBuild> arrayBuild = null;
103   if (doc != null) {
104     Elements links = getAllLinks(doc);
105
106     String baseUrl = doc.baseUri();
107     arrayBuild = new ArrayList<ElementoTablaBuild>();
108     int i = 5;
109
110     for (; i < links.size(); i++) {
111       String nombre = links.get(i).attr("href");
112       String link = baseUrl + links.get(i).attr("href");
113
114       ElementoTablaBuild buildTemp = new DefaultElementoTablaBuild(nombre,
115           link);
116       arrayBuild.add(buildTemp);
117     }
118   }
119
120   return arrayBuild;
121 }
```

**Figura 6.40** Implementación del método *extraerDatos*

El método recibe un *Document* de la página de la que se quiere extraer la información. Los pasos seguidos han sido:

1. Se comprueba que el objeto *Document* no es nulo. Si es cierto se procede a extraer la información, en caso contrario se devuelve un *arrayList* nulo.
2. Se obtiene todos los links de la página web mediante el método *getAllLinks*. La figura 6.41 muestra la implementación:

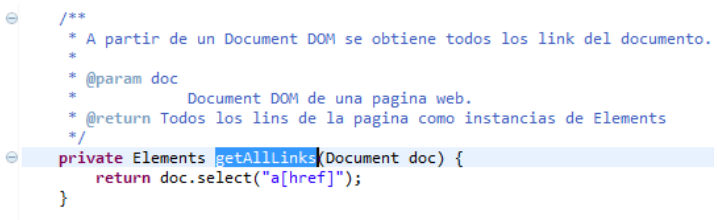



**Figura 6.41** Implementación del método *getAllLinks*

3. El método delega la funcionalidad en la operación *select* del documento, la cual devuelve una colección de nodos a partir de una *queryCSS*. En este caso devuelve todos los elementos de tipo <*a*> con atributo *href*.
4. El siguiente paso es obtener la dirección URL de la página para construir los enlaces de forma correcta. Esto es necesario debido a que los nodos de tipo *a* tiene como valor en el atributo *href* cadenas de tipo "*2220/*".
5. Inicializamos la colección de elementos de la tabla.
6. Inicializamos el índice para el recorrido de elementos. Esta parte es muy importante debido a que hay elementos que en la parte superior de la página web que no nos interesan. Por esta razón se inicializa el índice a "1" para evitar dichos elementos.
7. Inicializadas las variables, debemos recorrer las colecciones obtenidas. En cada iteración se obtiene el nombre y el link del elemento y se instancia un objeto temporal de tipo *ElementoTablaBuild* con dicha información para luego añadirlo a la colección.
8. Una vez recorridos todos los elementos devolvemos la colección con toda la información que nos interesa lista para representarla.

- *crearTablaDirectoriosBuilds ( String version )*

A partir de una versión se obtienen todas las *builds* con sus directorios y archivos. El resultado es un objeto de tipo *TablaBuilds* con toda la información. La figura 6.42 muestra la implementación:

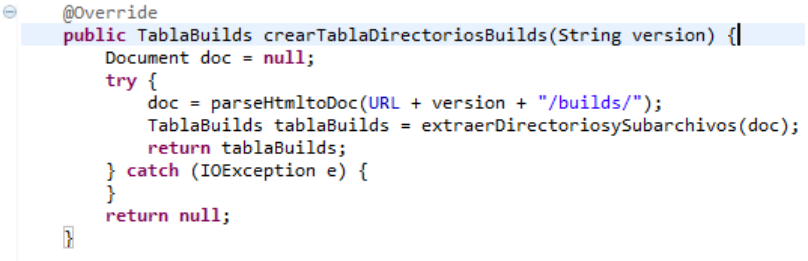



**Figura 6.42** Implementación del método *crearTablaDirectoriosBuilds*

Los pasos seguidos han sido:

1. Obtener el objeto *Document* de la página web donde se encuentran las *builds* de la versión especificada como parámetro.
2. Invocamos al método *extraerDirectoriosySubarchivos (Document doc)* encargado de extraer la información relevante de todos los directorios y subarchivos a partir del objeto obtenido anteriormente. La figura 6.43 muestra la implementación:

```
/**
 * A partir de un Document DOM se obtiene los datos de los directorios como
 * nombre, link, ultima modificacion, tamaÃ±o y archivos que contiene.
 *
 * @param doc
 *            Document DOM con la informacion.
 * @return ArrayList de instancias de ElementoTablaBuild con la informacion
 */
private TablaBuilds extraerDirectoriosySubarchivos(Document doc) {
    TablaBuilds tablaDirectorios = new DefaultTablaBuilds(extraerDatos(doc));

    for (ElementoTablaBuild directorio : tablaDirectorios.getListaBuilds()) {
        String url = directorio.getLink();
        try {
            Document docSubarchivos = parseHtmltoDoc(url);
            directorio.setSubArchivos(extraerDatos(docSubarchivos));
        } catch (IOException e) {
            return null;
        }

    }

    return tablaDirectorios;
}
```

**Figura 6.43** Implementación del método *extraerDirectoriosySubarchivos*

3. Los pasos seguidos han sido:
4. Obtener primero todos los directorios de las *builds.* Debido a que la información se distribuye igualmente en todas las páginas de la fuente de datos, se ha reutilizado el método *extraerDatos* mostrado en la figura 6.40.
5. Una vez obtenidos todos los directorios de las *builds*, se ha recorrido todos los directorios para extraer la información de cada uno de ellos. Se ha obtenido la dirección URL de cada uno, se ha obtenido el objeto *Document* y finalmente de nuevo se ha reutilizado el método *extraerDatos* para extraer a información interesante. El resultado de este método es asignado a la propiedad *subArchivos* que tiene cada elemento de una tabla.
6. El resultado es una colección de *builds* que a su vez dentro de cada elemento hay una colección con todos los directorios y archivos.

Implementado el controlador del *portlet*, queda la parte de la vista. La vista estará formada por dos partes. La primera parte se corresponde con la visualización de los datos en tablas y la segunda parte se corresponde con la interfaz de configuración del *portlet* que permite al usuario especificar la versión y *build* del producto que desea mostrar. A continuación se explicará la implementación de la parte de visualización y seguidamente la parte de configuración.

La visualización de la información en tablas en este *portlet* de Liferay básico está definida en el archivo JSP *view.jsp.* Con el objetivo de entender mejor la implementación, se va a dividir el archivo en diversas figuras con una explicación de segmento de código mostrado.

La figura 6.44 muestra la primera parte de la implementación:

```
<%
    /**
     * Copyright (c) 2000-2013 Liferay, Inc. All rights reserved.
     *
     * This library is free software; you can redistribute it and/or modify it under
     * the terms of the GNU Lesser General Public License as published by the Free
     * Software Foundation; either version 2.1 of the License, or (at your option)
     * any later version.
     *
     * This library is distributed in the hope that it will be useful, but WITHOUT
     * ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS
     * FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more
     * details.
     */
%>

<%@ taglib uri="http://java.sun.com/portlet_2_0" prefix="portlet"%>
<%@ taglib uri="http://liferay.com/tld/ui" prefix="liferay-ui"%>
<%@ page import="com.liferay.descargasbuild.api.ControladorDescargasBuilds,
    com.liferay.descargasbuild.api.ElementoTablaBuild,
    com.liferay.descargasbuild.api.TablaBuilds"
    %>

<%@ page import="com.liferay.descargasbuild.impl.DefaultControladorDescargasBuilds"%>

<%@ page import = "javax.portlet.PortletPreferences" %>


<portlet:defineObjects />
```

**Figura 6.44** Primera parte de la implementación de la vista.

La primera parte está compuesta por una cabecera generada automáticamente que informa al desarrollador o futuros desarrolladores que la librería es de código abierto, es redistribuible y se puede modificar bajos los términos de la licencia GNU. Seguidamente encontramos dos tipos de directivas explicadas a continuación:

- **Taglib**: directiva usada para incorporar librerías de *tags* al motor de JSP. Los *tags* nos proporcionan funcionalidades encapsuladas. Está compuesto por dos atributos, *uri* y *prefix. Uri* indica el archivo que define los *tags* de toda la librería y el atributo *prefix* es usado para especificar, dentro de nuestro JSP, el prefijo para invocar los *tags*. En este caso al especificar como *prefix* "portlet"para invocar un *tag* del archivo http://java.sun.com/portlet_2_0 se debe realizar del siguiente modo: <*portlet:tagName*>.
- **Page:** directiva usada para incorporar clases librearías Java al motor de JSP. En Java se corresponderían a los *imports* de las clases.

Retomando la implementación de la vista, por una parte debemos añadir mediante las directivas *taglib* las librerías **portlet_2_0** y **liferay-ui.** Por otra parte debemos añadir las clases Java de nuestro modelo y la clase *PortletPreferences* mediante la directiva *page*.

La librería **liferay-ui** nos proporciona una amplia funcionalidad, desde insertar elementos tales como calendarios o buscadores hasta un sistema de traducciones mediante clave que será el que usaremos en nuestro caso [jsptglu].

La librería **portlet 2.0** mediante la sentencia <*porlet:defineObjects*> nos permite definir una serie de objetos implícitos dentro del archivo JSP que ayudan al desarrollador a acceder a información necesaria del *portlet* [jsptbp2]. En nuestro nos interesa la variable *renderRequest.* Antes de explicar la utilidad de esta variable, es necesario explicar y entender el funcionamiento del *portlet* y sus dos fases: ***render phase*** y ***action phase***.

Nuestro *portlet* necesita dos fases de ejecución para poder funcionar. Supongamos por ejemplo, que tenemos un *portlet* que ofrece al usuario una selección de artículos con opción a compra. El usuario elige el artículo en el que está interesado y realiza una compra. El portal debe recargar el contenido por defecto del *portlet*. De nuevo el usuario hace *click* en un enlace del *portlet*. Esta acción crea un ciclo de envíos y respuestas HTTP que provoca que el contenido del *portlet* cambie. Pero todos los parámetros del *portlet* han sido preservados, incluso lo datos de la tarjeta de crédito, realizando de nuevo la compra anteriormente hecha. Esto ocurre debido a que el portal no es capaz de diferenciar si el *porltet* solamente debe "refrescarse" o debe realizar una acción. Debido a esto los *portlet* tiene dos fases de ejecución, una para el refresco de información y otra para realizar acciones:

- *Action phase:* solamente un portlet puede invocar una acción en un mismo instante de tiempo. En esta fase el *portlet* puede cambiar de estado así como sus preferencias.
- *Render phase:* es invocada en todos los *portlets* después de realizar una *action phase.* Según la especificación de un *portlet*, no podemos asumir nada sobre el orden de invocación En Liferay podemos añadir un atributo al archivo *liferay-portlet.xml* llamado -*render-weight* que permite especificar al desarrollador el orden de ejecución de la fase *render*. *Portlet* con mayor *render-weight* ejecutarán la fase antes.

Volviendo a nuestro *portlet*, el objeto *renderRequest* contiene información acerca la petición enviada al *portlet* para realizar la fase *render*. Este objeto nos permite obtener información acerca de las preferencias del *portlet.* Nos es muy útil debido que la información sobre que versión debemos mostrar se encontrará alojada dentro de las preferencias. Esto es posible gracias a la fase *action* que realizará nuestro portlet cuando implementemos las parte de configuración que recordemos que permitirá al usuario establecer la versión, pero esto se explicará un poco más adelante. [ldgutppe14]

La figura 6.45 muestra la segunda parte de la implementación de la vista:

```
<%
    ControladorDescargasBuilds controlador = new DefaultControladorDescargasBuilds();
    PortletPreferences prefs = renderRequest.getPreferences();

    String distribucion = (String) prefs.getValue("distribucion", "");
    String esFinal = (String) prefs.getValue("esFinal", "");
    String build = "";

    if (!distribucion.equals("") && !esFinal.equals("")) {
        boolean error = controlador.comprobarUrls(distribucion);
        if (!error) {
            TablaBuilds tablaArchivosBuilds = controlador
                    .crearTablaArchivosBuild(distribucion);
            TablaBuilds tablaDirectoriosBuilds = controlador
                    .crearTablaDirectoriosBuilds(distribucion);
            build = controlador.ultimaBuild(distribucion);
%>
```

**Figura 6.45** Segunda parte de la implementación de la vista

Observamos el primer fragmento de código Java dentro de nuestro archivo JSP. Estos han sido los pasos seguidos:

1. Creación de un objeto controlador para que nos proporcione la funcionalidad necesaria
2. Obtención de las preferencias del *portlet* mediante la sentencia *renderRequest.getPreferences().*
3. Mediante la operación *prefs.getValue(String key, String def)* obtenemos la distribución o versión y si dicha versión es final o no. El primer parámetro corresponde con la *key* del valor y

el segundo es el valor que devuelve el método en caso de que la *key* especificada no se encuentre dentro de las preferencias.

4. Comprobamos que los valores no estén vacios, en el caso de que las cadenas estuvieran vacías se mostrará un mensaje de información (dicha implementación se encuentra en la última parte), en caso contrario comprobamos mediante el método *comprobarUrls* que la versión especificada es correcta. En caso de ser correcta obtenemos las dos tablas con todos los datos mediante el controlador y los métodos *crearTablaArchivosBuilds y crearTablaDirectoriosBuilds*. En caso de ser incorrecta se mostrará un mensaje de error. (Dicha implementación también se encuentra en la última parte)
5. Además, obtenemos la última *build* de la versión para indicar que versión se está mostrando en la primera tabla.

La figura 6.46 muestra la tercera parte de la implementación:

```
<h4>
  <liferay-ui:message key="distribucion" />
  :<%=distribucion%>
</h4>
<h4>
  <liferay-ui:message key="final" />
  :<%=esFinal%>
</h4>
<div id="containerBuildTables">
  <div class="tableHeader toggler-header toggler-header-expanded">
    <div class="icon-download-alt"></div>
    <a> <liferay-ui:message key="descargar" /> (<liferay-ui:message
        key="build" /> <%=build%>):
    </a>
  </div>
  <div class="tableContent toggler-content toggler-content-expanded">
    <table class="table table-striped">
      <thead>
        <tr>
          <th><liferay-ui:message key="nombre" /></th>
        </tr>
      </thead>
      <tbody>
        <%
          for (ElementoTablaBuild item : tablaArchivosBuilds.getListaBuilds()) {
        %>
        <tr>
          <td><a href="<%=item.getLink()%>"><%=item.getNombre()%></a></td>
        </tr>
        <%
      </tbody>
    </table>
  </div>
  <div class="tableHeader toggler-header toggler-header-collapsed">
    <div class="icon-download-alt"></div>
    <a> <liferay-ui:message key="builds" />
    </a>
  </div>
```

**Figura 6.46** Tercera parte de la implementación de la vista

En la figura observamos el código HTML que generará nuestro portlet. La primer parte está compuesta por dos títulos de tipo *h4* indicando la distribución y si es final o no. A continuación comienza la construcción de la primera tabla. Recordemos que el portlet debe generar dos tablas, la primera se corresponde con la última *build* de la versión especificada y la segunda tabla contiene todos los archivos y directorios de todas las *builds*. Para generar las tablas se ha usado las clases CSS de **Bootstrap** para seguir con la misma línea de estilos que el portal. Debido a esto englobaremos en un *div* la primera tabla y le asignaremos un id, en este caso es: *containerBuildTables*. Este *div* debe englobar a dos elementos *div*, el primero contendrá un

elemento *a* enlazado al segundo que permitirá al usuario expandir o contraer la tabla. El segundo elemento se corresponde con el cuerpo de la tabla.

Las tablas siguen la estructura HTML. Se componen de un elemento global *table* que engloba a dos: *thead* y *tbody*.

*Thead* define la cabecera de la tabla y el número de columnas mediante las etiquetas *th* englobadas por un bloque *tr*. En nuestro caso la tabla tendrá tres columnas: nombre, última modificación y tamaño por lo tanto se debe de crear un bloque *tr* con tres elementos *th* dentro cada uno con el nombre de la columna.

*Tbody* define el cuerpo de la tabla. Cada fila se corresponde con cada bloque de etiquetas *tr* y cada valor dentro de una fila se corresponde con la etiqueta *td*. Por lo tanto si necesitamos que la información se represente en filas se deberá recorrer la colección con un *bucle for* de Java, que ha sido obtenido anteriormente mediante el controlador, e ir creando bloques *tr* con cada elemento de la colección. Además, cada atributo de un elemento estará englobado por un bloque *td*. La figura 6.47 muestra el resultado de la creación de la primera tabla:


**Figura 6.47** Visualización de la primera tabla del *porltet descargasBuilds*

Una vez creada la primera tabla, se debe crear el listado con todas las *build* de la versión. Cada punto de la lista debe corresponderse con una tabla con los archivos del nodo. La lista se mostrará por defecto comprimida debido a que el número de *builds* puede ser bastante grande haciendo que la página sea extremadamente larga. La figura 6.48 muestra la cuarta parte de la implementación del archivo view.jsp.

```
87  <div id="contentTablesSubfiles"
88    class="tableContent toggler-content toggler-content-collapsed">
89    <ul>
90      <%
91        for (ElementoTablaBuild item : tablaDirectoriosBuilds
92                .getListaBuilds()) {
93      %>
94      <li><a class="headerDir"><%=item.getNombre()%></a>
95        <div class="contentDir toggler-content toggler-content-collapsed">
96          <table class="table table-striped">
97            <tbody>
98              <%
99                for (ElementoTablaBuild subitem : item.getSubArchivos()) {
100             %>
101             <tr>
102               <td><a href="<%=subitem.getLink()%>"> <%=subitem.getNombre()%></a></td>
103             </tr>
104             <%
105               }
106             %>
107           </tbody>
108         </table>
109       </div></li>
110     <%
111       }
112     %>
113   </ul>
114 </div>
```

**Figura 6.48** Cuarta parte de la implementación de la vista

Se observa de nuevo código HTML. La estructura es muy semblante a la primera tabla explicada anteriormente. Creamos un bloque *div* que servirá de contendor del listado con el id *contentTableSubFiles.* A continuación debemos de crear un listado con todas las *build* para ello haremos uso de las etiqueta *ul* y *li* para crear una lista no ordenada. Para ello debemos anidar dos recorridos *for* de Java.

Por una parte el primer recorrido se realizará sobre la colección de *builds.* En cada iteración se deberá crear un nodo del listado con la etiqueta *li*. Dentro de cada nodo del listado se ha creado una tabla siguiendo la estructura propia de HTML. Por otra parte, el segundo recorrido se corresponde con los archivos y directorios de cada build. Estos archivos y directorios se corresponden con cada fila de la tabla del nodo así que con cada iteración crearemos un bloque *tr* y dentro de estos bloques, crearemos bloques *td* con los atributos de los elementos de forma parecida a la primera tabla explicada anteriormente. La figura 6.49 el resultado.

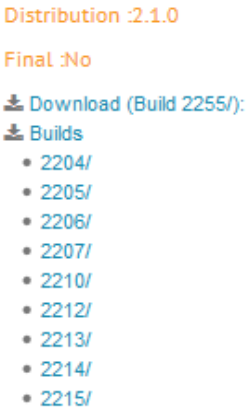



**Figura 6.49** Visualización contraída del listado de tablas del *portlet descargasBuilds.*

La figura 6.50 muestra el resultado al hacer *click* sobre algún elemento de la lista:


**Figura 6.50** Visualización expandida del listado de de tablas del *portlet descargasBuilds.*

Finalmente solo queda por implementar dos casos que también pueden darse, cuando al acceder a las preferencias del *portlet* para obtener la versión y si es final o no obtenemos como resultado una cadena vacía y cuando el usuario introduce una versión que no existe.

Si al intentar obtener los valores de las preferencias obtenemos una cadena vacía significa que o bien que el usuario acaba de añadir el *portlet* a la página o se ha guardado una configuración del *portlet* vacía. En dicho caso se deberá mostrar un mensaje informativo al usuario avisándole de que es necesario configurar la versión. La figura 6.51 muestra dicha implementación:

```
28 <%
29   ControladorDescargasBuilds controlador = new DefaultControladorDescargasBuilds();
30   PortletPreferences prefs = renderRequest.getPreferences();
31
32   String distribucion = (String) prefs.getValue("distribucion", "");
33   String esFinal = (String) prefs.getValue("esFinal", "");
34   String build = "";
35
36   if (!distribucion.equals("") && !esFinal.equals("")) {

        Implementación de la vista

123    }
124    else {
125  %>
126  <div class="alert alert-info">
127    <liferay-ui:message key="infoMsg" />
128  </div>
129  <%
130    }
131  %>
132 </div>
```

**Figura 6.51** Cuarta parte de la implementación de la vista

Observamos que se hace uso de la librería de *tags* que proporciona Liferay. Dicha librería proporciona soporte para mostrar mensajes en el idioma del portal establecido por el usuario. Dicho soporte se obtiene mediante la invocación de la operación *message*, que tiene como parámetro la *key* de traducción. Estas claves o *keys* se definen mediante archivos *properties* alojados dentro del directorio docroot/WEB-INF/src. El nombre de estos archivos debe seguir la siguiente estructura: Language.properties para el idioma por defecto del portal y Language_xx.properties para el resto de idiomas, substituyendo xx por el código ISO del idioma. Por ejemplo si deseamos añadir la clave de traducción *infoMsg* al idioma italiano, primero deberemos crear el archivo Language_it.properties y añadir la siguiente línea: "*infoMsg = Hey! Fare clic su configuracione per selezionare la distribuzione!* ". Con esto conseguimos que si un usuario accede a nuestro portal y cambia el idioma por defecto al italiano, todos los mensajes con la clave *infoMsg* muestren la cadena anterior. Volviendo a la implementación de la vista, una vez conseguido que el mensaje se ajuste al idioma del usuario le añadimos las clases oportunas de Bootstrap para darle un estilo de mensaje informativo. La figura 6.52 muestra el resultado cuando añadimos el *portlet* a la página:


**Figura 6.52** Mensaje de información del *portlet descargasBuilds*

Finalmente solo queda el otro caso nombrado anteriormente. Si el usuario por error introduce una versión que no existe el portlet debe de mostrar un mensaje de error advirtiendo que la versión especificada no existe. La figura 6.53 muestra la implementación:

```
28  <%
29    ControladorDescargasBuilds controlador = new DefaultControladorDescargasBuilds();
30    PortletPreferences prefs = renderRequest.getPreferences();
31
32    String distribucion = (String) prefs.getValue("distribucion", "");
33    String esFinal = (String) prefs.getValue("esFinal", "");
34    String build = "";
35
36    if (!distribucion.equals("") && !esFinal.equals("")) {
37      boolean error = controlador.comprobarUrls(distribucion);
38      if (!error) {
                Impelementación de
                la vista
116     }
117       else {
118   %><div class="alert alert-error">
119       <liferay-ui:message key="errorMsg" />
120   </div>
121   <%
122     }
```

**Figura 6.53** Quinta parte de la implementación de la vista

Observamos que si al ejecutar el método *comprobarUrls* que nos ofrece el controlador obtenemos *true* debemos mostrar el mensaje de error. Se invoca al *tag message* con la *key errorMsg* y se encapsula en un *div* con las clases Bootstrap oportunas para conseguir un estilo de error. La figura 6.54 muestra el resultado:


**Figura 6.54** Mensaje de error del *portlet descargasBuilds*

Se ha implementado la vista del *portlet*. A continuación se explicará como hacer nuestro *portlet* configurable mediante la opción de configuración que ofrece Liferay. Es interesante usar esta opción debido a que la configuración de los *portlets* solo debe estar accesible a usuarios con los permisos suficientes y usando esta opción delegamos esta funcionalidad en el sistema. La configuración debe de ser un formulario con dos campos para indicar la distribución y si es final o no, además de un botón para guardar los valores. Si los valores se han guardado correctamente se debe de mostrar un mensaje de confirmación.

Al igual que hemos separado la lógica de la vista en el *portlet* también deberemos hacerlo cuando implementemos una configuración. Por un lado la vista serán los campos de opciones visibles por el usuario y la lógica se corresponderá con la acción de guardar la configuración. A continuación se explicará la implementación de la vista y posteriormente la implementación del guardado.

La vista de la configuración también la genera un archivo JSP llamado config.jsp. Este archivo se ha situado en docroot/html/configuration dentro de la estructura del *portlet*, pero se puede alojar en el sitio más apropiado desde el punto de vista del desarrollador. La figura 6.55 muestra la primera parte de la implementación del archivo config.jsp:

```
<%@ taglib uri="http://liferay.com/tld/portlet" prefix="liferay-portlet" %>
<%@ taglib uri="http://java.sun.com/portlet_2_0" prefix="portlet"%>
<%@ taglib uri="http://liferay.com/tld/aui" prefix="aui"%>
<%@ taglib uri="http://liferay.com/tld/ui" prefix="liferay-ui"%>

<%@ page import="javax.portlet.PortletPreferences, javax.portlet.RenderRequest"%>

<portlet:defineObjects />
```

**Figura 6.55** Primera parte de la implementación de la configuración del portlet *descargasBuilds*

Al igual que la vista del portlet, el archivo *config.jsp* también debe de importar las librerías de *tags* necesarias para facilitar la implementación. En este caso las librerías importadas son **portlet, portlet_2_0, aui** y **ui.**

La librería **portlet** nos proporciona funcionalidad sobre las dos fases de ejecución propias del portlet: *action phase* y *render phase* explicadas anteriormente. En nuestro caso, no permitirá indicar al *portlet* que clase java debe ejecutar cuando un usuario guarde las preferencias. La librería **aui** nos proporciona *tags* para la construcción de componentes gráficos de forma fácil tales como formularios, botones... En nuestro caso se usará para crear el formulario de preferencias. Las librerías **portlet_2_0** y **ui** ya se han explicado con anterioridad.

La figura 6.56 muestra la segunda parte de la implementación:

```
<%
   PortletPreferences prefs = renderRequest.getPreferences();
   String distribucion = prefs.getValue("distribucion", "");
   String esFinal = prefs.getValue("esFinal", "");
   String actualizado = prefs.getValue("actualizado", "");

   if(actualizado.equals("si")){
     %>
     <div class="alert alert-success">
     <liferay-ui:message key="successMsg"/>
     </div>
     <%
     actualizado = "no";
     prefs.setValue("actualizado", actualizado);
     prefs.store();
   }
 %>
```

**Figura 6.56** Segunda parte de la implementación de la configuración del *portlet*

La segunda parte se corresponde con el código Java que comprueba que las preferencias han sido guardadas correctamente mostrando un mensaje de confirmación. Los pasos seguidos han sido los siguientes:

1. Se obtiene el objeto que contiene las preferencias del *portlet* mediante el objeto *renderRequest.*
2. Obtenemos los valores de las claves *distribucion*, *esFinal* y *actualizado*.
3. Si el valor de la variable *actualizado* es igual que "si" significa que el usuario ha pulsado el botón de guardar. Se debe mostrar el mensaje de confirmación informando al usuario de que se han guardado correctamente las propiedades. Al mostrar el mensaje debemos actualizar la variable *actualizado* en las preferencias para que la próxima *render phase* no se muestre el mensaje. Para ello asignamos el valor "no", ejecutamos la sentencia *prefs.setValue* y guardamos los cambios con *prefs.store().*
4. En caso de que la variable actualizado sea distinto de no hay que mostrar el mensaje de confirmación.

Una vez implementada la lógica para obtener los valores y mostrar el mensaje de confirmación, solo queda construir el formulario y el botón de guardado. La figura 6.57 muestra la tercera parte de la implementación:

```
<liferay-portlet:actionURL portletConfiguration="true" var="configurationURL" />

<aui:form action="<%=configurationURL%>" method="post">
  <liferay-ui:message key="distribucion" />
  <aui:input label="" name="distribucion" type="text" value="<%=distribucion%>" />
  <liferay-ui:message key="esFinal" />
  <aui:input label="" name="esFinal" type="text" value="<%=esFinal%>" />
  <aui:button type="submit" />
</aui:form>
```

**Figura 6.57** Tercera parte de la implementación la configuración del *portlet*

Para la construcción del formulario se ha usado el *tag form* que nos permite crear formularios de forma rápida y sencilla. El *tag form* recibe como parámetros *action* y *method. Action* se corresponde con la URL a la que se envía los datos y *method* se corresponde al tipo de envío. La URL a la que debemos enviar la información la obtenemos a partir de otro tag llamado *actionURL*. Este *tag* se encarga de generar la URL según los parámetros que recibe para ejecutar la *action phase* del *porltet*. Recibe como parámetro *portletConfiguration* y *var. PortletConfiguration* indica que la URL que debe de generar el *tag* es para configurar el *portlet* mientras que el parámetro *var* indica el nombre de la variable que contiene la URL.

Una vez obtenida la URL de configuración, procedemos a construir el formulario. Recordemos que el formulario debe tener dos campos de texto y un botón para enviar la petición a la URL anteriormente obtenida. Para ello se ha usado los *tags input* y *button*. El tag input recibe como parámetros el nombre, el tipo de campo y el valor por defecto mientras que el tag button recibe como parámetro el tipo de botón. En nuestro caso los campos serán de tipo texto y cada uno mostrará el valor de la variable que representa consiguiendo así que cuando se acceda a la configuración los campos muestren la configuración actual. El botón en nuestro caso será de tipo *submit* debido a que debe enviar los datos del formulario.

Una vez implementada la vista debemos de implementar la acción que se ejecuta al pulsar el botón. Para ello crearemos una clase dentro del directorio *docroot/WEB-INF/src*. Dicha clase extenderá de *DefaultConfigurationAction* para simplificar nuestra tarea ya que dicha clase proporciona una implementación por defecto. Nuestra clase deberá sobrescribir el método *processAction* que es el método que se ejecuta en la *action phase*. La figura 6.58 muestra la implementación:

```java
/* Copyright 2014 DiSiD Technologies S.L.L. All rights reserved.*/
package com.liferay.descargasbuild.configuracion;

import javax.portlet.ActionRequest;

public class ConfiguracionPortlet extends DefaultConfigurationAction {

  @Override
  public void processAction(PortletConfig portletConfig,
                            ActionRequest actionRequest,
                            ActionResponse actionResponse) throws Exception {

    super.processAction(portletConfig, actionRequest, actionResponse);

    PortletPreferences prefs = actionRequest.getPreferences();

    String distribucion = (String) actionRequest.getParameter("distribucion");
    String esFinal = (String) actionRequest.getParameter("esFinal");


    if (distribucion != null && esFinal != null) {
      String actualizado = "si";
      prefs.setValue("distribucion", distribucion);
      prefs.setValue("esFinal", esFinal);
      prefs.setValue("actualizado", actualizado);
      prefs.store();
    }

  }

}
```

**Figura 6.58** Implementación de la lógica de la configuración del *portlet*

Se han seguido los siguientes pasos:

1. Ejecutamos la implementación por defecto alojada en la clase padre.
2. Obtenemos las preferencias del portlet.
3. Obtenemos los valores introducidos en el formulario que se encuentran dentro del objeto *actionRequest* mediante el método *getParameter(String key)*.
4. Si los valores obtenidos son distintos de *null* los guardamos en el objeto preferencias del *portlet* y asignamos a la variable *actualizado* el valor si para que el JSP de la página de configuración muestre el mensaje de confirmación. Con esto conseguimos que los valores estén disponibles en la *render phase*. El objeto *PortletPreferences* actúa como puente entre las dos fases permitiendo el envío de información.
5. Guardamos la información con la sentencia *prefs.store()*.

Las figura 6.59 y 6.60 muestran el resultado:

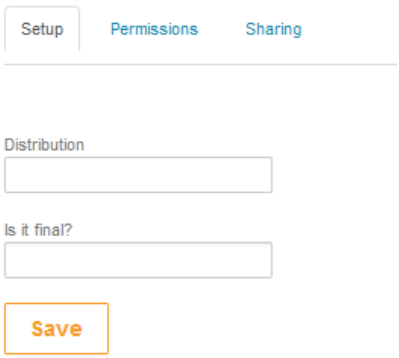



**Figura 6.59** Primera parte del resultado implementación de la configuración del *portlet*

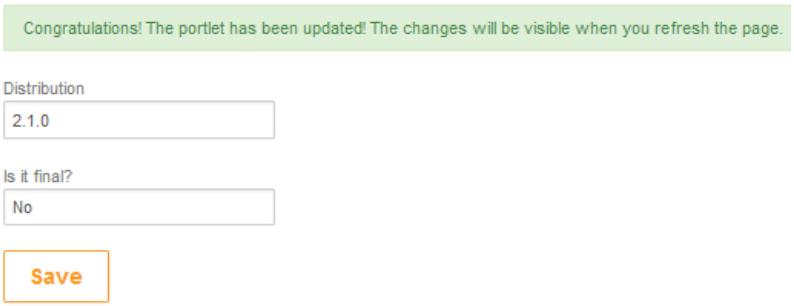



**Figura 6.60** Segunda parte del resultado implementación de la configuración del *portlet*



# Capítulo 7. Fase de transición

En este capítulo pretende introducir los detalles de transición del proyecto. Los objetivos de esta fase es especificar puesta en marcha y mantenimiento del sistema. Se considera que el producto está listo para ser entregado al cliente y pueda probarlo sin ningún problema.

En la sección 7.1 se va explicar la instalación del servidor de Liferay, Apache y el servidor postgresql, seguidamente en la sección 7.2, se explicará la implementación de un *plugin* de tipo *hook* para añadir claves de traducción al servidor.

## 7.1 Instalación del servidor Liferay de producción

Liferay permite es muy flexible en cuanto a términos de instalación se refiere. Se puede instalar bajo varios servidores de aplicaciones, adaptándose a una amplia variedad de preferencias. En nuestro caso la instalación se realizará junto con el servidor de aplicaciones Tomcat, como ya se ha comentado anteriormente.

El primer paso es elegir que versión de Liferay deseamos instalar. Existen varias versiones que van desde la más antigua hasta la más actual. Las versiones disponibles son: 5.1, 5.2, 6.0, 6.1 y 6.2. En nuestro caso instalaremos la versión 6.2 ya que ofrece más funcionalidades respecto a las anteriores versiones además de una interfaz de usuario renovada y más amigable frente a usuarios con pocos conocimientos sobre gestiones de CMS. Dentro de la versión 6.2 existen dos distribuciones: ***entrerprise*** y ***community edition*****.** Se recuerda que al realizar el análisis del problema el coste del portal era una de los factores claves dentro de la propuesta de la solución, por lo tanto se eligió la versión **community edition** totalmente gratuita. A grandes rasgos la diferencia entre las dos versiones reside en que la edición e*nterprise* proporciona mantenimiento y soporte técnico por parte de Liferay a los usuarios de sistema. Además, existe un equipo de Liferay encargado de mantener y corregir errores que reportan los usuarios. En cambio la versión *community edition* carece del soporte de Liferay y del mantenimiento comentado anteriormente [lugel14].

Antes de explicar la instalación del servidor de producción, se debe tener en cuenta que debido a que el autor de este proyecto carece de un dominio público la instalación del servidor apache, Liferay y base de datos se realizará sobre una misma máquina en local. Por otra parte tampoco se explicará la instalación y configuración del NAS. Aclarar que en el servidor real, la arquitectura del sistema sigue la arquitectura explicada en este proyecto.

La instalación se realizará sobre un entorno Linux Ubtuntu 14.04 con permisos *root*. Siguiendo la estructura de la figura 5.4 el primer paso es instalar servidor web Apache en modo *proxy reverse.* Para ello ejecutamos el siguiente comando en la consola del sistema:

```
apt-get install apache2 libapache2-mod-proxy-html
```

Con este comando instalamos apache en nuestra máquina junto con el módulo *proxy* necesario para nuestra arquitectura. Una vez que el proceso de instalación se ha llevado a cabo, el servidor arranca automáticamente.

El siguiente paso es activar el módulo *porxy_http* con el comando:

```
a2enmod proxy_http
```

Activado el módulo reiniciamos el servidor con:

```
service apache2 restart
```

Una vez reiniciado el servidor debe añadir la configuración necesaria para redirigir las peticiones http a la dirección http://localhost:8080, que es la dirección por defecto de la instalación de Liferay. Para ello accedemos al directorio *etc/apache2/sites-available* y creamos un fichero llamado **gvSIG-portal.conf** y añadimos el siguiente texto:

```
<VirtualHost *:80>

ProxyPreserveHost On
ProxyPass / http://localhost:8080/
ProxyPassReverse / http://localhost:8080 /
ServerName gvsig.com

</VirtualHost>
```

Una vez añadido el fichero de configuración debemos de desactivar el sitio web por defecto y activar el sitio creado. Para ello se ejecuta las sentencias:

```
a2dissite 000-default.conf

a2ensite gvSIG-portal.conf

service apache2 reload
```

Configurado el servidor Apache se procede a la instalación de Liferay. Para instalar Liferay 6.2 *Community edition* primero debemos de descargarla. Desde la pagina web oficial de Liferay en el apartado descargas[6] se debe de descargar el *pack* Liferay + Tomcat 6.2. Una vez obtenido el zip descomprimir el contenido. Una vez descomprimido, se debe cambiar los permisos sobre algunos archivos para poder arrancar el servidor. Los archivos son: startup.sh, catalina.sh y shutdown.sh. Se debe de acceder a la ruta: *~/tomcat-7.0.42/bin* y ejecutar los siguientes comandos:

```
chmod +x catalina.sh

chmod +x startup.sh

chmod +x shutdown.sh
```

[6] Link: http://www.liferay.com/es/downloads/liferay-portal/available-releases

Con estos comandos otorgamos permisos de ejecución sobre estos tres archivos. Hecho esto, solo queda ejecutar el archivo startup.sh y esperar a que el servidor arranque. Tras realizar el primer arranque del servidor, se abre una pestaña del navegador con la dirección. http://localhost:8080 La configuración por defecto establece esa dirección y puerto.

Se comprueba que el servidor Apache redirige bien las peticiones si accedemos a la dirección: localhost:80. Si todo esta correcto, la página debe ser muy parecida a la figura 7.1:

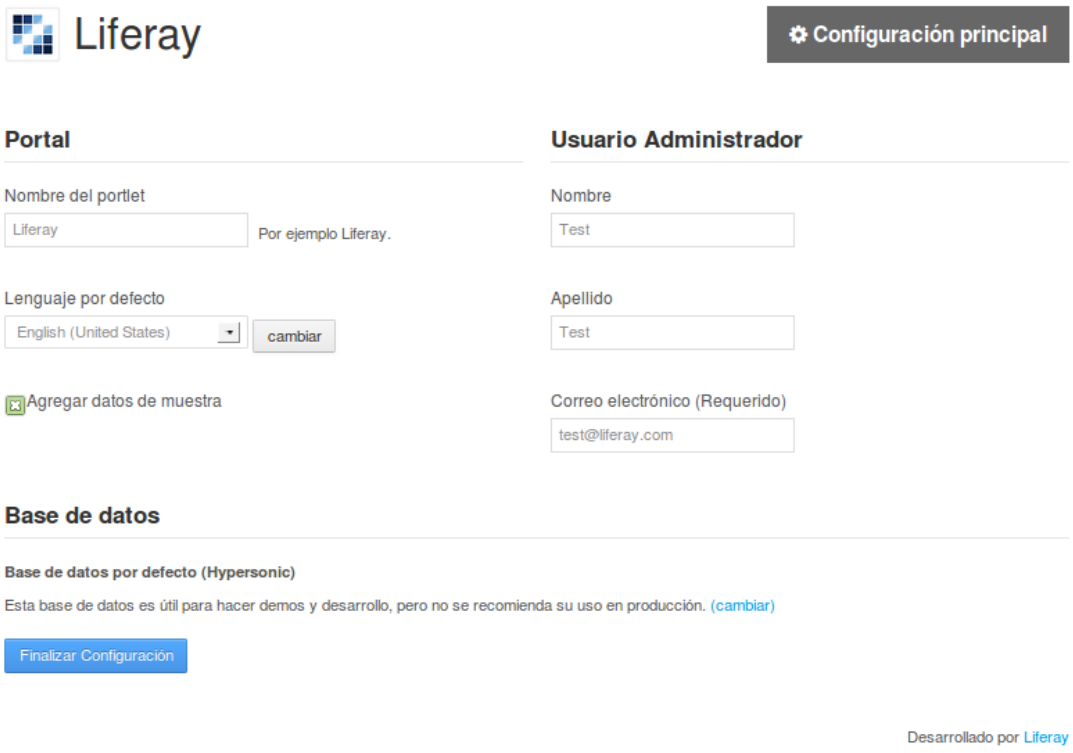



**Figura 7.1** Página de inicio del servidor de Liferay

La figura muestra la configuración inicial del servidor Liferay. Debemos escribir el nombre del portal, lenguaje por defecto, nombre y apellidos del usuario administrador y correo electrónico. Se debe de desactivar la opción de añadir datos de muestra. Y finalmente hacer *click* en **Finalizar configuración.** Aceptamos las condiciones de uso del sistema y asignamos la contraseña del usuario administrador. Elegimos la pregunta secreta y escribimos la respuesta. La figura 7.2 muestra la finalización del proceso y la página web por defecto al instalar Liferay []:

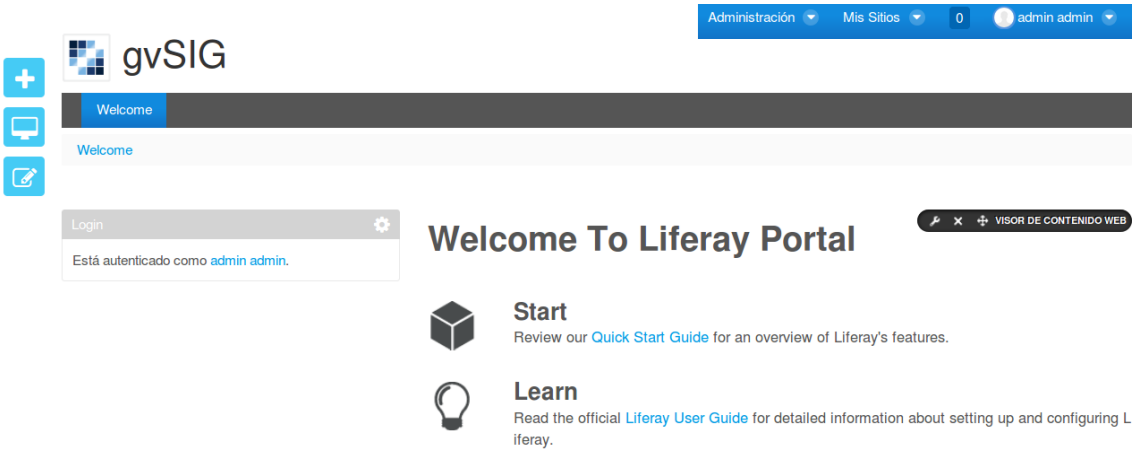



**Figura 7.2** Finalización del proceso de configuración del servidor Liferay

Antes de configurar y desplegar todo el contenido sobre el servidor vamos a crear y configurar la base de datos de Liferay. Si el administrador del sistema no configura una base de datos, Liferay por defecto usa el gestor de base de datos HyperSonic que guarda el contenido en memoria. Esta base de datos es útil para entorno de pruebas o desarrollo pero no para producción.

El primer paso es instalar postgresql 9.3. Para ello ejecutamos el siguiente comando:

```
apt-get install postgresql-9.3
```

Tras finalizar la operación estará disponible el usuario de sistema **postgres.** Este usuario tiene los permisos necesarios para gestionar el servidor de postgresql. Es altamente recomendable cambiar la contraseña por defecto del usuario. Para ello ejecutamos los siguientes comandos:

```
Sudo su

*contraseña de root*

passwd postgres

*nueva contraseña*

*repetimos nueva contraseña*
```

Una vez cambiada la contraseña del usuario cambiamos de usuario para ello ejecutamos:

```
sudo su postgres
```

Cambiado de usuario ejecutamos la siguiente sentencia:

```
psql postgres postgres
```

Con este comando accederemos al cliente por consola que postgres instala por defecto. Una vez conectados a la base de datos de ejemplo cambiamos la contraseña del usuario de la base de datos postgres. Ejecutamos la siguiente sentencia:

```
alter user postgres with password '*nueva contraseña*';
```

Una vez cambiada la contraseña salimos del cliente con el comando: \q. Opcionalmente, existe un cliente gráfico para postgres llamado pgAdmin 3 totalmente gratuito. Para instalarlo ejecutamos el comando:

```
apt-get install pgadmin3
```

Una vez instalado, añadimos una conexión nueva con el servidor postgres haciendo click en "Añadir conexión a un servidor" en la parte superior izquierda. Y rellenamos el formulario con los siguientes datos:

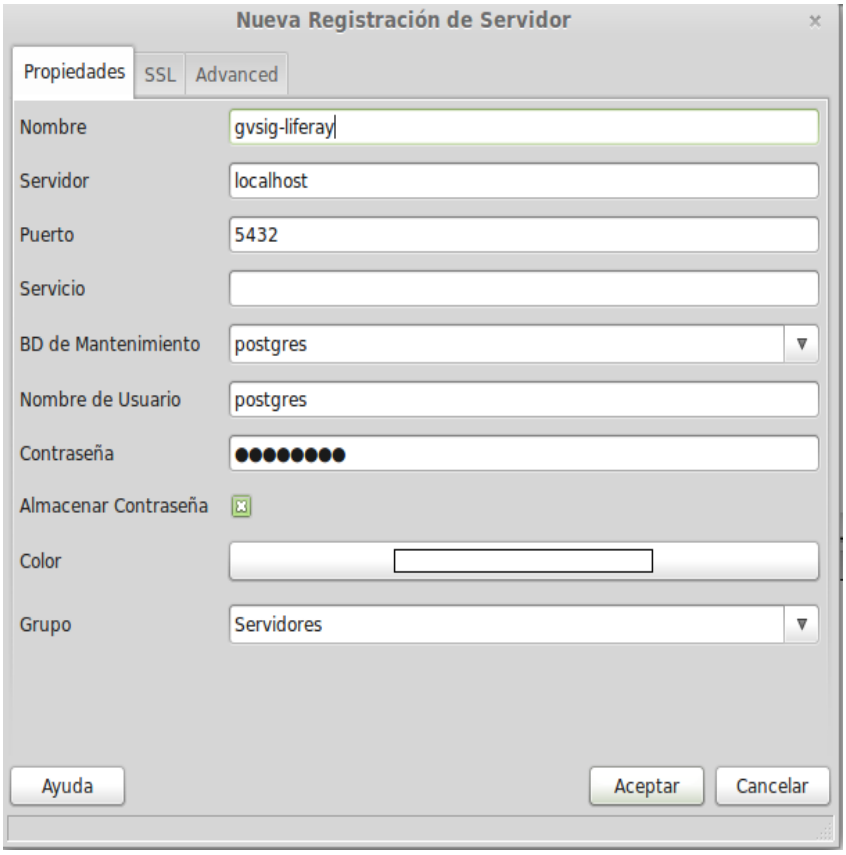



**Figura 7.3** Añadir nueva base de datos

Si todo ha ido bien, en la parte izquierda observaremos un árbol con todos los servidores disponibles. Pulsamos botón derecho sobre el nodo **Base de datos** que cuelga de nuestro servidor y seleccionamos **Nueva base de datos.** Se mostrará una ventana con todas las opciones de configuración posibles. En nuestro caso, indicamos el nombre **Liferay** y el propietario **postgres** y hacemos *click* en **Aceptar.** Con estos pasos hemos creado una base de datos nueva para que el servidor Liferay pueda crear sus esquemas, tablas, vistas… [pugsso14]

Instalado ya el servidor, configurado los dos usuarios y creado la base de datos solo nos queda configurar el servidor de Liferay para que se conecte a la base de datos recién creada. La configuración de la base de datos en Liferay se realiza con el archivo **portal-ext.properties.** Este archivo se crea en el directorio de instalación de Liferay y se carga cuando arrancamos el servidor. Existen numeras opciones de configuración como los idiomas disponibles, la franja horaria del servidor, opciones de seguridad, opciones de usuarios, roles…

El primer paso es acceder al directorio de instalación de Liferay. Una vez dentro, creamos el archivo **portal-ext.properties** y añadimos las siguientes líneas:

```
#
# PostgreSQL
#
jdbc.default.driverClassName=org.postgresql.Driver
jdbc.default.url=jdbc:postgresql://localhost:5432/liferay
jdbc.default.username=postgres
jdbc.default.password=*contraseña del usuario de la base de
datos*
```

Con estas líneas indicamos al servidor que el driver que debe usar es *org.postgresql.Driver*, que la URL de la base de datos es *jdbc:postgresql://localhost:5432/liferay* y el usuario y contraseña de la base de datos. Se arranca de nuevo el servidor Liferay y comprobamos que la configuración de la base de datos es correcta.

Se ha finalizado un ejemplo de configuración a modo de demostración. En un entorno real, deberíamos *de* crear las diferentes máquinas virtuales y separar por un lado Liferay del gesto de base de datos. Además de que son necesarias algunas opciones de configuración para obtener un sistema más robusto y mejor adaptado.

# PARTE III CONCLUSIONES

# Capítulo 8. Conclusiones

En este capítulo se presentan las conclusiones del trabajo. En la sección 8.1 se resumen los principales pasos llevados a cabo para obtener la solución a la problemática planteada, recopilando los aspectos más importantes. Finalmente, en la sección 8.2 se relaciona con posibles trabajos futuros.

## 8.1 Conclusiones del trabajo

La principal conclusión del trabajo realizado es la demostración de que aplicar una metodología con sus fases y principios para el desarrollo de software es la mejor forma de asegurar que el trabajo realizado no fracase y se genere software de calidad. El trabajo se ha enfocado en dos partes: planteamiento del problema, análisis y propuesta de solución y la explicación de las diversas fases del proceso llevadas a cabo para la obtención de la solución propuesta fruto de la primera parte.

La motivación y definición de objetivos por un lado ha puesto de manifiesto el trasfondo que posee este proyecto y por otro, en que dirección debe ir este proyecto. El proyecto no solo aporta la solución a un problema planteado, sino que ayuda a la difusión de conocimiento con el fin de aumentar la comunidad de usuarios y personas que puedan aportar su granito de arena al proyecto y cambiar el modo actual de negocio, donde prevalece que el conocimiento se privatice en vez de compartirlo. Los objetivos propuestos marcan una dirección hacia el seguimiento del proceso de ingeniería del software, pero además también marcan una dirección personal de ampliación del conocimiento del autor y toma de contacto con el mundo laboral.

El problema propuesto se basa en la falta de organización del contenido web debido a que se encuentra en dos portales diferentes los cuales no están definidos bien los contenidos que deben de mostrar. Además, su estructura está muy segmentada produciendo así muchos apartados y muchas páginas web. Y finalmente, necesita actualizar su estilo y adaptarse a las nuevas plataformas que cobran vital importancia en el mundo actual. Se propone una solución fruto del análisis de varias opciones en base a los criterios definidos por el cliente. La opción elegida es más se adapta al problema.

La visión del proyecto establecida marca que el resultado debe ser un portal web que cumpla con los requisitos especificados por el cliente y ofrecer un mantenimiento a posibles errores que puedan surgir. La especificación de requisitos funcionales y no funcionales establece que el portal debe de ayudar a mejorar la comunicación hacia la comunidad, debe de ser accesible desde todas las plataformas y fácil de gestionar. Se concluye también que la solución propuesta ya cubre, en mayor o menor medida, gran parte de las funcionalidades indicando que posiblemente sea la más acertada.

Con la definición de los casos de uso, se ha definido que comportamiento debe de tener el sistema frene a los distintos requerimientos. A grandes rasgo, la mayor parte de las acciones se realizarán a través de un menú de administrador y de forma clara y sencilla. La visión lógica ha determinado la

estructura interna del *portlet descargasBuilds*. Se ha decidido que el *portlet* implementará el patrón modelo-vista-controlador ya que conseguimos separar la lógica de negocio de la vista teniendo así un código menos acoplado y más mantenible. El *portlet* estará formado por tres clases que modelan una tabla, un elemento de la tabla y el controlador. El paso de mensajes entre la vista y el modelo de datos se realizará a través de un controlador. Y finalmente, se ha diseñado una arquitectura formada por un servidor físico en el cual se alojan máquinas virtuales para aprovechar sus ventajas. El resultado es una arquitectura que ofrece protege los datos de la organización por parte de ataques externos como fallos en el sistema.

La fase de construcción está compuesta por dos iteraciones. En la primera se explica los pasos a seguir para llevar a cabo el diseño realizado en las fases anteriores. Partiendo del análisis y diseño previos, aprovechando las funcionalidades que ofrece Liferay se crean las estructuras y plantillas que dan como resultado un modelo de datos fácil de gestionas y muy adaptativo a las necesidades actuales o futuras. En la segunda fase se lleva a cabo la implementación del *portlet* teniendo en cuenta, al igual que en la primera iteración, el análisis y diseño previos. Gracias a esto, el resultado cumple con el requisito definido.

Y por último, la fase de transición expone la configuración necesaria para realizar la entrega y poner en marcha todo el sistema. La configuración, aunque no sea la real, exponen que funcionalidad le aporta al sistema cada parte de la arquitectura. El servidor apache a modo de *proxy* a nuestra arquitectura, el servidor Liferay núcleo del proyecto que se apoya en un servidor de base de datos para la persistencia.

## 8.2 Futuros trabajos

El proyecto desarrollado abre nuevas vías de continuación partiendo del proyecto realizado. Entre ellas se destaca:

- **Diseño y desarrollo de un tema gráfico en una plataforma Liferay para mejorar el aspecto visual del portal**: partiendo de este proyecto, ampliarlo añadiendo el diseño y desarrollo de un tema gráfico en Liferay y la integración con otros *framework* como AlloyUI.

- **Análisis, diseño y desarrollo de un *portlet* en una plataforma Liferay usando Spring MVC:** partiendo del *portlet* diseñado, reimplementarlo añadiendo más funcionalidades para la obtención de un resultado más sofisticado usando el *framework* Spring MVC.

- **Creación de una comunidad dentro de un portal basado en Liferay:** diseño y realización de una comunidad de usuarios dentro del portal con foros, páginas privadas personalizables, correo interno, *wikis*, gestión de roles y permisos.

# Bibliografía


ant Apache Ant http://ant.apache.org/

bsmqmus Balsamiq Mockups http://balsamiq.com/products/mockups/

eclipse Eclipse https://eclipse.org/

gmvv14 Asociación gvSIG (2014). Misión, valores y visión. Consultada el 28 de Marzo del 2014, http://www.gvsig.org/plone/home/organization/mision-vision-y-valores/

jcwdi13 Eric Tiggeler,, *Joomla! 3, Crea sitios web dinámicos e interactivos, pp 25-27* ANAYA, 2013. Consultada el 15 de Julio de 2014.

jsoup JSoup http://jsoup.org/

jsptglu JSP Taglib Liferay-ui: http://docs.liferay.com/portal/6.2/taglibs/liferay-ui/tld-summary.html

jsptbp2 JSP Taglib Portlet 2_0: http://docs.liferay.com/portlet-api/2.0/javadocs/

lsdk Liferay SDK http://sourceforge.net/projects/lportal/files/Liferay%20Portal/6.2.1%20GA2/liferay-plugins-sdk-6.2-ce-ga2-20140319114139101.zip/download

lppo14 Liferay Portal (2014), Liferay Portal, Productos, Incio. Consultada el 15 de Julio del 2014, http://www.liferay.com/es/products/liferay-portal/overview/

lugacst14 User guide Liferay Portal 6.2, *Advanced Content with Structures and Templates*, 2014. Consultada el 25 de Agosto de 2014, http://www.liferay.com/documentation/liferay-portal/6.2/user-guide/-/ai/advanced-content-with-structures-and-te-liferay-portal-6-2-user-guide-03-en

lugel14 User guide Liferay Portal 6.2, *Edtions of Liferay*, 2014. Consultada el 10 de Octubre del 2014, http://www.liferay.com/documentation/liferay-portal/6.2/user-guide/-/ai/editions-of-liferay-liferay-portal-6-2-user-guide-15-en

ldgapp14 Developer guide Liferay Portal 6.2, *Anatomy of a Portlet Project*, 2014. Consultada el 28 de Agosto del 2014, http://www.liferay.com/documentation/liferay-portal/6.2/development/-/ai/anatomy-of-a-portlet-project-liferay-portal-6-2-dev-guide-03-en

ldgutppe14 Developer guide Liferay portal 6.2, *Understanding the Two Phases of Portlet Execution*, 2014. Consultada el 30 de Agosto del 2014, http://www.liferay.com/documentation/liferay-portal/6.2/development/-/ai/understand-portlet-execution-phases-liferay-portal-6-2-dev-guide-03-en

muml07 Paul Kimmel, *Manual de UML: guía de aprendizaje, 2007, McGaw-Hill Interamericana.*

mid710 Todd Tomlinson, *Manual imprescindible de Drupal 7*, p 21, ANAYA, 2010. Consultada el 15 de Julio de 2014.

piac01 Bruce Momjian, *PostgreSQL, Introduction and Concepts*. p. 236. Pearson Education Corporate Sales Division, 2001. Consultada el 2 de Diciembre del 2014.

pugsso14 PostgreSQL 9.3.5 Documentation User Guide, *Server setup and Operation*, 2014. Consultada el 10 de Noviembre del 2014, http://www.postgresql.org/docs/9.3/static/runtime.html

se911 Ian Sommerville, Mass Boston, *Software Engineering 9*, pp. 50-53, Pearson, 2011. Consultada el 2 de Diciembre del 2014.

vug14 Apache Velocity Project, *User guide*, 2014. Consultada el 25 de Agosto de 2014, http://velocity.apache.org/engine/devel/translations/user-guide_es.html

wcms14 Wikipedia (2014). CMS. Consultada el 1 de Abril del 2014, http://es.wikipedia.org/wiki/Cms

wcw14 Wikipedia (2014). Contenedor web. Consultada el 1 de Abril del 2014, http://es.wikipedia.org/wiki/Contenedor_web

wmv14 Wikipedia (2014). Máquina virtual. Consultada el 2 de Abril del 2014, http://es.wikipedia.org/wiki/Maquina_virtual

wnas14 Wikipedia (2014). NAS. Consultada el 2 de Abril del 2014, http://es.wikipedia.org/wiki/Network-attached_storage

wpcabw13 Christophe AUBRY, *Worpres 3.5, Un CMS para crear y administrar blogs y sitios webs*, 2013. Consultada el 15 de Julio del 2014.

wplo14 Wikipedia (2014). Plone. Consultada el 31 de Marzo del 2014, http://es.wikipedia.org/wiki/Plone

wsvlt14 Wikipedia (2014). Servlet. Consultada el 1 de Abril del 2014, http://es.wikipedia.org/wiki/Servlet

wsw14 Wikipedia (2014). Servidor Web. Consultada el 2 de Abril del 2014, http://es.wikipedia.org/wiki/Servidor_web