Escuela Técnica Superior de Ingeniería Informática
Universitat Politècnica de València

Diseño e implementación con tecnologías JSF y JPA de una web para un grupo de investigación en biotecnología

Proyecto Final de Carrera

Ingeniería técnica informática de sistemas

Autor: Carlos García-Miguel Pérez-Luengo

Director: Germán Moltó Martínez

Co-director: Guillermo Rodrigo Tárrega

30-06-2013

Resumen

La visibilidad en Internet de los grupos de investigación es hoy esencial para adquirir notoriedad en la comunidad científica pertinente, para una correcta divulgación de los avances realizados, y para el continuo desarrollo del mismo al posibilitar la captación de nuevos miembros y la colaboración con otros investigadores. En este Proyecto Fin de Carrera, hemos diseñado e implementado una página web para un nuevo grupo de investigación en biotecnología de sistemas. Nuestro desarrollo web se ha llevado a cabo usando las nuevas tecnologías Java Server Faces (JSF) y Java Persistence API (JPA). La web contiene información pública relativa al grupo de investigación, y es de fácil gestión y mantenimiento para el investigador principal del grupo a través del propio portal. Ésta muestra una página principal donde aparecen noticias destacadas, con un menú que redirige al usuario para ver los contenidos relativos al personal del grupo, las líneas principales de investigación, las publicaciones científicas, los proyectos financiados, el wetware y software disponible, y la información de contacto. La web se actualiza de forma dinámica, realizando consultas para extraer la información de tablas o ficheros. La web también permite la descarga del software desarrollado por el grupo. De relevancia, se ha desarrollado el sistema para poder ser escalado, a fin de seguir siendo válido para una gestión de contenidos más complejos de los que se implementan en este proyecto. Para dar soporte real a la web, se ha creado una máquina virtual en un nodo Xeon Quad del cluster del Instituto de Biología Molecular y Celular de Plantas, CSIC-UPV (Ciudad Politécnica de la Innovación).

Objetivos

- Desarrollo de una web con especificaciones propias de un grupo de investigación en biotecnología y con fácil mantenimiento a través del mismo portal.

- Aprendizaje y puesta en funcionamiento de las nuevas tecnologías para el desarrollo web Java Server Faces (JSF) y Java Persistence API (JPA).

- Generación de una solución eficiente al problema general de la visibilidad en Internet, que puede ser aplicado tanto al mundo académico, como ha sido el caso, o al mundo empresarial.

Tabla de contenidos

1. Introducción

Proponemos desarrollar una página web para un grupo de investigación con una fácil y eficiente gestión de contenidos a través del propio portal. Para ello proponemos usar tecnologías Java.

1.1. Visión global

El proyecto web a desarrollar consiste en una página web pública para un grupo de investigación en biotecnología. Este proyecto ha sido encargado por el investigador principal del grupo y será con él con quien realicemos las entrevistas necesarias para detallar las especificaciones de diseño. Se desarrollará una página web desde cero y totalmente adaptada a las necesidades y aportaciones del grupo de investigación.

En el proyecto que se va a llevar a cabo se han detallado unas especificaciones concretas. El objetivo principal que tiene que cumplir la web es la divulgación de contenidos científicos. A través de la web, cualquier usuario podrá saber qué líneas de investigación está siguiendo el grupo y se facilitará el acceso a todos los artículos de investigación publicados. Por otro lado, el grupo de investigación para el cual se va a desarrollar la web trabaja en el campo de biotecnología de sistemas y, por lo tanto, muchos de sus artículos publicados se basan en el desarrollo de diversos algoritmos. A través de la web, se creará una plataforma para tener acceso a estos algoritmos, permitiendo su descarga para que cualquier usuario pueda reproducir los resultados obtenidos. Por último, la web servirá de soporte de trabajo para el grupo de investigación, de manera que través de la misma se podrá tener un inventario del material del laboratorio. El listado de consulta podrá ser modificado en cualquier momento por un administrador.

La visibilidad en Internet de los grupos de investigación es hoy esencial para adquirir notoriedad en la comunidad científica pertinente, para una correcta divulgación de los avances realizados, y para el continuo desarrollo del mismo al posibilitar la captación de nuevos miembros y la colaboración con otros investigadores [1].Por ejemplo, aunque existen buscadores de artículos por científico, es muy útil que cada grupo de investigación posea una herramienta para poder mostrar, de una forma detallada, fácil y efectiva, el trabajo desarrollado por el mismo. A partir de la página web se mostrará también información de contacto con el grupo, a finde permitir la interacción con otros. Esto fomentará la colaboración, aumentando así la productividad e impacto del trabajo.

En suma, este proyecto web ayudará al grupo de investigación para ser su carta de presentación (o ventana) al mundo exterior. La web tendrá tres funciones claramente diferenciadas. En primer lugar, se mostrará información del grupo, tanto de personal como de las líneas de investigación. En segundo lugar, aportará accesos a través de enlaces a los artículos publicados por el grupo en revistas científicas. Y por último, tendrá una herramienta que permita la gestión del propio laboratorio, con información pública y privada.

1.2. Motivación

Se ha decidido desarrollar este proyecto porque existe un equipo de investigación nuevo que desea y necesita tener su propio portal web para poder comunicarse con la comunidad científica. Por tanto, con este proyecto se va a desarrollar una herramienta útil que cumpla las necesidades y especificaciones proporcionadas por el investigador principal del grupo, aportando contenidos y funcionalidad. Por otro lado, otra de las razones que nos han impulsado a desarrollar este proyecto ha sido la construcción desde cero y de forma independiente de una web. Según la funcionalidad, se escogerá una tecnología conocida que ofrezca un producto de calidad.

Resulta especialmente interesante el aprendizaje de la tecnología sobre la que se hará el desarrollo de la página web. En consecuencia, el proyecto estará basado en un diseño en tres capas, que implicará una separación entre capas bien definidas que se ejecutarán sobre plataformas independientes. Estas tres capas habitualmente se conocen como capa de presentación, capa de negocio y capa de datos. Más adelante, se detallará cada una de ellas.

Por último, ha sido también motivante el hecho de conocer cómo es el trabajo en un grupo de investigación a la hora llevar a cabo este desarrollo. Ha sido especialmente interesante ver las similitudes y diferencias de trabajo con respecto a una empresa privada. Con este proyecto ganaremos conocimientos y métodos de trabajo que serán muy útiles en el futuro. Se pretende que, con este proyecto, el grupo de investigación alcance gran visibilidad y promoción en Internet y que, en un futuro, este grupo consiga los reconocimientos pertinentes por su trabajo.

1.3. Estructura del proyecto

El desarrollo del proyecto se llevará a cabo mediante cuatro procesos seriados. En el primer proceso se definirán las especificaciones del cliente en relación a la funcionalidad y contenidos de la web. En el segundo, se obtendrá un diseño de la web

analizando diferentes estrategias. Una vez se tenga el diseño final, se procederá en el tercer paso a la implementación. Finalmente, se lanzará la web a fin de evaluarla junto con el cliente.

La primera fase consiste en las **especificaciones de diseño**, donde se realizarán diferentes reuniones con el cliente. En este caso, el cliente será el líder del grupo de investigación con el que se ha contactado para el desarrollo de la web. En estas reuniones, el cliente expondrá sus necesidades y se le realizarán todas aquellas preguntas necesarias a fin de establecer los requisitos requeridos para la construcción de la web. Puesto que el cliente no posee conocimientos técnicos en programación web, se realizará todo tipo de recomendaciones a fin de conseguir una buena herramienta adaptada a todas las necesidades del grupo. Es importante una buena definición de contenidos y tareas a realizar, ya que el correcto desarrollo del proyecto depende en gran medida en esta fase.

La fase de **diseño** es el segundo paso y aquí existen tres objetivos claros. Para empezar, se definirá la estrategia de diseño de la web. Seguidamente, tendremos que crear una estructura para soportar la información. Optaremos por una arquitectura de tres capas por la integridad y robustez que ofrece al proyecto. Además, es conveniente producir un diseño visual que satisfaga las necesidades del cliente. En esta fase deberemos entender cuáles son las necesidades científicas (en otros casos de negocio) que motivan el proyecto. Esto implicará conocer cuáles son las características diferenciadores de nuestro cliente. Una vez que entendamos esto, se podrá producir un diseño que enfatice estas características.

La tercera fase es la de **implementación**. Antes de programar, se buscarán las herramientas apropiadas para construir nuestro proyecto y se justificará la elección de una determinada tecnología. La implementación es la parte del proceso que más varía entre proyectos ya que es algo que depende en gran medida del programador. A día de hoy, existe un gran número de herramientas, muchas de ellas de código abierto (en *Open Source)*. Será tarea del desarrollador elegir una tecnología apta para el fin del proyecto que va a desarrollar. Más adelante se explicará brevemente qué ventajas nos aporta la tecnología que se ha escogido.

La cuarta fase consiste en la **post-implementación**, seguida de la **evaluación y pruebas** del proyecto. Una vez implementada la web, dejaremos un tiempo para ajustar el contenido de la misma, permitiendo al cliente poder hacer alguna modificación en la parte visual o incluso en la de funcionalidad. Durante esta fase, habrá una serie de entregas o citas donde el cliente podrá probar y ver el contenido, así como comprobar el mantenimiento o uso de la web, verificando así si se cumplen los requisitos exigidos para cubrir las necesidades. Ciertamente, esta fase pudiera dilatar el tiempo para completar el proyecto debido a nuevas peticiones del cliente. También se procederá a la implantación de la web.

1.4. Breve descripción de tecnologías

Para el desarrollo de este proyecto se ha optado por la tecnología Java[2,3, 4]. Concretamente, por la tecnología Java Server Faces (JSF) como núcleo principal para desarrollo de la capa de presentación y de negocio. Esta tecnología ofrece componentes extendidos (respecto a HTML estándar) muy usados en el mundo de la programación web, muchos de estos componentes son muy robustos e interactúan bien unos con otros, dando la capacidad de generar una página web completa con pocas líneas de código.

Para gestionar la capa de persistencia, que es la capa de base de datos, se ha escogido la tecnología Hibernate[5] ya que se integra perfectamente con JSF, pudiendo llevar a cabo desarrollos sólidos y eficaces. Hibernate además, nos ofrece la posibilidad de trabajar con Java Persistencia API (JPA) que nos facilita el acceso e interacción de los datos, debido al mapeo de entidades con anotaciones.

2. Especificaciones de diseño

En este capítulo, explicaremos en qué consiste la fase inicial del proyecto web, conocida como especificaciones de diseño o como fase de negociación. Los puntoscomunes a cualquier proyecto de desarrollo en esta fase son los siguientes: i) entender claramente cuáles son los requisitos del proyecto, y ii) llegar a un acuerdo con el cliente sobre el alcance del mismo.En este proyecto, obviaremos la parte de rentabilidad, ya que se trata de un proyecto creado en beneficio del cliente (también del programador) y, por tanto, éste no tendrá ningún coste económico.Esta fase del proyecto es especialmente importante para fijar con claridad las especificaciones de diseño a seguir, así como la estructura y la funcionalidadde la página web. Con ello evitaremos en fases posteriores grandes cambios en la implementación ypodremos visualizar de antemanola solución que buscamos. En esta fase nos ayudaremos de los gráficos y esquemas que estimemos oportunos y que iremos detallando en la memoria.

2.1. Descripción general

En líneas generales, el proyecto consiste en una web de acceso público, por tanto, no se requerirá registrarse para acceder a su contenido. A través de la web, podremos acceder a información relativa al grupo de investigación. A través de un menú, accederemos a diferentes contenidos como son el personal que forma el grupo de investigación, las líneas principales de investigación, las publicaciones científicas, los proyectos financiados, el software disponible o la información de contacto (ver por ejemplo [6]).

En cuanto al mantenimiento, éste ha de ser lo más sencillo posible y, una vez puesta en marcha la web, será una preocupación para el administrador de la web, que en este caso será el investigador principal del grupo de investigación. El sitio web se actualizará de forma dinámica y se rellenará todo el contenido a partir de los ficheros fuentes que podrán ser creados, editados o modificados por el administrador de la web.

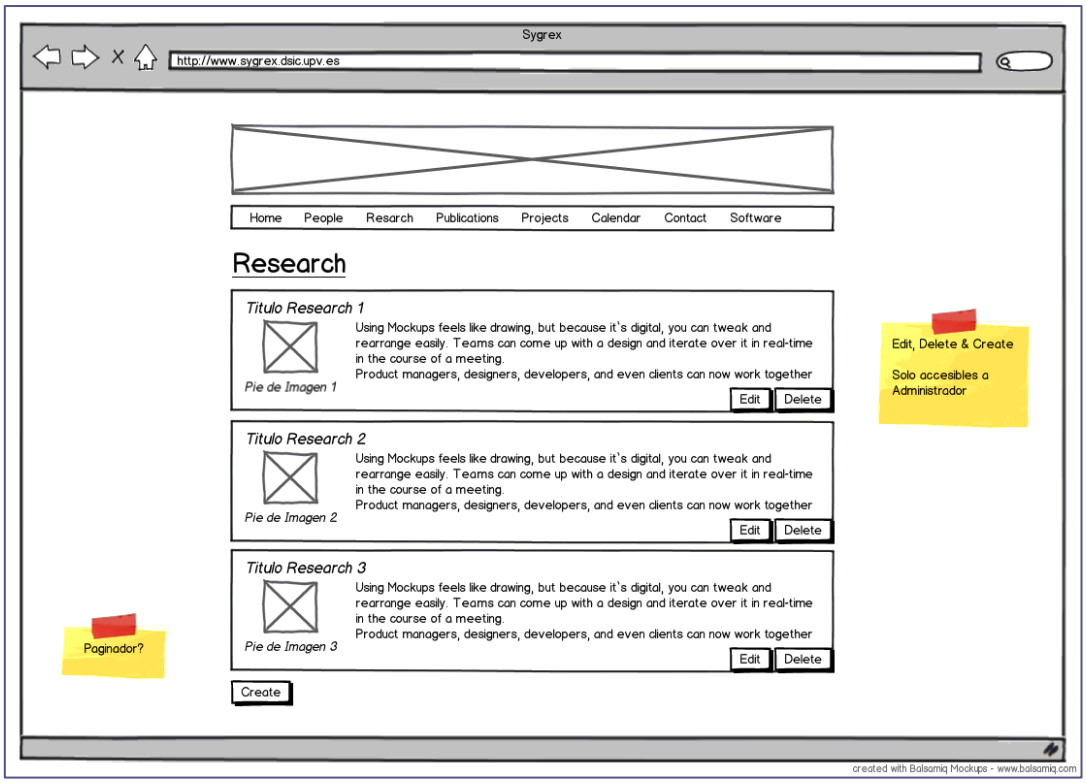

Fig. 2.1. Esquema de especificaciones de diseño de la página web.

2.2. Requisitos específicos.

Uno de los requisitos de la web es la estructura del diseño de la página web y debe seguirse una serie de reglas que se encuentran estandarizadas gracias al uso continuo. Los usuarios, a partir de la práctica de navegar, esperan encontrar cada elemento web en un determinado sitio, por lo tanto, un buen diseñador web tendrá en cuenta estas reglas con el fin de facilitar el uso de la web. Para comenzar, el cliente, nos ha dado unas indicaciones de estructura básicas que consisten en un encabezado con el nombre del laboratorio, un menú principal horizontal en la parte superior de la página y un cuerpo principal donde se alojará el contenido de los diferentes apartados.

Concretando la estructura del diseño, los diferentes apartados del menú principal de la página son: las líneas principales de investigación, las publicaciones científicas, los proyectos financiados, el material o recursos del laboratorio (wetware), el software disponible, y por último, la información de contacto. El nombre de las secciones es precisamente éste, por lo que queda muy claro el contenido de cada una de ellas. Por tanto, esquemáticamente la página tendrá, más o menos, la apariencia que se muestra en la Fig. 2.1.

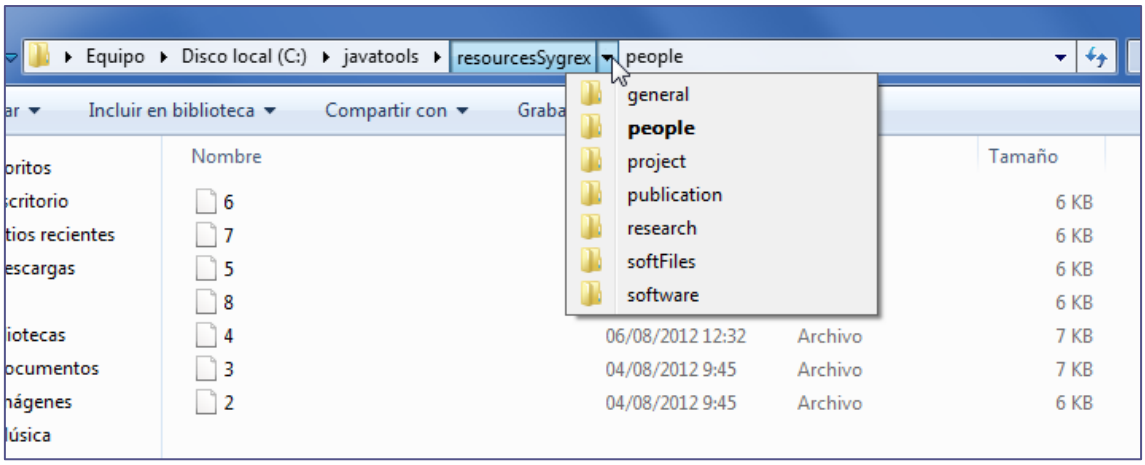

Fig. 2.2. Estructura de directorios.

Tal y como puede apreciarse en la Fig. 2.1, la página web dispondrá de un menú general con un número concreto de elementos para irse desplazando por las diferentes páginas de la aplicación. Este menú constara de los siguientes apartados: *People* para visualizar al personal que trabaja en el laboratorio, *Research* para conocer las diferentes investigaciones que se están llevando a cabo, *Publications* para ver las diferentes publicaciones del grupo de desarrollo ordenadas por año de publicación, *Projects* para ver los proyectos conseguidos por el grupo, *Wetware* para llevar un inventario sobre todos los tipos de elementos principales necesarios y gestionados por el grupo de investigación, y *Software* que será una página para la gestión y descarga de diferentes herramientas que pueden ser útiles para el grupo de investigación.

Siguiendo, más o menos, el estándar de cualquier página web, existe una página de bienvenida a la aplicación mediante una opción de menú llamada *Home* con un listado de noticias o de elementos de utilidad para cualquier usuario de la aplicación y un carrusel a modo de bienvenida a la página. Además, se ha incluido una apartado llamado *Contact* cuyo propósito es servir de punto de información sobre la localización del grupo de investigación y como ponernos en contacto con el mismo.

La página web se va a desarrollar en un único idioma que será el inglés, que es el idioma para divulgar la investigación. Por tanto, no será necesario hacer una web multi-idioma.

2.2.1.Estructura de archivos

Es necesario que una página web tenga una estructura de archivos ordenada tanto para facilitar su mantenimiento como la gestión de contenido. Cuando nos iniciamos en el diseño de una página web tiende a pensarse que la estructura de los archivos y directorios en los que se encuentra organizada la página carece de importancia. Sin

embargo, es una idea equivocada, es importante y aconsejable definir esta estructura de archivos previamente, de una manera lógica y ordenada.

Como puede verse en la Fig. 2.2, se ha creado un directorio de fuentes en el proyecto que hemos llamado *ResourcesSygrex*, donde disponemos de un directorio por cada una de las pestañas del menú principal de la web. Cada directorio tiene una estructura en función de la información que debe gestionar y que es directamente dependiente del contenido de la base de datos. Estos directorios están pensados para que se vayan regenerando según sea necesario acceder al contenido de los datos. Además, toda la gestión de ficheros se efectúa a través de la web de manera automática y únicamente el administrador de la web tendrá acceso a estos ficheros para modificar cualquier información que no sea coherente con el estado de la base de datos.

Esta ruta general se define a nivel de aplicación en el web.xml para que si la aplicación cambiase de entorno, simplemente hayamos de cambiar este valor al arrancar la aplicación y todo funcionaría correctamente si la nueva estructura tiene los permisos suficientes para poder leer en los directorios, crear nuevos archivos en ellos o borrar los elementos si fuera necesario.

2.2.2. Funciones de la página web

La funcionalidad principal de la web consiste en la visualización ordenada de una serie de campos dependientes de cada pantalla, para todos los roles de la aplicación, y un mantenimiento sobre todas las páginas de la aplicación disponible para un perfil administrador. En este punto iremos detallando que necesita implementarse para conseguir la funcionalidad que se espera de la web así como otros requisitos.

Para cada pantalla de la aplicación: *Home, Research, People, Publications, Projects, Wetware, Software* y *Contact* se requiere implementar los siguientes puntos 2 perfiles diferentes. Por una parte, se implementará un perfil anónimo que solamente podrá leer el contenido de la página web. Y por otra parte, se implementará un perfil administrador que podrá leer, crear, modificar y borrar cualquier elemento de información de la base de datos, es decir, podrá realizar cualquier operación sobre la base de datos. Todas estas acciones deberán de ser lo suficientemente intuitivas para que este usuario administrador pueda hacer los mantenimientos sin llegar a leer un manual de usuario. Hemos de tener en cuenta diseños intuitivos con una o dos pantallas, evitando cualquier complejidad.

Además de los diferentes perfiles, hay que implementar las siguientes funcionalidades: implementación de un Carrusel para la página de inicio, el sistema de autenticación para el usuario administrador y, por último, el sistema de procesado de imágenes para reducir la carga de la imagen enviada y que el envío de información Servidor-Cliente sea más rápido.

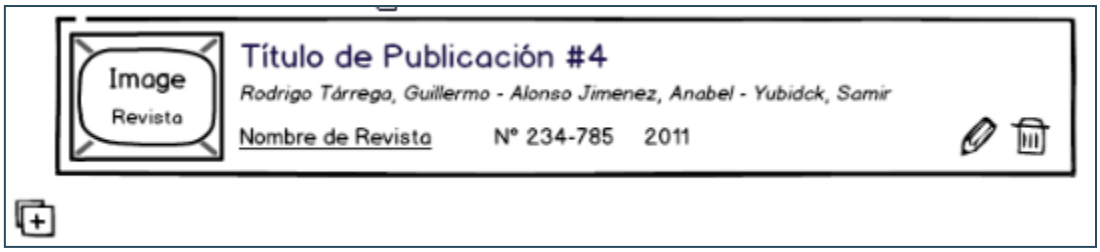

Fig. 2.3. Ejemplo de mantenimiento.

La funcionalidad de la web consiste en mostrar información de interés del laboratorio y vamos a priorizar la calidad a la cantidad, por ello buscaremos un producto que sea fácil y rápido para usar. Cualquier texto que sea publicado será un texto resumido de manera que el usuario pueda leer la máxima información de interés.

Por otro lado, se ha optado por una estructura de diseño minimalista, ordenada, clara, evitando menús repetidos con la intención de que el usuario pueda encontrar rápidamente qué está buscando, sin necesidad de perderse entre los contenidos de otra sección.

2.2.3. Características de usuario

La web a desarrollar se trata de una web pública de forma que cualquier usuario se conectará de forma anónima, sin necesidad de haberse registrado previamente. Sin embargo, para realizar cualquier cambio en el contenido de la página web será necesario haberse autenticado como administrador de la web. Sólo el administrador de la web conocerá la contraseña que le permita validarse con permisos de administrador y, en este caso, el administrador será únicamente el investigador principal del grupo.

Hay que tener en cuenta que el usuario administrador no posee conocimientos informáticos, por tanto, se desarrollará una página en la que la actualización de la información sea algo sencillo que no requiera ninguna complejidad. Para facilitar el mantenimiento de los datos permitiremos editar o borrar cada elemento, así como también agregar un nuevo elemento. En la Fig. 2.3 puede apreciarse un ejemplo sobre el mantenimiento de un registro de publicación. Hemos de elegir un modo de operar con el contenido web similar al de la imagen y definirlo igual en todas las pantallas donde existan elementos modificables.

2.3. Análisis

Antes de comenzar con la implementación, realizaremos unos diagramas que nos ayudarán a estructurar el proyecto y a definir la funcionalidad general de la web. Existen muchos tipos de diagramas en UML para definir gran cantidad de funcionalidad o comportamiento. Para nuestro desarrollo web hemos decidido optar por los modelos más comunes, con el fin de que resulten de ayuda para ver en perspectiva el contenido lógico de la web y las funcionalidades principales que han de implementarse.

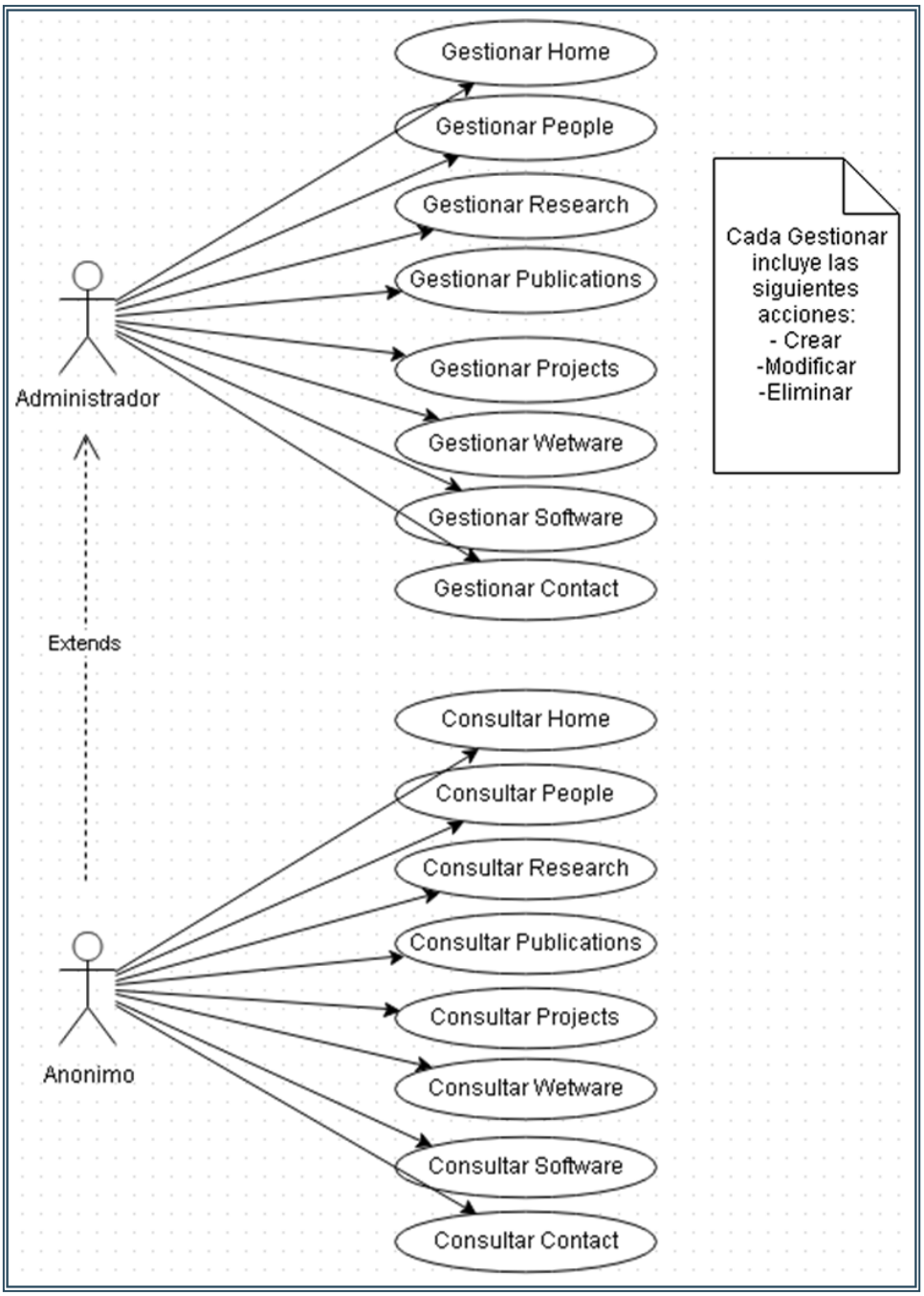

Fig. 2.4. Diagrama de Casos de Uso.

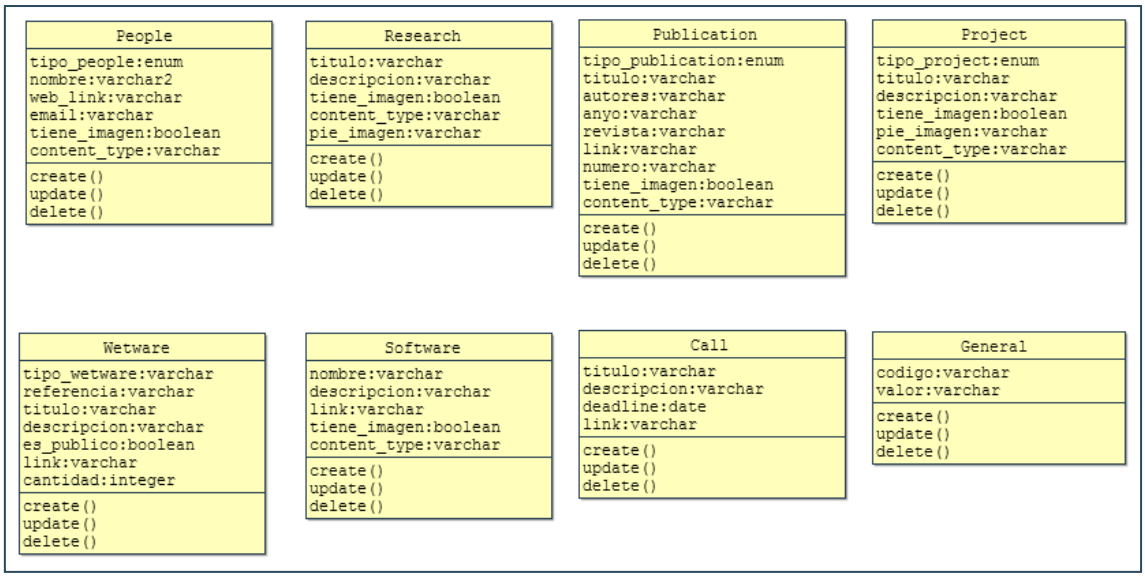

Fig. 2.5. Diagrama de clases.

2.3.1. Casos de uso

Podemos hacernos una idea de la funcionalidad que es requerida en el conjunto de la aplicación a partir de la Fig. 2.4 donde está presente toda la funcionalidad que contendrá la web. Existen otros elementos pero son estáticos y no requieren de interacción tipo de usuario por lo que no se muestran.

Como se puede ver, el usuario administrador puede ejecutar todas las acciones del usuario anónimo más un conjunto de funcionalidad extra para poder mantener cada pantalla.

2.3.2. Diagramas de clases

Una vez definida el conjunto de funcionalidad de la aplicación vamos a ver cuáles son las clases necesarias para poder mantener toda la información y las acciones requeridas sobre cada una de ellas.

Debido a que se va a usar la tecnología JSF y que las acciones sobre la base de datos pueden sufrir cambios según se desee mostrar la información, en el diagrama de clases sólo vamos a colocar las acciones básicas que estarán disponibles para el usuario administrador y vamos a obviar toda la parte de recuperación de datos.

En la Fig. 2.5 se han detallado todas las clases implicadas en el desarrollo de la aplicación. Podemos distinguir cada una de las propiedades definidas, así como los métodos requeridos para modificar el contenido de los diferentes elementos de la web.

2.3.3. Diagramas de secuencia

Para hacernos una idea sobre cómo llevar a cabo algunas de las acciones que se esperan más comunes vamos a verlas usando diagramas de secuencia de manera que así podamos ver y estructurar las diferentes capas que vamos a utilizar o las acciones que vamos a emplear.

En la Fig. 2.6, Podemos apreciar cómo se van a organizar las diferentes llamadas entre las clases y como estarían estructuradas la gran mayoría de llamadas de la aplicación a los objetos de base de datos.

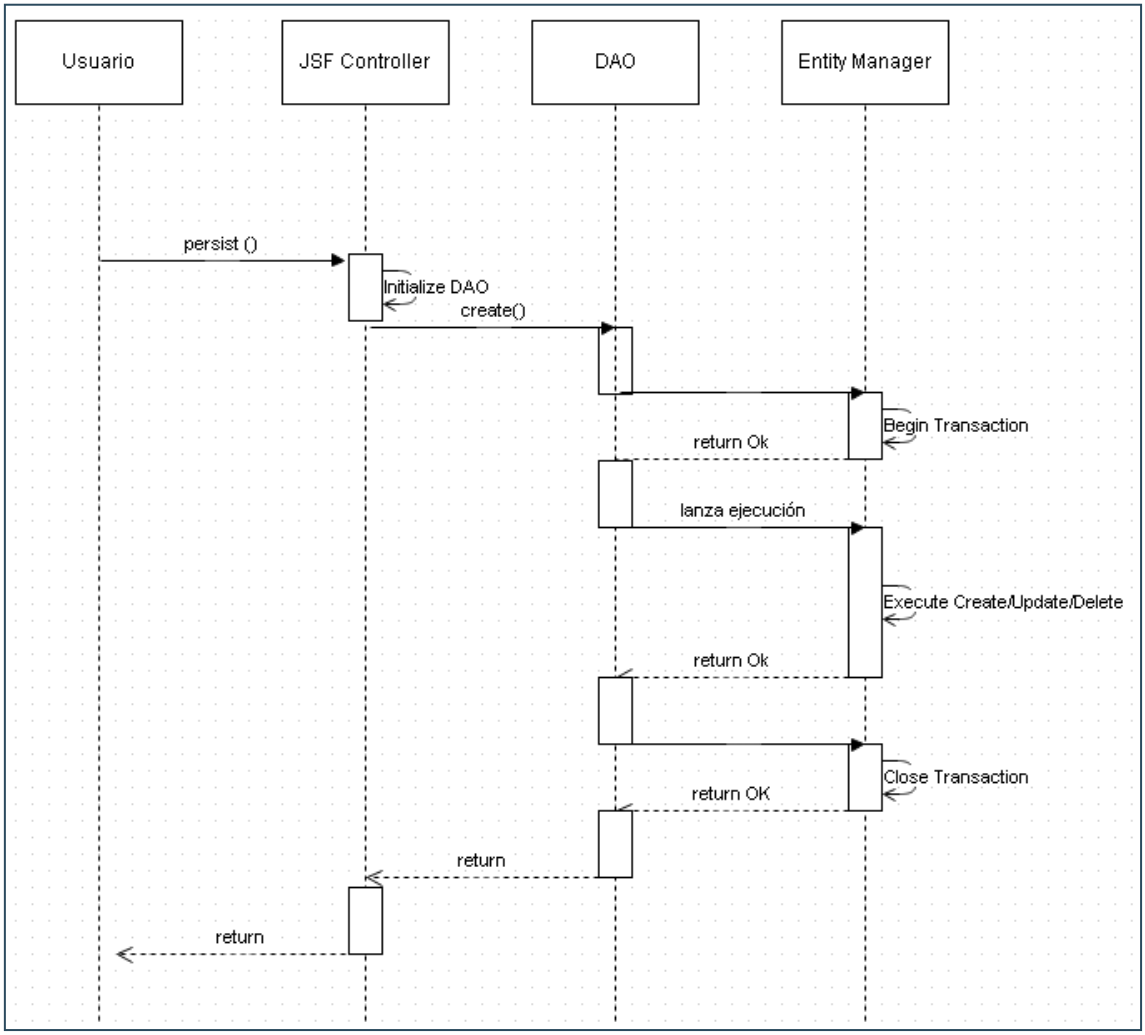

Fig. 2.6. Diagrama de secuencia para la modificación básica.

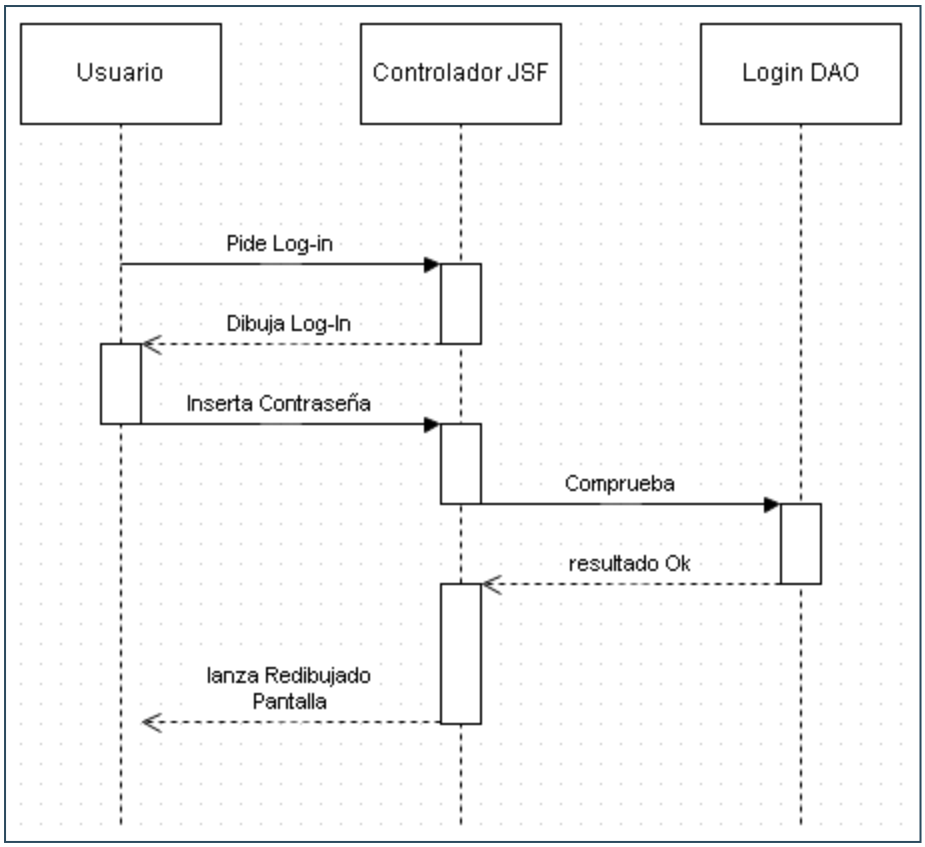

Fig. 2.7. Diagrama de secuencia para el sistema de login.

Por otra parte, también era importante conocer cómo hacer el cambio de usuarios así que hemos creado un diagrama de secuencia para que el cliente pueda pasar de un usuario anónimo a un usuario administrador. Este cambio se hará tal y como se muestra en la Fig. 2.7, donde una vez validado el cliente tendrá todos los permisos necesarios para poder realizar cualquier modificación en el contenido de la web.

También nos es útil a nosotros este diagrama a la hora de implementarlo ya que nos puede servir para definir los diferentes estados por los que puede pasar el proceso de validación y sobre qué estados pudiera ocurrir un error o cancelarse la acción.

3. Diseño y estrategias para el desarrollo web

En este capítulo, diseñamos la web y nos planteamos cuáles son las mejores estrategias para llevar a cabo tal diseño. Esto implica conocerlas características del cliente, principalmente aquéllas que lo diferencian, con el objetivo de seleccionar las tecnologías y herramientas más adecuadas. Es habitual que el cliente tenga una idea preconcebida del sitio web que quiere implementar, pero pudieran haber otras alternativas más eficaces. Por tanto, el programador tiene que visualizar las necesidades reales del cliente y ofrecerle una solución eficiente para satisfacerlas. En esta fase, se asesorará al cliente y se presentarán alternativas para apoyar su estrategia en la web.

3.1. Arquitectura de tres capas

Para el diseño de la web se ha optado por un modelo de tres capas (Fig. 3.1). Actualmente, éste es el modelo más utilizado en programación. La programación por capas es una arquitectura cliente-servidor cuyo objetivo principal es la separación de la lógica de negocios de la lógica de diseño, así como también separar la capa de datos de la capa de presentación al usuario. En este proyecto se distinguirán tres capas, y explicaremos la necesidad y funcionalidad de cada una de ellas.

La capa de presentación es la capa perceptible (de forma visual) para el usuario. De hecho, también es conocida como capa de usuario. Esta capa tiene la función de presentar el sistema, para comunicar y capturar la información del usuario. También es conocida como interfaz gráfica y debe tener la característica de ser *user-friendly* (esto es, entendible y fácil de usar). Esta capa se comunica únicamente con la capa de negocio.

En la capa de negocio residen los programas que se ejecutan. Ésta recibe las peticiones del usuario y envía las respuestas tras el proceso. Se denomina capa de negocio (o también de lógica del negocio) porque es aquí donde se establecen todas las reglas que deben cumplirse. Esta capa se comunica con la capa de presentación para recibir las solicitudes y presentar los resultados, y se comunica con la capa de datos para solicitar al gestor de base de datos almacenar o recuperar información de la misma. También se consideran aquí los programas asociados con la aplicación.

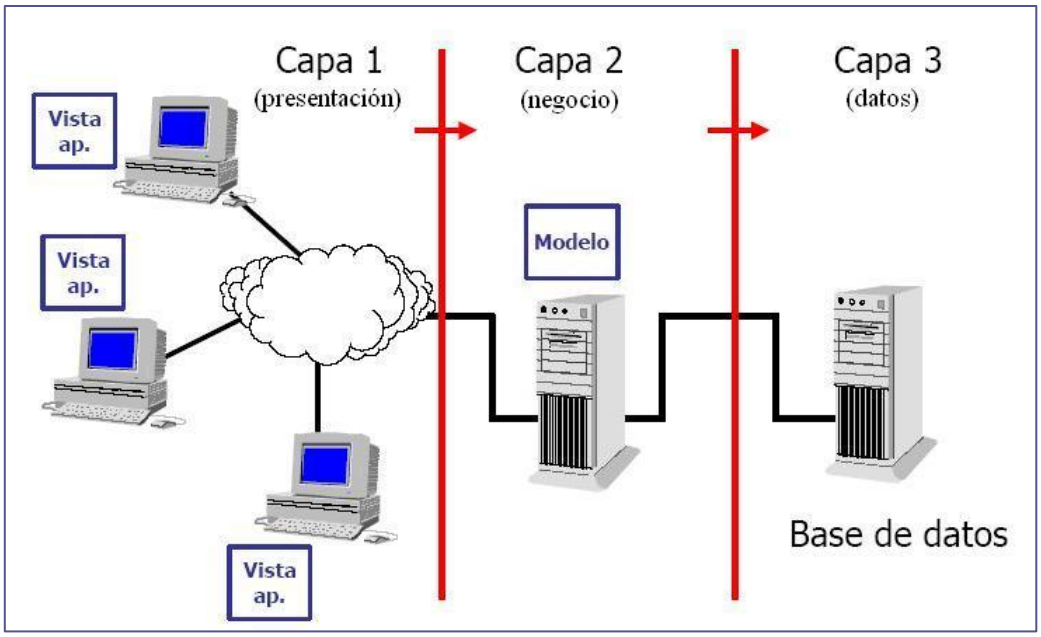

Fig. 3.1. Esquema de arquitectura de 3 capas.

En la capa de datos residen todos los datos que carga la web, y esta capa es la encargada de acceder a los mismos. Está formada por uno o más gestores de bases de datos que realizan todo el almacenamiento. Estos gestores reciben solicitudes de almacenamiento y pueden recuperar la información desde la capa de negocio.

3.1.1. Interfaz de usuario (Capa de presentación)

Una vez finalizada la especificación de requisitos, hemos procedido con el diseño visual o interfaz de usuario. Éste es uno de los componentes más importantes del sitio web. De hecho, la web es un medio visualmente rico y los usuarios han aprendido a confiar en sitios con presentaciones visuales de alta calidad. Con el fin de conseguir un producto atractivo, hemos pedido al cliente una exposición de sus gustos. Con esto podremos intuir el tipo de diseño más atractivo desde el punto de vista del cliente y al mismo tiempo buscaremos siempre un diseño funcional. Si el cliente no fuera capaz de comunicar claramente sus preferencias, una solución podría ser preguntarle por páginas que están activas, conoce y le gustan, así podremos captar las ideas que intenta transmitirnos.

En esta primera capa de presentación, se ha optado por la tecnología JSF. Esta tecnología nos genera un código muy limpio y modular, no siendo necesarios grandes trozos de código (propensos a acumular errores) asociados a tecnologías como PHP, ASP o JSP, entre otras. Gracias a la programación modular, hemos sido capaces de dividir en módulos nuestro programa, haciéndolo más legible y manejable. De esta forma, hemos podido reutilizar código y esto es una buena práctica en programación.

3.1.2. Nivel lógico (Capa lógica de negocio)

En la capa de negocio se encuentran todos los elementos de la aplicación que pueden ser reutilizados. La capa de negocio (o capa intermedia) aísla la capa de presentación de la capa de servidor. Además, la estructura de datos y lógica que usa son independientes de la capa de presentación. Es necesario tener en cuenta el concepto de la capa de negocio, ya que los sistemas coexisten y, por tanto, habrá muchos factores externos relacionados con ellos. Estos factores evolucionarán y, en consecuencia, nuestro sistema deberá también evolucionar con ellos. El hecho de usar una arquitectura de tres capas e independizar cada una de ellas, hace mucho más sencilla cualquier modificación y adaptación en un futuro. Esto permitirá a nuestra aplicación ser más sencilla de mantener, siendo esto uno de los requisitos esenciales de la web.

La capa de lógica de negocio se ha implementado con Java básico por sencillez y porque es lo suficientemente potente por si sólo para tal implementación. Además, utilizar Java en esta capa, la cual es la que más modificaciones sufre en realidad, nos permite prever los problemas que puedan surgir en la ejecución. Dentro de esta capa podemos distinguir dos subcapas. Por un lado, la capa que controla la presentación (clases bean) y, por otro lado, la que hace de *Direct Access Object*(DAO),esto es, clases de utilidad para la recuperación y almacenamiento de datos. Esto nos ha permitido hacer una representación más centrada en la integración entre aplicaciones usando Servlets, Web services y EJBs, entre otros.

3.1.3. Nivel de persistencia (Capa de base de datos)

El concepto de persistencia de objetos de datos, que de forma abstracta se representa por una capa más en la arquitectura de las aplicaciones, surge de la necesidad de vincular objetos de bases de datos relacionales a objetos Java para su manejo en las aplicaciones. El marco de desarrollo propuesto por la plataforma Java, JPA, y cuya implementación más extendida es Hibernate, es el que se ha utilizado en el proyecto para este nivel. Esta capa de persistencia nos ha permitido abstraernos al programar en Java de las particularidades de una determinada base de datos. También nos ha proveído con clases Java que envolverán (patrón de diseño Wrapping) los datos recuperados de los registros de las tablas.

Con Hibernate es posible construir varios ficheros independientes que relacionen bases de datos diferentes con las mismas clases Java según la base de datos que estemos usando en ese momento, de esta manera, nuestro mapeo de las clases java con las tablas de bases de datos podría existir antes incluso de que dichas tablas hubieran sido creadas. También nos permite cambiar los nombres de las tablas de manera muy sencilla sin tener que rehacer todas las consultas en nuestra aplicación. Además, una de las principales características, desde un punto de vista de programador, es que con Hibernate no emplearemos habitualmente SQL para acceder a datos. En lugar de esto, el propio motor de Hibernate, mediante el uso de factorías (patrón de diseño Factory) y

otros elementos de programación, construirá estas consultas por nosotros. Esto es muy útil para este proyecto, ya que nuestro cliente no desea ejecutar grandes consultas, pero sí que desea un fácil mantenimiento. Esta tecnología permite no hacer múltiples consultas SQL, sólo serán requeridas en casos muy concretos. Por otro lado, cabe mencionar que como motor de base de datos hemos utilizado MySQL [7] por ser una de las bases de datos más utilizadas en el mundo junto con Oracle.

3.2. Tecnologías usadas en la implementación

En este proyecto, la mayor dificultad ha residido a la hora de llevar a cabola implementación. Las bases de datos y las relaciones entre ellas no suponen grandes complicaciones para el desarrollo. Atendiendo a que el cliente desea ser lo más flexible posible sin invertir grandes cantidades de tiempo y no desea un sistema muy complejo que requiera un elevado aprendizaje inicial, se ha optado por un desarrollo en Java básico.

Tras considerar varias opciones, se ha optado por desarrollar una aplicación Java desde el principio que pudiera ser mantenida fácilmente y que permitiera trabajar con tecnologías en auge en los últimos años. En este proyecto hemos explotado la gran cantidad de herramientas que ofrecen estas nuevas tecnologías. Esto ha permitido adquirir un mayor conocimiento de las tecnologías actuales, que se ha materializado en el desarrollo de una aplicación muy rica en funcionalidad. Además, como hemos priorizado las tecnologías basadas en software libre, podremos utilizar cualquier herramienta existente en el mercado para dichas tecnologías.

Una posible solución para nuestro cliente hubiera sido una web tipo Joomla[8] donde se pueden definir zonas de trabajo, módulos y otras cosas. También se hubiera sido posible hacer la web con Liferay[9], donde las pantallas y los campos, así como los documentos e imágenes, habrían podido ser mantenidas directamente desde la propia web con un perfil de administrador. Sin embargo, finalmente, descartamos estas opciones.

A continuación se presentan las tecnologías que se han seleccionado para el desarrollo del proyecto y las razones por las que las hemos escogido. Expondremos sus principales ventajas e inconvenientes. Además, al haber profundizado en el aprendizaje de estas tecnologías, hemos extraído conclusiones mediante la práctica de programación.

3.2.1. PHP: Hypertext Preprocessor(PHP) y Java Server Pages(JSP)

Las tecnologías PHP: Hypertext Preprocessor (PHP), y Java Server Pages (JSP) son muy similares en la manera de funcionar y la principal diferencia entre ellas es el lenguaje de programación.

Ventajas	**Inconvenientes**
• Fácil aprendizaje y uso. • Lenguaje muy rápido. • Soporta la orientación a objetos. • Lenguaje multiplataforma. • Capacidad de conexión a la mayoría de gestores de bases de datos. • Posee documentación en su página oficial la cual incluye descripción y ejemplos de cada una de sus funciones. • Es libre, por lo que se presenta como una alternativa de fácil acceso para todos. • Incluye gran cantidad de funciones. • No requiere definición de tipos de variables ni manejo detallado del bajo nivel. • No requiere de una configuración excesiva para crear una página con esta tecnología. • Todo está centralizado en un mismo sitio por lo cual tenemos la certeza de que nada influye fuera de la propia página.	• Se necesita instalar un servidor web. • Todo el trabajo lo realiza el servidor y no delega al cliente. Por tanto puede ser más ineficiente a medida que las solicitudes aumenten de número. • La legibilidad del código puede verse afectada al mezclar sentencias HTML y PHP. • La programación orientada a objetos es aún muy deficiente para aplicaciones grandes. • Dificulta la modularidad. • Dificulta la organización por capas de la aplicación. • El mantenimiento de las páginas se complica si el tamaño aumenta. • Se entremezclan los códigos de lógica y presentación accediendo ambos al mismo grupo de variables locales. • El código generado es directamente HTML lo cual a veces hace que ciertos trozos de código se repitan muchas veces, aunque se pueden definir componentes. • Incluir AJAX en estas tecnologías se vuelve una labor complicada.

Tabla3.1. Ventajas e inconvenientes de PHP y JSP.

Ambas tecnologías llevan muchos años en el mercado y, en consecuencia, el mantenimiento de este tipo de aplicaciones puede complicarse. En la Tabla 3.1 exponemos sus principales ventajas e inconvenientes.

En la práctica, si trabajamos con estas tecnologías, es habitual que el código lógico se entremezcle con el código HTML, puesto que ambos códigos se utilizan en la misma página y se cambia de uno al otro mediante el uso de unos *tags* especiales. Usando estas tecnologías podemos llegar a tener la funcionalidad muy dispersa por la página. Sin embargo, si la implementación se hace intentando agrupar todos los bloques de recuperación de datos podemos llegar a tener dos bloques. Un primer bloque se encarga de rellenar una serie de indicadores de visibilidad, permisos o acciones y enviarlo hacia un segundo bloque cuyo objetivo es generar el HTML necesario para componer todo lo que estos indicadores significan como pueden ser listas de elementos, campos, botoneras o códigos de texto, entre otros. Estas tecnologías proporcionan mucha libertad a la hora de implementar prácticamente cualquier cosa puesto que estas tecnologías no generan ningún tipo de etiqueta HTML por su cuenta. Esto, en ocasiones, también es un problema puesto que al final los componentes tienden a repetirse y, con ellos, la manera de hacerlos funcionar.

3.2.2. Java Server Faces (JSF)

La tecnología JSF nació en el año 2004, pero no fue hasta 2006 cuando comenzó a ser popular al integrarse con Java EE 5. Actualmente, la última versión disponible de JSF es la versión 2.1, la cual ha cambiado considerablemente desde 2006. JSF difiere en gran medida de los desarrollos con PHP o JSP. Para empezar se suele separar en dos grupos de archivos, los que contienen la presentación de la página propiamente dicha (xhtml), y los que contienen la lógica de presentación (Java). En los archivos xhtml (*.xhtml) se usan componentes de JSF que gestionan todos los formularios, campos, tablas de datos, así como los estilos y javascript para los mismos. Mientras tanto, en los archivos java (*.java) configurados, se realizan las peticiones a base de datos, el procesado de la información o el registro de acciones, entre otros. En la Tabla 3.2 se muestran las ventajas e inconvenientes de JSF.

Esta tecnología genera una división física entre la parte puramente estética y la parte puramente lógica que favorece notablemente la legibilidad del código y el mantenimiento del mismo, haciendo su aprendizaje mucho más sencillo. Algunas implementaciones en JSF permite tener utilidades muy atractivas como campos para subir listas de documentos, campos ‘drag&drop’ o tablas expandibles y editables solamente añadiendo un par de líneas de código. Cabe destacar, además, que la mayoría de herramientas de JSF en el mercado son OpenSource, por tanto pueden usarse de manera libre y sin licencia. Por otro lado, tenemos la posibilidad de editarlas y modificarlas a nuestras necesidades. Por estas razones su uso se está incrementando

notablemente en los últimos años, dejando en una minoría los desarrollos en tecnologías como ASP.NET que es de pago.

Ventajas	Inconvenientes
• Debido al amplio repertorio de implementaciones o de componentes, JSF permite optimizar mucho el código. • Se divide físicamente la parte de presentación y lógica para la presentación. • Permite utilizar Ajax de manera integrada creando navegaciones muy flexibles • Ofrece gran cantidad de componentes Opensource. • Los APIs de la tecnología JavaServer Faces se han creado directamente sobre el API JavaServlet. Esto permite hacer algunas cosas interesantes como usar otra tecnología de presentación junto a JSP como crear componentes propios personalizados. • Ofrece una rápida adaptación para nuevos desarrolladores • Proporciona una rica arquitectura para manejar el estado de los componentes, procesar datos, validar la entrada del usuario y manejar eventos	• Cada implementación de JSF potencia una manera de trabajar permitiendo definir una estrategia de implementación efectiva. • Requiere de buen aprendizaje. La integración con otros lenguajes como JavaScript o JQuery complica más la tecnología. • Su naturaleza como estándar hace que la evolución de JSF no sea tan rápida como pueda ser la de otros entornos como WebWork, Wicket, Spring, etc

Tabla 3.2. Ventajas e inconvenientes de JSF

3.2.3. Enterprise Java Beans (EJB)

La tecnología Enterprise Java Beans (EJB) se suele usar como una capa de separación y/o abstracción entre la lógica de la presentación y las instrucciones hacia la base de datos, evitando así complicaciones en la conexión con la base de datos. En la Tabla 3.3 hemos detallado algunas de las ventajas e inconvenientes de esta tecnología.

Ventajas	Inconvenientes
• Permite conocer fácilmente que acciones se ejecutan contra la base de datos aunque no se puede conocer la frecuencia 'a priori'. • Permite cambiar la configuración de persistencia y el sistema seguiría funcionando igual si se usa como una interfaz de acciones contra la base de datos. • Un EJB puede llamar a otros EJB en serie permitiendo la interoperabilidad entre aplicaciones.	• Es una tecnología muy pensada para aplicaciones de cierta envergadura. El uso en pequeñas y medianas aplicaciones puede ser mas una complicación que una ayuda.

Tabla 3.3. Ventajas e inconvenientes de EJB

En la práctica, esta tecnología suele tener asociado un fichero de configuración con parámetros comunes y de uso cotidiano como, por ejemplo, el muestreo de tiempos de respuesta, la construcción de consultas o las transacciones. Generalmente, suele usarse para no olvidarse de operaciones tan importantes como 'Commit' o 'Rollback'en las transacciones. También es muy útil el hecho de que permite hacer ejecuciones remotas, lo cual es muy útil a nivel de reutilización cuando existen dos aplicaciones trabajando sobre una misma base de datos, algo muy habitual en aplicaciones protegidas contra fallos.

3.2.4. Java Persistence Api (JPA)

La tecnología Java Persistence Api(JPA) contiene un conjunto de anotaciones, clases y métodos para trabajar con la base de datos usando las funcionalidades de Java. Además de esto, también define un lenguaje de ejecución de consultas propio llamado JPQL cuya diferencia más notoria con SQL es que usa nombres de clases en lugar de nombres de tablas y nombres de atributos en lugar de nombres de columnas. En la Tabla 3.4 hemos resumido algunas de las ventajas e inconvenientes del uso de JPA.

Ventajas	Inconvenientes
• Requiere de una configuración muy básica para comenzar. • Es de gran utilidad la inserción y borrado de datos sobre una base de datos sin generar ningún tipo de consulta. • No se requiere recuperar un ResultSet y procesarlo para asignar los valores de las tablas a los diferentes campos etc. Esto queda implícito al uso de JPA y, por tanto, oculto, basta con acceder a las propiedades que se requieran. • Permite reemplazar una librería por otra o cambiar de versión sin tener que tocar nada de código si se requiere por problemas de rendimiento u otros. • JPA no solo está vinculado a JPQL sino que también permite ejecutar consultas SQL normales llamadas consultas nativas.	• Se requiere una configuración correcta, ya que de lo contrario habría grandes problemas en la configuración de entidades (tablas) que podrían obtener muchos datos indeseados. • El uso de subconsultas con JPQL es un poco complejo de utilizar y en muchos casos no se resuelven problemas que con SQL tendrían fácil solución.

Tabla 3.4. Ventajas e inconvenientes de JPA

Una de las principales características de JPA es que las relaciones entre diferentes tablas se pueden modelar con clases o listas de clases haciendo que el acceso a objetos relacionados sea muy fácil. Esto no siempre es una ventaja pues, si no se configura correctamente el entorno, puede hacer que el acceso a una clase sencilla cree muchas consultas relacionadas e inútiles penalizando el acceso al registro concreto. Por otro lado, otra características es la posibilidad de mantener una tabla sin crear ninguna consulta específica puesto que JPA tiene métodos para hacer consultas autogeneradas de uso muy extendido como son, por ejemplo, la búsqueda de todos los elementos, la búsqueda de un elemento concreto o la creación, edición o borrado de un registro.

Sin embargo, JPA no es una utilidad práctica para la implementación, si no una utilidad pensada para definir un grupo de reglas a seguir. Sin embargo hay tecnologías que implementan esta funcionalidad básica y la extienden como es el caso de Hibernate, OpenJPA, EclipseLink, y otras más. Realmente, lo que importamos al proyecto son las librería de una de estas tecnologías.

Ventajas	Inconvenientes
• MySQL software es Open Source. Licencia GPL. • Buen rendimiento, velocidad en las operaciones. • Facilidad de configuración e instalación. Soporta gran variedad de Sistemas Operativos. • Baja probabilidad de corromper datos. • Proporciona una base sólida a la hora de programar aplicaciones pues su uso está muy extendido en todo el mundo • Contiene un tipo de dato llamado Autoincrement que funciona como una secuencia automática.	• Un gran porcentaje de las utilidades de MySQL no están documentadas.

Fig. 3.5. Ventajas e inconvenientes de MSQL

3.2.5. MySQL

MySQL es uno de los gestores de bases de datos relacionales más usado hoy en día [6]. Tiene también una aplicación llamada MySQL workbench que facilita la configuración de bases de datos. Esta tecnología contiene todas las funcionalidades que una buena aplicación puede necesitar y está disponible para instalarse, prácticamente, en cualquier sistema operativo. Por otro lado, también dispone de librerías para poder ser consultado desde cualquier lenguaje de programación y, además, su uso es Gratis.

Hemos resumido algunas de las ventajas e inconvenientes que tiene la tecnología MSQL en la Tabla 3.5.

Ventajas	Inconvenientes
• Al ser muy sencillo de utilizar, pueden usarse en cualquier aplicación cualquier punto con conexión a Internet como móviles o tablets. • Tiene gran variedad de oportunidades para explotarse en casi cualquier tipo de desarrollo web. • Su uso puede ser transparente si se desea y esto es de gran utilidad.	• Carece del concepto de 'Sesión' o de 'Aplicación'. La mayor parte de los usos de los Servlets se producen a nivel de 'Request', no obstante, podrían implementarse para tener en cuenta estos conceptos más generales. • Al ser la integración en ocasiones transparente, podemos estar ejecutando código que realmente no queremos que se ejecute. Según la aplicación, despreciar errores mínimos podría ser un problema.

Tabla 3.6. Ventajas e inconvenientes de Servlets Java

3.2.6. Servlets Java

En los desarrollos web es muy común el uso de Servlets Java. Estas clases suelen ser usadas para atender peticiones que se realicen desde una URL. Los Servlets pueden tener muchos usos en la práctica y están presentes en muchos desarrollos de manera muy integrada. Sin ir más lejos, toda la implementación de JSF se basa en que todas las peticiones sobre páginas JSF se redireccionan a un Servlet que nos da soporte, según la petición, del elemento que necesitemos. Esta funcionalidad es muy importante y cada día se usa más a menudo pues te permite tener el código separado en varias clases que son el verdadero núcleo del sistema. El Servlet solamente sirve como repartidor de peticiones a las diferentes clases.

Por otro lado, los Servlets no solo sirven para separar código, tienen otros muchos usos. Suponiendo que disponemos de cierta funcionalidad que se ejecuta en determinado momento en segundo plano, un Servlet podría hacer las funciones de dashboard o panel de control para conocer el estado. Si fuera necesario que dos aplicaciones se comuniquen y compartan información entre sí, podríamos, de manera muy sencilla y definiendo unas pocas reglas, diseñar un sistema de transmisión de datos de una aplicación a otra.

Hay muchos usos que se pueden hacer sobre los Servlets, es una tecnología muy útil porque su funcionamiento es sencillo y se puede usar desde cualquier punto que tenga una conexión a internet, sin necesidad de definir una estructura y sin necesidad de librerías, basta con el mismo lenguaje Java de siempre. Hemos definido algunas de sus ventajas en inconvenientes en la Tabla 3.6.

3.3. Herramientas de desarrollo

Ha sido imprescindible para el desarrollo del proyecto elegir las herramientas adecuadas, ya que permiten optimizar los tiempos de desarrollo. Las herramientas que se han seleccionado cubren las necesidades que tenemos para llevar a cabo un desarrollo de calidad y representan una pequeña parte de todas las herramientas que existen actualmente en el mercado. A continuación, se detallan las herramientas que se han usado, explicitando para qué y por qué han sido útiles.

3.3.1. Balsamiq Mockups

Puesto que con frecuencia el cliente no conoce con detalle lo que realmente quiere, máxime sin haber visto una implementación con datos reales, es de gran utilidad hacer diseños no-funcionales. Para ello se emplean herramientas que permiten el diseño de bocetos. Nosotros hemos optado por usar Balsamiq Mockups, ya que es una herramienta conocida por haberla utilizado en otros proyectos y que se ajustaba perfectamente a nuestras necesidades.

3.3.2. Eclipse (Versión Indigo)

Eclipse es uno de los entornos de desarrollo más utilizados en todo el mundo. Una de las grandes ventajas de Eclipse es su fácil integración con servidores de aplicaciones y esto permite lanzar el programa desde el propio entorno de desarrollo. Sin embargo, al tratarse de una herramienta tan potente, Eclipse requiere bastantes recursos. Para conseguir un funcionamiento óptimo con las versiones más actuales, se suele necesitar mucha memoria RAM.

Debido a que Eclipse lleva muchos años en el mercado, disponemos de gran variedad de plugins preparados para diferentes tecnologías. La instalación de estos plugins es sencilla y, en las tecnologías habituales, los plugins suelen estar muy actualizados. Por otro lado, durante la programación en Eclipse podremos hacer uso de los puntos de interrupción, una utilidad muy valiosa que ayuda al programador a resolver errores. En cuanto a la versión de la herramienta, nos hemos decantando por la versión Indigo porque de las versiones de Eclipse que se podían integrar con todas las tecnologías que necesitábamos, ésta era la más actual.

3.3.3. Plugin de IceFaces para Eclipse

El plugin de IceFaces para Eclipse, distribuido por IceSoft, permite la utilización de software que tiene que ver específicamente con IceFaces. Este plugin permite crear un proyecto IceFaces (un proyecto JSF con la configuración para la librería de IceFaces) con parte del entorno configurado y preparado para cualquier implementación. Además, tiene una utilidad para crear proyectos sobre plataformas móviles con la versión de IceFaces para móvil.

3.3.4. MySQL Workbench

MySQL Workbench es una herramienta visual de diseño de bases de datos que integra desarrollo de software, administración de bases de datos, diseño de bases de datos, creación y mantenimiento para el sistema de base de datos MySQL.

Esta herramienta está disponible para Windows y permite desde diseñar un esquema de base de datos para MySQL hasta ejecutar consultas sobre un esquema ya existente. La herramienta permite hacer las operaciones más comunes sobre un esquema, como por ejemplo hacer claves ajenas, claves únicas, claves personalizadas, definición de columnas, roles o permisos, entre otros. Una vez diseñado un esquema esta herramienta también permite exportar todo lo hecho en una de las bases de datos enlazadas con la aplicación de manera que todos los cambios los tienes cuando quieras en la base de datos. Por otro lado, otra utilidad de gran ayuda es la recuperación de un esquema desde una base de datos ya existente, ya que de esta manera podemos recuperar la información de un esquema ya existente.

Además, MySQL Workbench dispone de una la utilidad de ejecución de consultas donde existe todo lo necesario para gestionar los datos de la base de datos y generar consultas. Este apartado es muy básico y sólo tiene funcionalidades muy concretas, no permitiendo lanzar aplicaciones más complejas como Toad, JDeveloper o Sql Navigator.

3.3.5. Thumbnailator

Thumbnailator es una biblioteca Java para la creación de miniaturas. Hemos elegido usar esta herramienta porque simplifica el proceso de cambio de tamaño de imágenes y la incorporación de marcas de agua o logos. Esto proporciona una interfaz fluida que reduce la cantidad de código necesario para lograr resultados. Además, crear thumbnails sobre imágenes reduce notablemente el tiempo de respuesta del servidor, ya que ha de enviar mucha menos información que si se enviasen imágenes completas. También permite que el espacio de memoria necesario para la aplicación sea mucho menor.

La herramienta se carga en nuestra aplicación en forma de librería. Al gestionar las imágenes de la aplicación, hemos usado esta herramienta para obtener una imagen de tamaño mínimo para acoplarla a un hueco dado. Esto ha supuesto una optimización

importante al ser el número de imágenes de la aplicación elevado, ya que con esta herramienta minimizamos los datos enviados desde el servidor hasta el cliente al redimensionar la imagen para acoplarla a un hueco concreto. Esta librería acepta los formatos de imagen más comunes como son JPG, PNG y GIF. Su uso es sencillo, ya que desde Java podemos generar una clase que actúa como generador. Sobre ésta se añaden propiedades y se ejecuta el método de generación para obtener la imagen en un directorio o directamente como un InputStream.

4. Implementación: estructura y contenidos de la web

En este capítulo, implementamos la web siguiendo el diseño obtenido. Tras finalizar las dos fases anteriores de especificaciones y de diseño, tenemos definida la estructura de la web así como su funcionalidad. Además, sabemos la tecnología y las herramientas que vamos a emplear para el desarrollo.

4.1. Configuración inicial

Comenzaremos esta fase con la configuración de un entorno que nos permita una programación de calidad y cómoda. Aunque podemos cambiar de entorno una vez hemos empezado con la implementación, no es recomendable que lo hagamos, ya que no es una tarea sencilla y puede llevarnos bastante tiempo. Además, la dificultad se incrementa si intentamos migrar un proyecto donde existen bastantes interacciones entre los diferentes componentes.

Al margen del entorno de programación, hemos de implantar e instalar nuestra aplicación en un entorno preparado para hacerlo funcionar durante todo su tiempo de vida. Por esta razón, es aconsejable cada cierto tiempo hacer una instalación en un entorno diferente del usado para el desarrollo y comprobar que todo funciona adecuadamente. Este trabajo puede hacerse de manera automática o semi-automática usando programas para generar casos de prueba que podemos lanzar cada cierto tiempo.

4.2. Presentación

Como ya hemos comentado anteriormente, en la capa de presentación de una aplicación basada en la estructura de tres capas se intenta centrar toda la funcionalidad en el código. Por lo tanto, podemos decidir en esta capa qué elementos son visibles, para qué usuarios y cómo estos elementos se muestran en la pantalla. Para implementar esta parte de la aplicación hemos usado la tecnología JSF. Esta tecnología está muy ligada con la capa de presentación pero también está muy ligada a la capa de lógica de negocio. Dependiendo de la manera de utilizar JSF, esta relación entre las capas puede ser más o menos fuerte.

Podemos distinguir la capa de presentación y la capa lógica por un determinado grupo o tipo de archivos. Mientras que la capa de presentación se suele concentrar en los archivos xhtml, la capa de lógica de negocio se suele concentrar en los archivos java. Naturalmente, la división entre estas capas no solo depende del grupo de archivos. De hecho, en las clases Java podemos añadir código sobre la presentación y en los archivos xhtml se puede añadir código sobre la lógica de negocio. La forma en que agruparemos los diferentes archivos dependerá de las necesidades y de la manera en que el programador decide organizar su proyecto.

Por otro lado, explicaremos brevemente cómo funciona la tecnología JSF para que puedan entenderse algunos ejemplos que se expondrán más adelante en esta memoria. Por un lado, la configuración de JSF se basa sobre un único y gran Servlet que se configura en el archivo xml de nuestra aplicación. Este servlet que se suele llamar FacesServlet y actúa a modo de 'Despachador' o 'Repartidor de tareas', cuya principal función es enviar toda la información que recibe a donde se requiera. Esta funcionalidad nos permite separar cómo se envía la información al hacer un Submit de una página. Por otro lado, para enlazar JSF con el código de nuestra aplicación usamos lo que se conoce como Managed Bean. Estas clases sirven como controladores de la información de una página en un momento dado y suelen tener métodos para el acceso de variables y métodos para la ejecución de acciones que tienen en cuenta los diferentes estados de una página, las acciones que se pueden generar o las variables afectadas. Estas clases, componen, junto con los archivos de presentación de la página, la base sobre la cual podemos trabajar con la tecnología JSF.

Como podemos ver en la Fig. 4.1, JSF permite abstraerse de varios de los problemas más comunes a la hora de implementar una página web. Sin embargo, también ofrece cierta libertad aunque existan procesos generales predefinidos como validaciones muy usadas o conversiones de tipos de datos base. Por otro lado, permite un alto grado de configuración al permitir diseñar y administrar nuestros propios componentes y funcionalidades tales como validadores de campos extendidos o convertidores de entidades complejas.

Un funcionamiento básico de JSF de ejemplo sería el siguiente. Por ejemplo, un cliente intenta entrar en una página y hace una petición al servidor web. El servidor web consigue mapear la ruta de la página que el cliente está pidiendo con el servlet de JSF y delega toda la recuperación de esta página al Faces Servlet. Este servlet recupera la página y procesa toda la información que necesita invocando a los métodos get o set, asociado a las variables del ManagedBean o al valor que necesitamos recuperar.

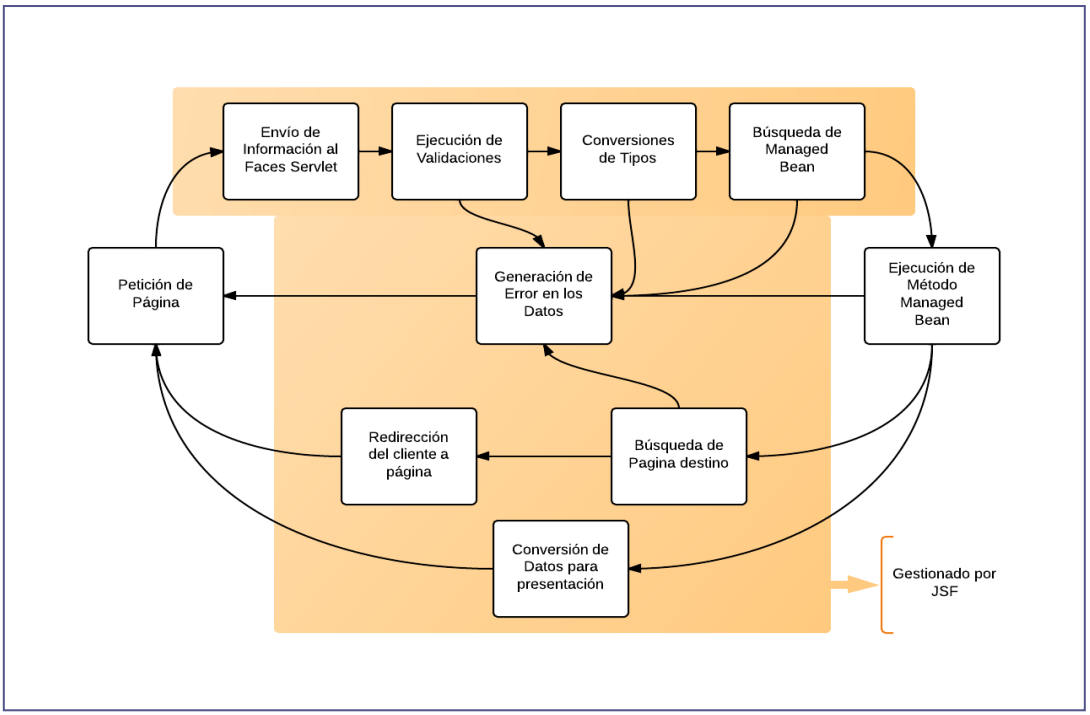

Fig. 4.1. Esquema del funcionamiento básico de JSF.

Un ejemplo de recuperación de atributos sería de cómo sigue. Por ejemplo, supongamos que tenemos el elemento '#{myBean.atributo}' en algún punto de la página que vamos a mostrar, pues bien, el motor de JSF intenta recuperar o instanciar el ManagedBean asociado al nombre 'myBean' que, por nomenclatura, se suele llamar MyBean.java. Cuando hemos recuperado la clase JSF invocaría al método 'getAtributo()' que deberá existir en la clase MyBean.java. Esta manera de funcionar sirve como base para ejecutar o recuperar la gran mayoría de elementos de un Managed Bean y es muy útil porque, usando un par de métodos que suelen existir por defecto, se puede integrar JSF con entidades complejas obtenidas de otras tecnologías.

A la hora de ejecutar métodos se usa la misma manera de funcionar pero colocando el nombre completo del método sin parámetros. Aunque los parámetros suelen utilizarse mucho según JSF, no vale la pena enviar más información de la exclusivamente obligatoria al cliente para que luego el cliente la envíe de vuelta. Es mejor enviar al cliente un Id, un único parámetro para identificar una entidad compleja o para acceder a un diccionario de datos.

Una vez hemos visto muy básicamente en qué consiste JSF veremos también más adelante que limitaciones nos impone y cómo podemos evadirlas o rediseñar algunos elementos para adaptarlas a nuestras funcionalidades. Como explicar toda la implementación de la capa de JSF puede llegar a ser muy tediosa vamos a intentar resumir toda esta parte en los puntos más importantes que se consideraron de alto riesgo en el desarrollo por su alto grado de complejidad, o su reutilización/usabilidad de cara a un futuro proyecto o a una segunda versión más completa de este mismo proyecto.

4.2.1. GridUtil

Uno de los problemas que hemos tenido a la hora de implementar nuestra aplicación es que el cliente deseaba que tuviéramos un listado de fotografías que al ir incrementándose saltase de línea y continuara en una línea más abajo sin ocupar toda la línea, solamente la celda que necesitase. En la siguiente Fig. 4.2, puede verse la idea de lo que tenemos (antes) y de lo que queremos conseguir (después). De esta visualizaremos a las diferentes personas que componen el grupo de investigación a modo de tabla, en vez de un clásico y típico listado, ganando así mucho más espacio, ya que buscamos que se vea bastante información en el menor espacio posible.

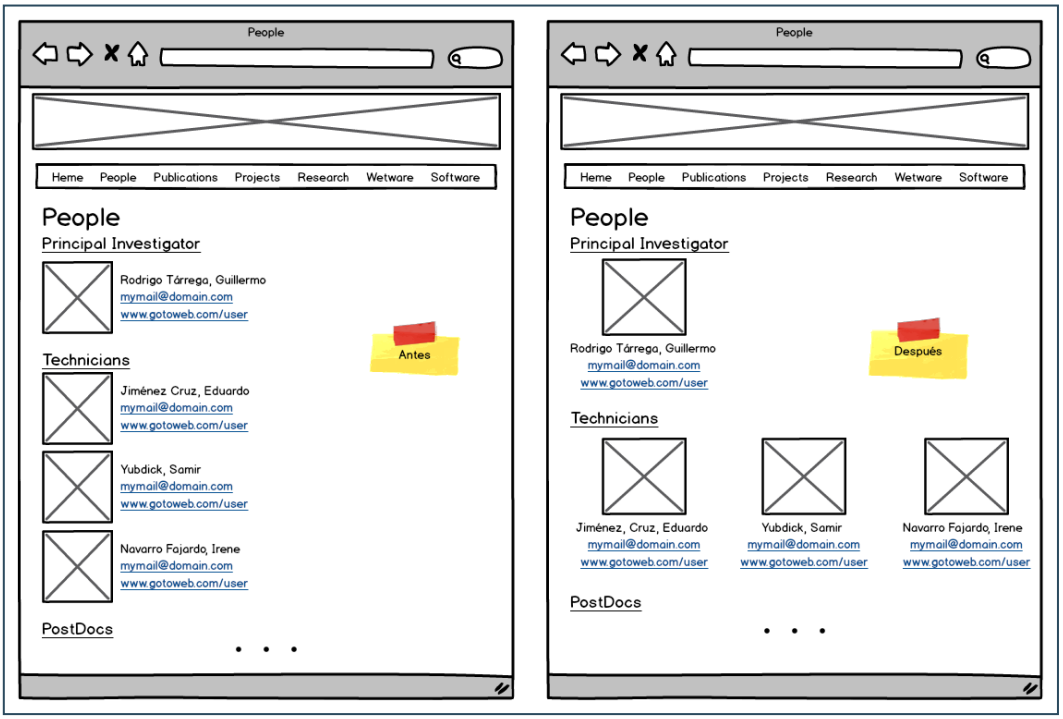

Fig. 4.2. Visualización del contenido en la pantallaPeople.

Conseguir esto suponía un reto ya que ningún componente de los disponibles nos proporcionaba esta funcionalidad. Por tanto, se ha requerido modelar este comportamiento a partir de otros componentes e ir probando a unir diferentes maneras hasta que todo funcionase, compactando el sistema y haciéndolo lo más reutilizable posible.

Para diseñar este comportamiento nos hemos basado en una tabla de datos. Las tablas de datos son muy utilizadas en cualquier aplicación cuyo propósito sea explotar una base de datos o mostrando una información determinada. Las tablas las encontramos en muchos contenidos webs y su comportamiento es muy intuitivo.

Normalmente una tabla de datos se usa para mostrar una lista de elementos con muchos atributos mostrando por cada fila un elemento diferente y agrupando los atributos en las columnas. Esta manera de funcionar se parece mucho a cuando intentamos hacer una representación en tabla de una consulta SQL, en cada fila elementos diferentes, en cada columna los mismos atributos.

Gracias a la librería de JSF que hemos utilizado en este proyecto, podemos dar valores a una tabla de datos usando unas clases propias de IceFaces que nos permiten diseñar un modelo de datos para nuestra tabla. Este modelo (ver Fig. 4.3) nos permite romper con el modelo típico de un objeto por columna y nos permite añadir varios elementos en una misma fila. Esto a su vez nos fuerza a recalcular manualmente el número de columnas disponible todo en función de una lista de Usuarios.

Sin embargo, al usar el modelo de Datos de JSF tenemos un problema con la recuperación de cada elemento en cada celda ya que no tenemos un acceso directo a cada elemento y nos vemos obligados a utilizar un recuperador que vaya obteniendo los

```
public GridUtil<E> initialize(List<E> elementList){
    currentCol = 0; //Iniciamos/Reiniciamos el flag de columna

    rows = elementList.size() / numColumns;
    int lastRowColumns = elementList.size() % numColumns;
    if (lastRowColumns != 0) { rows ++; } // A◆adimos una fila mas si nos falta hueco aun

    colDataModelList = new ArrayList<DataModel<Integer>>(rows);
    dataMap = new HashMap<Integer, E>();

    List<Integer> rowList = new ArrayList<Integer>(rows);
    List<Integer> colList;
    int columns = numColumns;

    Iterator<E> it = elementList.iterator();

    for(int i = 0; i < rows; i++){
        rowList.add(i);
        colList = new ArrayList<Integer>(numColumns);

        // Si llegamos a la ▯ltima fila y nos quedan menos elementos
        // de los que se necesitarian para rellenar una fila
        if(i+1 == rows && lastRowColumns != 0){ columns = lastRowColumns; }

        for(int j = 0; j < columns && it.hasNext(); j++){
            colList.add(j);
            dataMap.put((numColumns*i)+j, it.next());
        }
        colDataModelList.add(new ListDataModel<Integer>(colList));
    }
    rowDataModel = new ListDataModel<Integer>(rowList);

    return this;
}
```

Fig. 4.3. Implementación de una tabla para visualizar grupos en la pantalla People

datos de cada celda. El método de la Fig. 4.4 explota una característica del modelo de datos que nos proporciona JSF para recuperar la fila y la columna actual y calcula el elemento que se asignaría a esa celda:

```
public E getElement(){
    DataModel<Integer> columnDataModel = colDataModelList.get(currentCol-1);
    if (rowDataModel.isRowAvailable() && columnDataModel.isRowAvailable()) {
        // get the index of the row and column for this cell
        int row = rowDataModel.getRowData();
        int col = columnDataModel.getRowData();
        return dataMap.get((numColumns*row)+col);
    }
    return null;
}
```

Fig. 4.4. Método para recuperar el objeto y posicionarlo en la tabla.

Usando este método conseguimos generar un modelo de filas y columnas que en presentación no se ve pero sí que nos permite acceder a un elemento genérico para mostrar y visualizar. El hecho de usar Genericidad en esta clase es para poder reutilizar esta funcionalidad, si nos hiciera falta, en cualquier parte de este proyecto o en cualquier proyecto si migramos estos componentes.

Si esta funcionalidad llegara a ser muy utilizada podría llegar a ser un componente por si mismo adaptándolo y convirtiéndolo a un proyecto para crear un componente de JSF. En este caso no se ha hecho porque antes de llegar a ser un componente hay que probar bien que funciona bajo ciertas condiciones y que no va a suponer un problema para el desarrollo de una futura página web.

4.2.2. Publications

En la pantalla de *Publications* acordamos desarrollar unos paneles, que pudieran estar o no desplegados y que permitieran agrupar las publicaciones del cliente por años. El primer panel mostraría las publicaciones del último año y, en orden decreciente, encontraríamos un listado de los años anteriores con sus respectivos artículos relaciones. Por otra parte, el cliente quiere dar una especial importancia a los últimos artículos publicados por lo que el último año será el único grupo que muestre todos los artículos publicados en este año y ordenados por fecha decreciente.

Este requisito ha supuesto un reto porque necesitaríamos una lista de años con una cabecera, el estado de estar o no visible y una lista de publicaciones. Por si misma, la información no parece complicada de manejar pero se complica cuando intentamos mostrar toda esta información en pantalla.

```
<h:panelGrid id="content">
    <h:outputText value="Publications" styleClass="bigTitle"/>
    <ice:form>
        <ice:panelCollapsible expanded="true" >
            <f:facet name="header">..
            <ice:dataTable value="#{publicationBean.mainPanel.list}" var="item">..
        </ice:panelCollapsible>
        <ui:repeat value="#{publicationBean.iterableList}" var="element">
            <ice:panelCollapsible expanded="true">
                <f:facet name="header">..
                <ice:dataTable value="#{element.list}" var="item">..
            </ice:panelCollapsible>
        </ui:repeat>
        <h:panelGrid id="botonera" rendered="#{loginBean.admin}">..
    </ice:form>
</h:panelGrid>
```

Fig. 4.5.Definición de panelescorrespondientes a los años de publicaciones.

Como ya sabemos, estamos utilizando JSF para hacer la capa de presentación y con JSF podemos basarnos sobre componentes estándar y componentes propios de alguna librería de JSF. En este caso el listado lo mostraremos como una lista de paneles sin embargo, el comportamiento de estos paneles es diferente cuando se itera sobre ellos (Fig. 4.5).Da la impresión de que sobre los paneles no se pueden asignar expresiones para mostrar algunos y otros no, ya que toda la información solamente afecta al primer panel. Esto supone un problema en nuestro desarrollo ya que nos obliga a buscar una alternativa menos efectiva.

Para poder desarrollar esta forma de visualizar las publicaciones hemos dividido el listado en dos partes: en la primera, un panel visible simple y la otra, una lista de paneles que no sean visibles. Colocando uno detrás de otro podemos llegar a implementar el comportamiento que deseamos.

```
private void calculateLists() {
    List<String> dateList = publicationDao.getPublicationYears();
    pastYearsList = new ArrayList<PublicationBean.CollapsablePanel>(dateList.size());
    CollapsablePanel panel;
    panel = new CollapsablePanel();
    panel.setHeader(dateList.get(0));
    panel.setList(publicationDao.getPublicationByPubAnyo(dateList.get(0)));
    lastYearPanel = panel;
    for(int i = 1; i < dateList.size(); i++){
        String anyo = dateList.get(i);
        panel = new CollapsablePanel();
        panel.setHeader(anyo);
        panel.setList(publicationDao.getPublicationByPubAnyo(anyo));
        pastYearsList.add(panel);
    }
}
```

Fig. 4.6. Método que calcula número de paneles requeridos.

Para hacer esto funcionar lo que hemos de hacer es obtener la lista de elementos que vamos a necesitar e irlo procesando para obtener en un lado el panel visible y en otro la lista de paneles (Fig. 4.6).

4.2.3. Carrusel

Uno de los requisitos que hemos considerado que le aportaba una visibilidad atractiva a la página ha sido la inserción de un carrusel en la página de inicio, haciendo que diferentes imágenes pudieran hacerse visibles cada cierto tiempo, de esta forma aportamos cierto dinamismo a la web.

La tecnología JSF no proporciona ninguna herramienta que, en este caso concreto, nos fuera a ser especialmente útil por lo que hemos tenido que buscar alternativas. Existen librerías de JSF que sí que incluyen herramientas o componentes para crear y mantener carruseles. El hecho de no usar estas herramientas es porque hay que escoger las librerías de JSF teniendo en cuenta la compatibilidad entre ellas y como hacer que fueran compatibles entre sí podría llegar a suponer un problema y el uso de esta nueva librería iba a ser más limitado, ya que solo se iba a usar en esta página, hemos optado por otra opción: la librería de javascript JQuery.

La librería JQuery proporciona un lenguaje para trabajar con javascript de manera más sencilla y con menos declaraciones que con javascript puro. Todas y cada una de las operaciones que son posibles con JQuery son posibles desde javascript normal pero para llegar a generar el mismo comportamiento son requeridas líneas y líneas de código difícil de mantener, de utilizar y de probar.

En el caso del carrusel que queremos implementar, el comportamiento es muy sencillo. Para empezar tenemos una lista de imágenes y una zona de visualización. Lo que tenemos que hacer es mover la lista de imágenes bajo la zona de visualización de manera que los elementos se vayan moviendo como si fuera una lista circular.La Fig. 4.7 muestra lo único que necesitaríamos para hacer funcionar nuestro carrusel.

```
<div id="carousel">

    <div class="clear"></div>
    <div id="slides">
        <ul>
            <li><h:graphicImage library="images" name="Carrusel1.png"
            <li><h:graphicImage library="images" name="Carrusel2.png"
            <li><h:graphicImage library="images" name="Carrusel3.png"
            <li><h:graphicImage library="images" name="Carrusel4.png"
        </ul>
        <div class="clear"></div>
    </div>
</div>
```

Fig. 4.7. Especificación de las imágenes para el carrusel de inicio.

Tal y como puede observarse, existe un principal y otros dos anidados para que JQuery los utilice usando para ello las clases y los identificadores de los paneles. Sin embargo, para que el carrusel funcione debe ir acompañado de dos cosas: por un lado, el propio código para mover las imágenes y, por otro lado, una hoja de estilos para que las imágenes se comporten como queremos.

En la Fig. 4.8 podemos ver el código JQuery necesario para que las imágenes se puedan desplazar transcurrido un tiempo. Como nota importante, cabe destacar que este código lo que hace es modificar el estilo del panel principal para que la imagen que se muestra sea la que queremos.

La hoja de estilos es demasiado larga para mostrarla pero está disponible con el código fuente del proyecto. Como punto principal destacamos que el estilo del panel principal se adapta al sub-panel que contiene las imágenes y muestra una especie de modo ventana para que el panel inferior muestre las imágenes.

4.2.4. ImagesServlet

Muchas de los requisitos que el cliente nos pedía incluyen poder ver una imagen asociada a un cierto elemento en un listado. Esto nos ha permitido poder definir un servlet que funcione como un despachador de imágenes. Este servlet tiene como particularidad el hecho de que las imágenes se piden como si estuvieran en un directorio de ficheros de manera que este servlet procesa la ruta que recibe como parámetro e intenta devolver un fichero en formato imagen. Esto es útil en nuestro caso ya que separamos la localización de las imágenes de la base de datos sin llegar a ser totalmente independiente y podemos parametrizar y redefinir la localización de las imágenes solamente tocando este servlet y los parámetros para el sistema de ficheros local.

Una de las funcionalidades que con este sistema es fácil de implementar puede ser, por ejemplo, el mostrar unas imágenes u otras en función del idioma de la aplicación sin tener que tocar el código del resto de la aplicación. Otra de las funcionalidades podría ser implementar un contador de imágenes visitadas y hacer estadísticas de cara a una posible explotación de fichas visitadas, etc.

La implementación de este servlet es sencilla si tenemos en cuenta que lo importante son las dos últimas partes de la ruta de acceso. En nuestro caso es lo único que necesitamos para acceder a un elemento ya que la primera parte nos dice de qué carpeta queremos sacar la información y el segundo nos da el identificador del elemento que queremos mostrar.

En caso de tener otros requisitos podríamos montar una criba para detectar accesos erróneos usando el identificador de sesión o una relación de roles y permisos pero como todas las imágenes son visibles desde las pantallas no hace falta diseñar todo este comportamiento aunque en caso de reutilizar esta funcionalidad debería tenerse en

cuenta y hacer las modificaciones oportunas. En la Fig. 4.9 podemos visualizar el código resultante de implementar este servlet.

Sobre este código cabe destacar dos cosas. En primer lugar, la ruta al archivo se encuentra siguiendo la misma estructura que la construcción de la URL salvo los parámetros de inicio. Y, en segundo lugar, la recuperación del fichero se ha hecho usando las clases de java directamente, esto puede ser positivo si deseamos un comportamiento alternativo. Además, ha sido muy testeado de modo que podríamos utilizar una librería para ayudarnos a recuperar el contenido a partir de un fichero.

```
// rotation speed and timer
var speed = 5000;
var run = setInterval('rotate()', speed);

// grab the width and calculate left value
var item_width = $('#slides li').outerWidth();
var left_value = item_width * (-1);

// move the last item before first item, just in case user click prev button
$('#slides li:first').before($('#slides li:last'));

// set the default item to the correct position
$('#slides ul').css({
    'left' : left_value
});

// if user clicked on prev button
$('#prev').click(function() {

    // get the right position
    var left_indent = parseInt($('#slides ul').css('left')) + item_width;

    // slide the item
    $('#slides ul:not(:animated)').animate({
        'left' : left_indent
    }, 400, function() {

        // move the last item and put it as first item
        $('#slides li:first').before($('#slides li:last'));

        // set the default item to correct position
        $('#slides ul').css({
            'left' : left_value
        });

    });

    // cancel the link behavior
    return false;

});
```

Fig. 4.8. Código en JQuery para generar el carrusel.

```
private void proces(HttpServletRequest req, HttpServletResponse res)
        throws ServletException, IOException {
    String[] url = req.getRequestURL().toString().split("/");
    String filePath = resourcePath;

    for(int i = 5; i < url.length; i++){
        filePath+=url[i];
        if(i != url.length-1){
            filePath+=File.separator;
        }
    }

    InputStream is = null;
    OutputStream os = null;
    try{
        is = new FileInputStream(filePath);
        os = res.getOutputStream();
        byte[] buff = new byte [1];
        while(is.available() > 0){
            is.read(buff);
            os.write(buff);
        }
        //Asignamos el ContentType, que viene por parametro
        res.setContentType(req.getParameter("cont"));
        os.close();
    }catch (Exception e) {
        e.printStackTrace();
    }finally{
        if(is != null){ is.close(); }
        if(os != null){ os.close(); }
    }
}
```

Fig. 4.9. Código del servlet para tratamiento de imágenes.

Esta manera de funcionar, según la versión de JSF se puede encontrar ya disponible para usar pero al ser un código sencillo de implementar según nuestros requisitos era preferible darle más peso a otras funcionalidades más críticas y añadir esta funcionalidad codificándola.

4.3. Lógica

La lógica de la aplicación, al usar JSF, se integra en gran medida con la capa de presentación, esto es consecuencia de que JSF funciona en dos capas, en la capa de presentación, y en la capa de lógica. Una capa necesita de la otra, son capas íntimamente dependientes. Como la parte de la presentación ya la hemos visto en el apartado anterior, vamos a hablar sobre las clases DAO (Direct Access Object – Clases de Acceso Directo de Objetos).

```
public List<String> getPublicationYears(){
    return getEntityManager().createQuery(" Select distinct p.pubAnyo from Publicat
}

@SuppressWarnings("unchecked")
public List<Publication> getPublicationByPubAnyo(String pubAnyo){
    Query query = getEntityManager().createQuery(" Select p from Publication p wher
    query.setParameter(1, pubAnyo);
    return query.getResultList();
}

public void persist(Publication r){
    EntityManager em = getEntityManager();
    EntityTransaction et = em.getTransaction();
    et.begin();
    try{
        em.persist(r);
        et.commit();
    }catch(Exception ex){
        et.rollback();
        ex.printStackTrace();
    }
}

public void merge(Publication r){
    EntityManager em = getEntityManager();
    EntityTransaction et = em.getTransaction();
    et.begin();
    try{
        em.merge(r);
        et.commit();
    }catch(Exception ex){
        et.rollback();
        ex.printStackTrace();
    }
}

public void destroy(Publication r){
    EntityManager em = getEntityManager();
```

Fig. 4.10. Clase DAO.

En esta aplicación hemos usado unas clases para dividir la lógica de la aplicación de la lógica de la persistencia para que así se puedan hacer modificaciones sobre la manera de interactuar con la base de datos y conocer dónde hacemos los accesos con la base de datos o cuantos accesos llegamos a hacer.

La idea es usar estas clases DAO como intermediarias añadiéndoles métodos para edición, creación y modificación como base mas algunos métodos que ejecutan consultas para buscar una lista de elementos según algún parámetro, ordenadas de alguna manera en concreto, o en general, según la SQL que nos haga falta en cualquier punto. En la Fig. 4.10, exponemos una clase DAO donde podremos visualizar que funcionalidad de las que nos aporta nos resulta de utilidad.

En este extracto de una clase DAO para *Publication*s vemos que, en este caso, y debido a que estamos usando Hibernate como motor de persistencia, todos los métodos para interactuar con la base de datos deben recuperar una instancia de la clase EntityManager de Hibernate y crear transacciones o lanzar consultas JPQL sobre la base de datos.

Esta es una buena manera de trabajar porque nos abstrae, en la parte de presentación, de toda la lógica que tiene que ver con la base de datos. Veamos en los

controladores como usar estos métodos, aunque son tan sencillos de utilizar como ejecutar el método asociado a la SQL que nos interesa recuperar con la lista de parámetros necesaria para ejecutarla (Fig. 4.11).

```
panel.setList(publicationDao.getPublicationByPubAnyo(anyo));
```

Fig. 4.11. Ejemplo de uso de un DAO.

4.4. Persistencia

Para hacer los cambios en base de datos se usa la tecnología JPA que permite reducir al mínimo las consultas o las SQL necesarias para mantener las diferentes tablas en nuestro sistema. Para poder utilizar JPA hemos de marcar con anotaciones o con un xml específico que columnas de las tablas de base de datos se corresponden con qué atributos de las clases que mapean estas clases. Una vez conseguido esto podremos utilizar una instancia de una clase para hacerla persistente en base de datos, actualizarla o hacer las acciones que normalmente se suelen hacer en una aplicación normal.

```
@Entity
@Table(name="SYG_PUBLICATION")
public class Publication extends ComonClass{

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    @Column(name="PUB_ID", nullable= false)
    private BigDecimal pubId;

    @Column(name="PUB_TITULO", length=400, nullable=false)
    private String pubTitulo;
```

Fig. 4.12. Ejemplo de clase JPA

La Fig. 4.12 muestra un ejemplo de una clase para que JPA pueda tratarla como una instancia a persistir. Como se puede ver, enlazar una tabla a una clase Java consiste en marcar la clase que nosotros queramos con la tabla a la cual referencia y los atributos de esta clase con las columnas de la tabla de base de datos. También, para evitar problemas sobre violación de claves ajenas o de claves primarias se le suelen definir anotaciones extra para marcar los campos que pueden, o no, estar vacíos.

```
EntityManager em = getEntityManager();
EntityTransaction et = em.getTransaction();
et.begin();
try{
    em.merge(r);
    et.commit();
}catch(Exception ex){
    et.rollback();
    ex.printStackTrace();
}
```

Fig. 4.13.Ejemplo de gestión de transacciones.

Una vez tenemos marcadas las clases JPA con las anotaciones básicas para que el motor de JPA que estemos usando las utilice como tal, podemos hacer instrucciones como la que se muestra en la Fig. 4.13.Como puede verse, la manera de hacer una actualización sobre una base de datos es tan sencilla como enviar el objeto hacia la base de datos para que sea persistente y confirmar los cambios. En este modelo concreto la transacción solamente se inicia ya que al hacer un commit o un rollback la transacción se cierra y por eso no hay una llamada especifica al et.close().

Existen varias maneras de gestionar las transacciones y son bastante dependientes de nuestro modelo de negocio, de la aplicación que estemos creando y de las tecnologías que estemos aplicando. En nuestro caso una transacción básica nos era suficiente ya que no hay clases que dependan unas de otras para poder ser persistidas ni tenemos una funcionalidad demasiado crítica como se pudiera implementar al hacer otro tipo de aplicaciones.

JPA no solo permite minimizar el número de consultas ejecutadas sobre base de datos sino que también permite abstraerse de qué base de datos se está utilizando al poder definir diferentes modelados sobre las mismas clases y un lenguaje de ejecución de consultas propio llamado JPQL que usa los nombres de las clases y los atributos de cada clase para ejecutar consultas.

El lenguaje de JPQL es muy parecido al lenguaje SQL con la salvedad de que, además de usar las clases en lugar de tablas, contiene expresiones que se regeneran al ejecutarse, para convertirse en consultas SQL nativas de la base de datos que estemos utilizando.

En el lenguaje JPQL podemos hacer uso de relaciones entre tablas sin ser relaciones directas. Estas relaciones se crean usando los tags propios para enlazar tablas mediante listas de valores. Este punto sobre JPA puede llegar a ser un poco confuso ya que en un escenario normal, lo que es común hacer es enlazar dos tablas mediante una condición. JPA, tal y como se muestra en la Fig. 4.14. En este aspecto es un poco más inteligente y enlaza las tablas usando los atributos de las clases.

```
FROM Author a INNER JOIN a.books b
```

Fig. 4.14. Ejemplo de enlace de tablas usando JPQL.

Para poder utilizar los atributos en las clases para enlazar tablas es necesario que existan relaciones del tipo @OneToMany, @ManyToOne o @ManyToMany. Estas relaciones permiten mapear todas las posibles relaciones de una base de datos relacional básica para ejecutar consultas complejas sobre objetos JPA sin necesidad de un intérprete específico para la base de datos.

No obstante, JPA también permite el uso de SQL directo contra la base de datos y será responsabilidad del programador que este código sea migrable entre bases de datos y entre versiones. Esta funcionalidad nos es útil a la hora de almacenar procesos que dependen de la base de datos en paquetes propios a la instancia de la base de datos que estamos manejando.

Cabe destacar que los objetos gestionados por JPA son fácilmente adaptables a JSF y, por regla general, si se han seguido un mínimo de buenas prácticas a la hora de generar estas clases, la integración es inmediata y bastante eficiente. Hay algunas herramientas que generan una estructura básica lo cual facilita aún más los primeros pasos con esta tecnología.

Además, JPA permite la no recuperación de ciertas relaciones o campos en el caso de que el usuario decida que son muy pesadas. Esto se usa añadiendo a algunas anotaciones la característica ‘fetch’. Esta característica tiene un gran impacto en cuestiones de optimización ya que si JPA no se entiende bien puede llegar a ser un lastre porque tiende a generar una gran cantidad de datos de la base de datos.

Así es como conseguimos integrar nuestra aplicación con el entorno persistente para que la aplicación se mantenga 100% funcional.

5. Post-implementación: evolución e implantación

Finalizada la implementación de la aplicación (programación), se procede en este capítulo a detallar la post-implementación de la misma, donde hay una interacción directa con el cliente a fin de ajustar contenidos y funciones. Una vez finalizado el desarrollo, se procede con la implantación de la aplicación en un servidor web real.

5.1. Evolución de la aplicación

Conforme el desarrollo va avanzando, es una buena práctica de cara a un desarrollo fructífero que se le vayan enseñando los diferentes módulos al cliente para que los valide y pueda visualizar un avance del proyecto.

En este desarrollo hemos puesto en práctica este concepto, entre otras cosas, porque disponíamos de una interacción con el cliente muy cercana y porque había una clara motivación por ofrecerle un producto de calidad. Esto nos ha permitido que todas las pantallas hayan evolucionado en mayor o menor medida desde su análisis inicial y que se ajusten a lo que el cliente esperaba. Este ha sido el caso de particular de la pantalla *People* o de *Publications*, que han evolucionado hasta conseguir la apariencia final y que, en consecuencia, creemos que ofrecen un resultado bastante atractivo.

En un primer momento, solo teníamos una pequeña información sobre qué era lo que el cliente deseaba en un primer momento: una lista de objetos donde poder poner una imagen opcional y unos cuantos campos para poder identificar cada publicación de manera única. A medida que se va pasando por cada una de las fases del proyecto vamos construyendo la estructura hasta visualizar un proyecto de calidad.

En las primeras versiones de la pantalla *Publications*, el contenido se visualizaba en forma de lista y simplemente se habían definido qué campos y cómo sería preciso mostrar, en función de la importancia. En la Fig. 5.1 podemos visualizar el prototipo que diseñamos para la versión inicial de la página. Este prototipo estaba pensado sobre todo para que el cliente validara el formato y comprobara si era necesario añadir nuevos campos o una funcionalidad adicional.

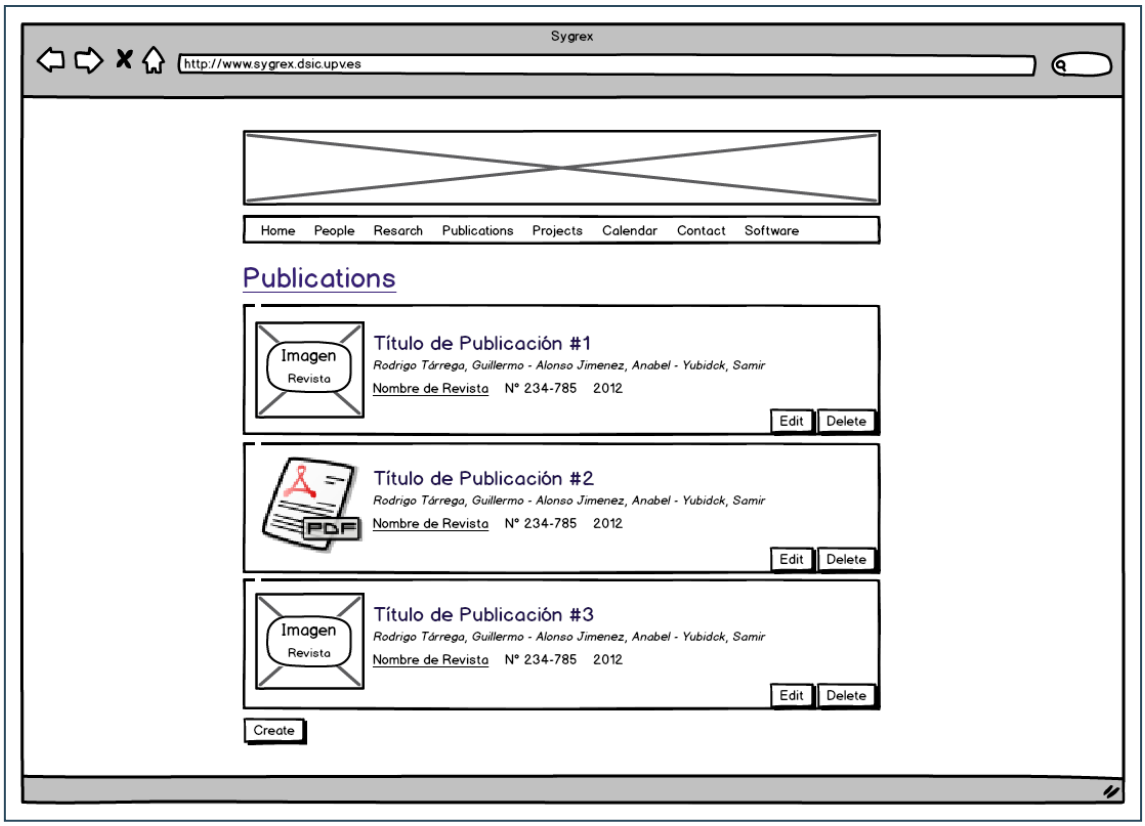

Fig. 5.1.Modelo inicial de la pantalla Publications.

Cuando el cliente pudo ver implementada la funcionalidad de este prototipo inicial tal y cómo la habíamos definido, sugirió la idea de organizar las diferentes publicaciones por año, desde la publicación más reciente hasta la publicación más antigua. De esta forma, se le daría más visibilidad al último trabajo realizado, que, en la mayoría de los casos, refleja las líneas actuales de investigación.

Por otro lado, nos solicitó que añadiéramos un enlace en el icono de la publicación, que podía ser una imagen o no, y que estaría relacionado con la publicación en sí. Este cambio, aunque no tenía una representación en las maquetas, fue otro de las mejoras que, finalmente, añadimos junto a la pantalla y cuyo boceto para validar le enviamos al cliente y que se ha detallado en la Fig. 5.2.

Conforme la base de datos fue rellenándose de información, volvimos a retomar el tema de las publicaciones ya que había mucha información, demasiada información divididas en demasiadas categorías de años. En este momento pensamos en cómo volver a darlesmás importancia a unas publicaciones frente a otras. Por supuesto todas las publicaciones debían de estar disponibles para ser consultadas en ese mismo instante por lo que hubimos de idear una manera de organizar la información mediante estilos, tablas y paneles.

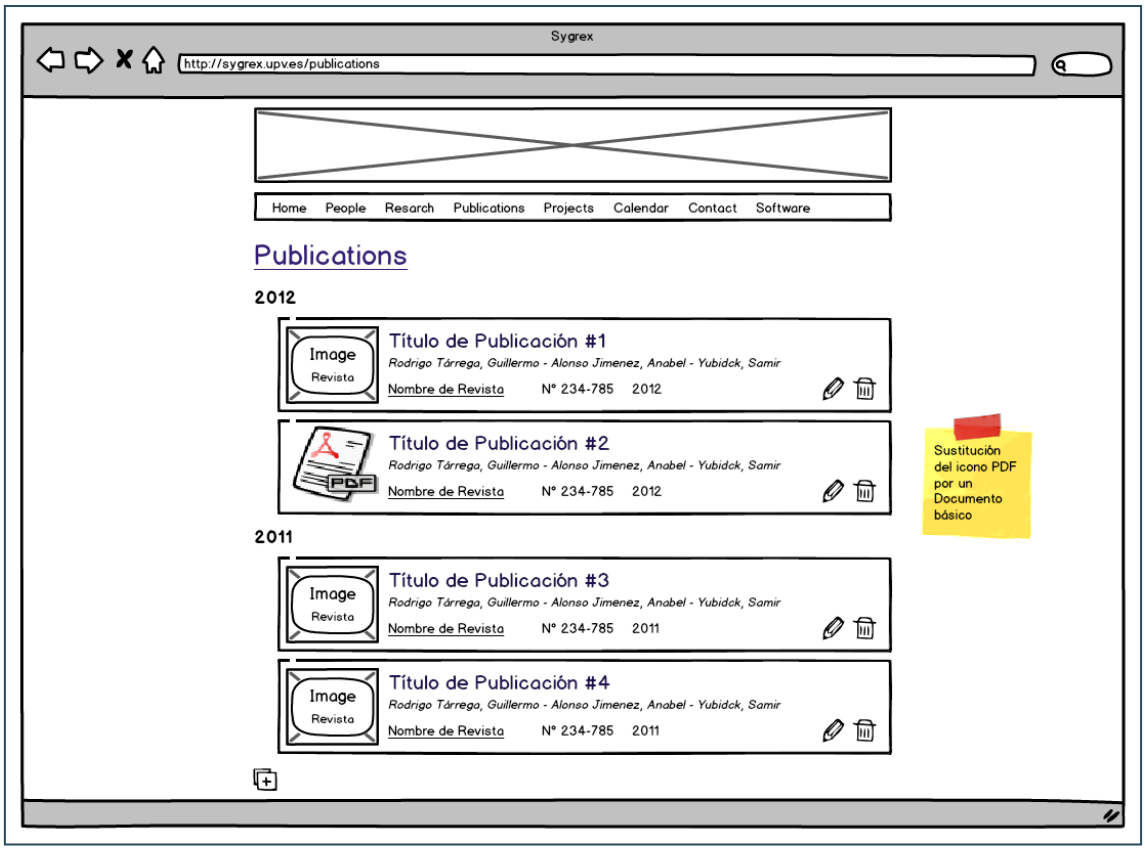

Fig. 5.2.Modelo mejorado para la pantalla Publications.

Tras varias conversaciones sobre como implementar este cambio llegamos a la solución de paneles colapsables, es decir, paneles que pudieran ser desplegados o no, mostrando así la información contenida en ellos. Estos paneles debían agrupar y ocultar las publicaciones por años. Las publicaciones de años anteriores las mantendría ocultas y dejaría ver las publicaciones del último año. Este boceto final, lo diseñamos y se lo mostramos al cliente que lo validó y cuya implementación la consideró óptima a sus necesidades. El resultado de esta idea podemos apreciarlo en la Fig. 5.3.

Es inevitable que una vez creada una página surjan nuevas peticiones. En muchas ocasiones el saber aceptar o denegar estas peticiones puede marcar una diferencia en cuanto a la evolución del proyecto y hacer que tenga un buen uso o que se quede en un proyecto que no se ha adaptado bien a las necesidades del cliente y cuando tenga ocasión, será sustituido por otro.

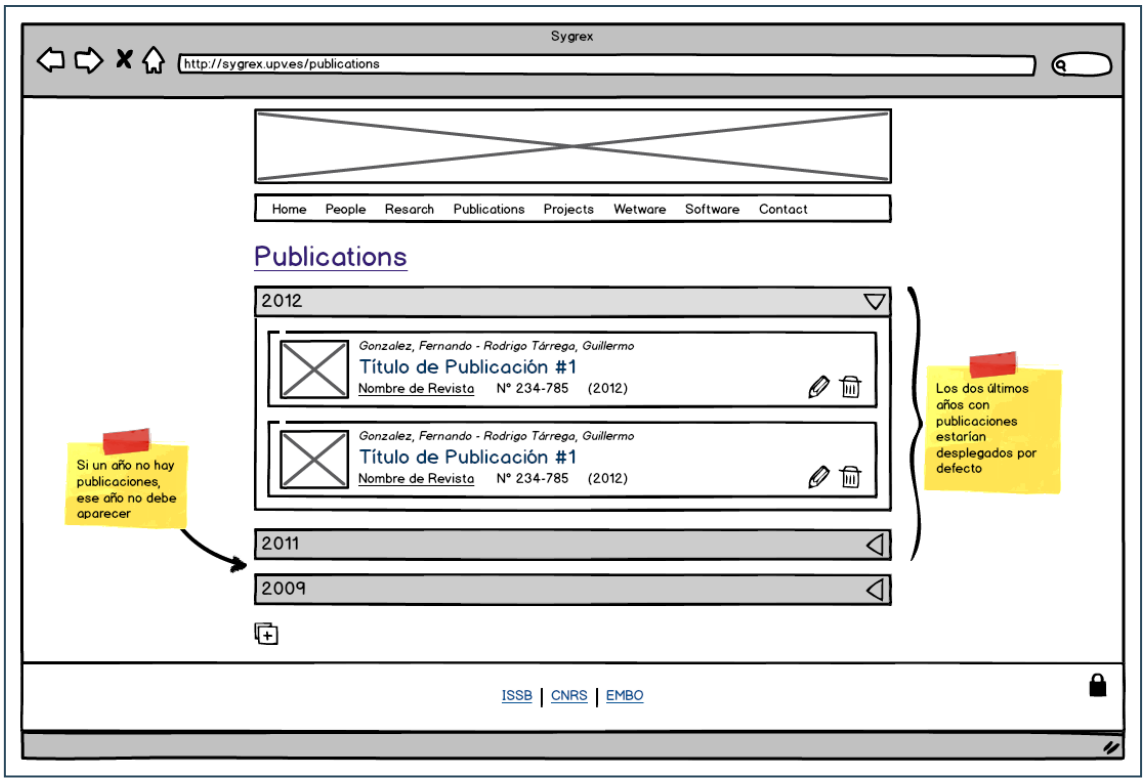

Fig. 5.3.Última versión del modelo para la pantalla Publications.

5.2. Implantación

Para que la web sea pública y que el administrador pueda gestionar el contenido desde cualquier plataforma con acceso a Internet, es necesario pasar por una fase de implantación. En nuestro caso, como la web está construida y diseñada con un fin real, hemos tenido que pasar por esta fase de implantación. Esta fase es muy dependiente del entorno donde se vaya a almacenar la web. Esto hace que la misma web pueda recorrer otro trayecto para su implantación en un nuevo entorno. En particular, vamos a implantar esta web en un nodo Xeon Quad del cluster del Instituto de Biología Molecular y Celular de Plantas (IBMCP), sito en la Ciudad Politécnica de la Innovación, en el campus de la Universidad Politécnica de Valencia.

Este entorno ofrece la posibilidad de conectarse mediante SSH para transferir, instalar y ejecutar programas o archivos, pero carece de un entorno gráfico. Sin embargo, hemos optado por hacer uso de un gestor de máquinas virtuales. Por lo tanto, para implantar nuestro sistema hemos tenido que instalar y configurar un sistema operativo dentro de la instancia de una máquina virtual compatible con la versión que se usa en el cluster donde vamos a hacer la implantación. Una vez tenemos el sistema operativo, ha sido necesario instalar la base de datos y el servidor de aplicaciones web Tomcat para almacenar y mantener el servicio arrancado. Es importante que una vez tenemos este sistema montado con la página web arrancada hagamos una prueba

completa de cada punto para detectar posibles incompatibilidades con el nuevo entorno y resolver cualquier problema que pudiéramos tener. A la hora de hacer la implantación, hemos tenido que tener en cuenta las versiones de los programas usados. También hemos generado una serie de scripts para arrancar o parar el servicio web si hubiera algún tipo de problemas. Cabe decir por último que para poder llevar a cabo esta implantación de manera satisfactoria, así como para configurar el entorno previa y posteriormente, hemos necesitado la ayuda y consejos del administrador del sistema (del servicio de bioinformática del IBMCP).

6. Evaluación y pruebas

Trasfinalizar con el desarrollo (implementación y post-implementación), evaluamos la web rellenado la misma con datos reales del cliente. Esto permite detectar en última instancia fallos cometidos durante la fase de desarrollo, y también evaluar junto con el cliente el resultado final. Al mismo tiempo, llevamos a cabo una fase de pruebas para comprobar quela aplicación es robusta. Es esencial ejecutar pruebas modulares intentando abarcar todas las situaciones que pudieran ocurrir o que un usuario pudiera ejecutar. En nuestro caso, serán pruebas de cada pantalla por separado, las cuales contienen las acciones de creación, edición y borrado.

6.1. Evaluación

La página de inicio de la web desarrollada es, como es habitual, la página de inicio o página *Home*. En ella encontramos un carrusel con imágenes relacionadas con el trabajo que se lleva a cabo en el laboratorio de investigación. Esto le da una gran visibilidad y atractivo a la página web. A continuación, se mostrarán una serie de noticias de interés para cualquier usuario que entre a la web. Por ejemplo, puede ser interesante mostrar el último artículo publicado de impacto, el último proyecto concedido al grupo de investigación, así como la disponibilidad de prácticas en el propio laboratorio para los alumnos que estuvieran interesados. Todo ello acompañado, si es necesario, de un enlace de interés donde poder ofrecer más información. En la Fig.6.1. se puede ver la apariencia que tiene la versión final.

La siguiente pantalla del menú nos permite visualizar información relacionada con las personas que lo componen el grupo de investigación (*People*). El personal aparece separado por categorías según el rol que ocupe en el propio laboratorio. Encontraremos desde estudiantes cursando prácticas, hasta técnicos de laboratorio o estudiantes de doctorado, entre otros. Además, de una fotografía por cada persona, se facilita la forma de contacto con cada una de ellas. En la Fig. 6.2 puede verse la apariencia de esta pantalla.

Otra de las pantallas a las que tendremos acceso en la web es la pantalla de investigación o *Research*, en la que se muestran las diferentes líneas de investigación sobre las que está interesado el grupo de investigación y, por tanto, sobre las cuales se están desarrollando nuevos trabajos. El investigador principal, si lo resulta conveniente, también podrá añadir información de líneas de investigación de trabajos anteriores. Por cada línea de investigación mostraremos una imagen relacionada, así como una breve

descripción sobre la línea de investigación. En la Fig. 6.3 puede verse el detalle de esta pantalla.

En la pantalla *Publications* tenemos un registro de cada una de los artículos publicados en revistas científicas por el grupo de investigación o por miembros que pertenecen a él, como es el caso de los artículos publicados por el investigador principal del laboratorio. Las publicaciones aparecerán agrupadas por año de publicación, quedando más visibles los artículos de los dos últimos años. En siguiente Fig. 6.4 se muestra el detalle de las publicaciones de dos años diferentes, así como paneles que contienen información de artículos de otros años anteriores.

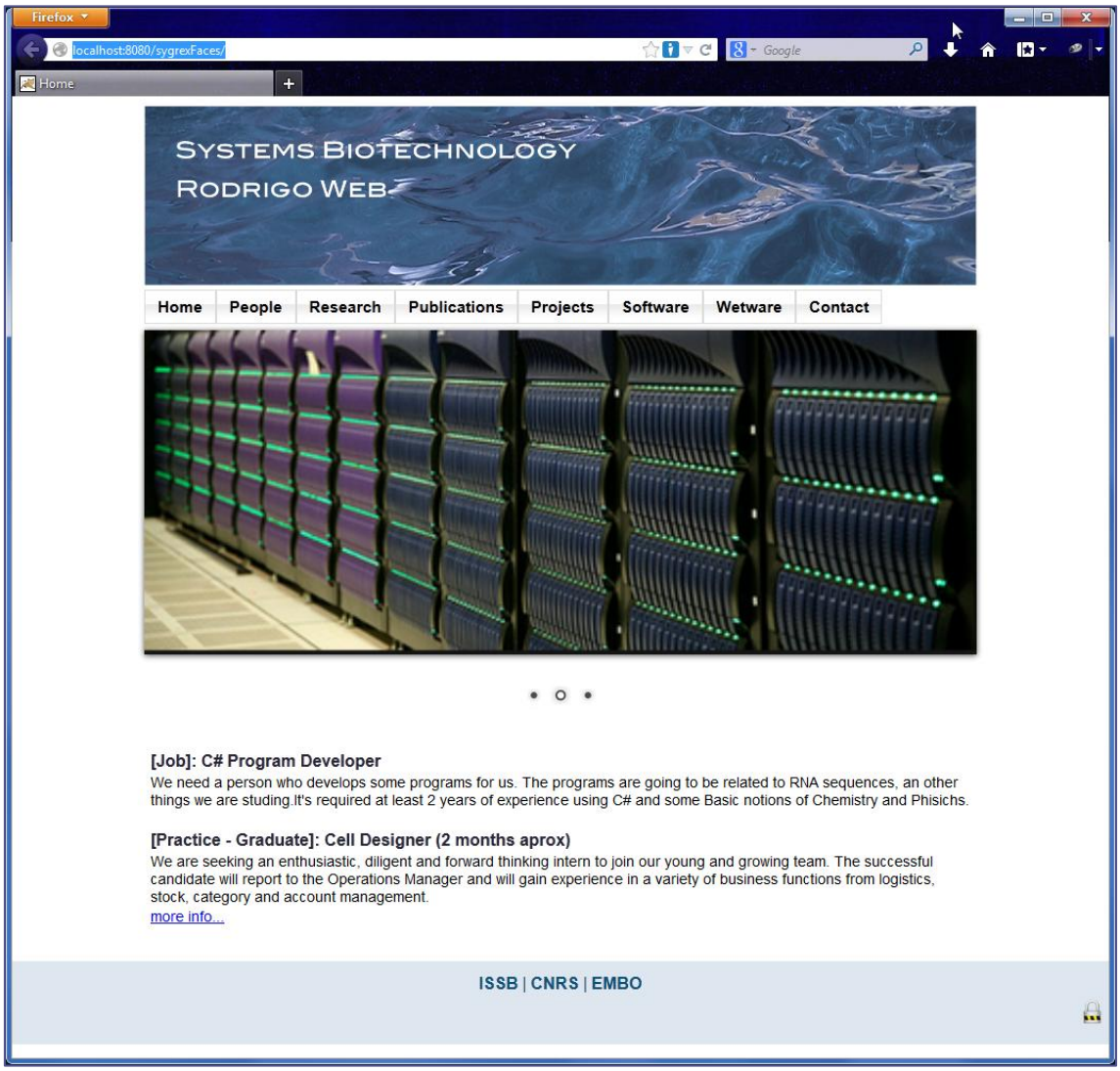

Fig. 6.1. Captura de la pantalla Home.

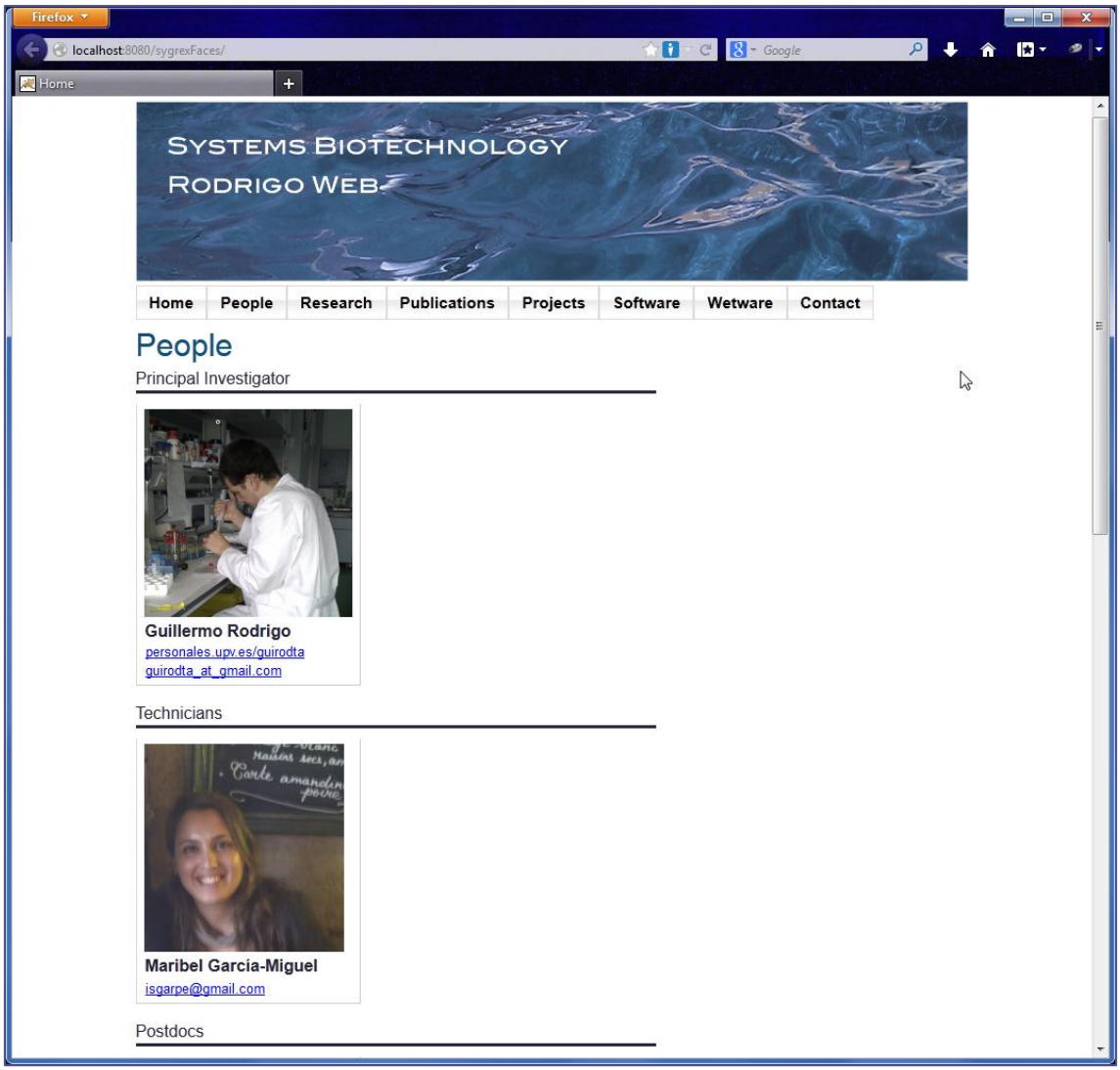

Fig. 6.2. Captura de la pantalla People.

En la Fig. 6.5 se muestra el contenido de la página de *Software*, la cual contiene un registro de algunas herramientas útiles para el grupo de investigación. Estas herramientas pueden ser de ámbito general, gratuitas, privadas o desarrolladas por el propio grupo. Por cada software se mostrará una imagen relacionada con él, así como una breve descripción de su utilidad o funcionalidad. Es importante mencionar que el software privado de pago, no se adjunta como fichero, si no que se adjunta como enlace a la página del desarrollador principal o al vendedor de la herramienta.

La pantalla *Wetware* corresponde a un pequeño inventario, útil sobre todo para el propio grupo de investigación sobre algunos de los recursos del laboratorio, como es el caso de plásmidos, células u otros componentes químicos necesarios. En la Fig. 6.6 podemos visualizar una muestra de posible inventario concreto.

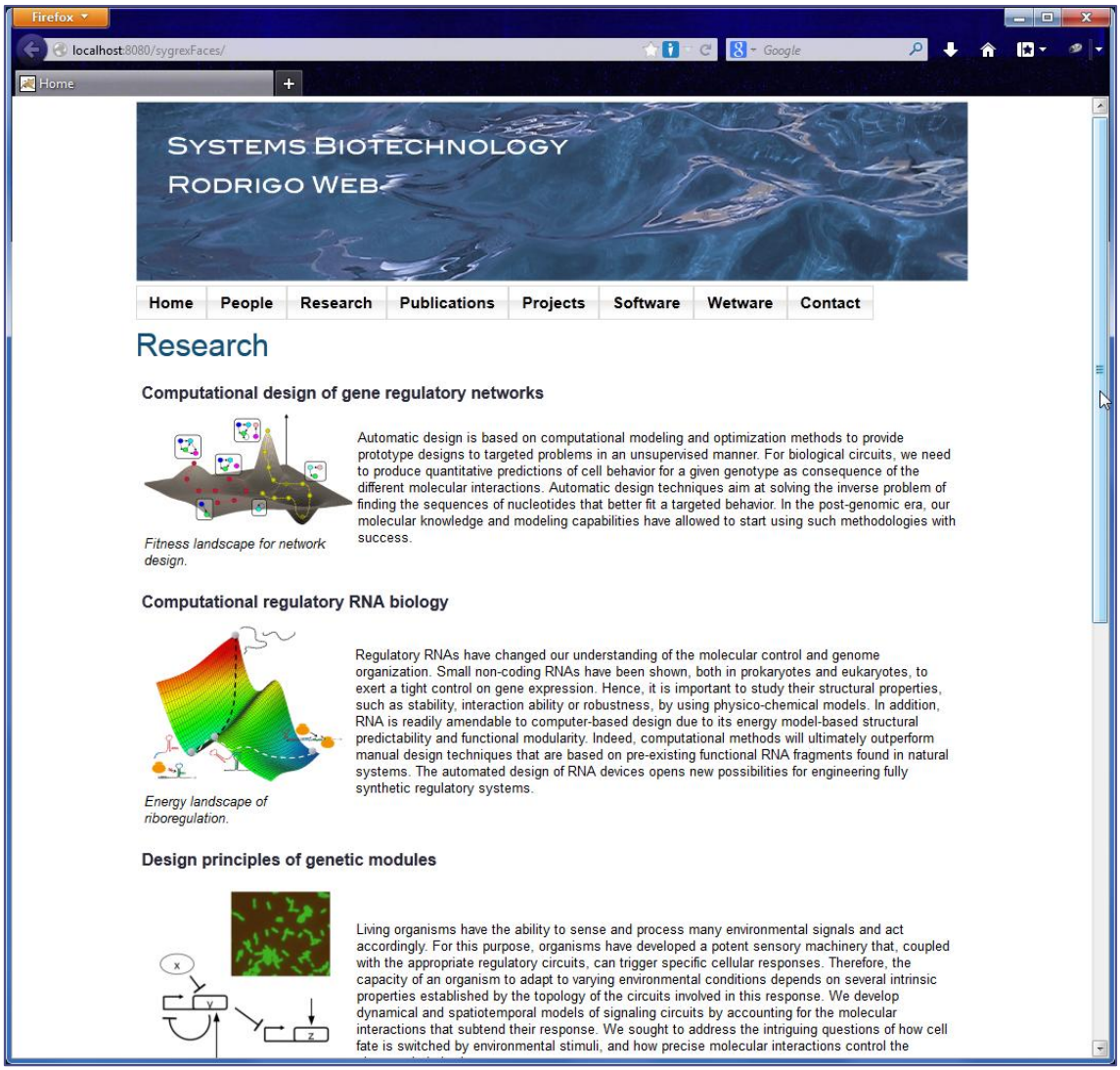

Fig. 6.3. Captura de la pantalla Research.

Por último, disponemos de la forma de contacto con el grupo de investigación (*Contact*). Esta pantalla es totalmente configurable. En la Fig. 6.7 podemos ver un ejemplo que incluye la localización del grupo de investigación, con información de interés, además de un mapa de Google con la localización.

Para el cambio de rol de usuario se ha optado por un sistema minimalista con un panel donde el usuario administrador podrá validarse mediante una contraseña, pudiendo así modificar todo el contenido web. En la Fig. 6.8 se muestra un ejemplo de esta validación.

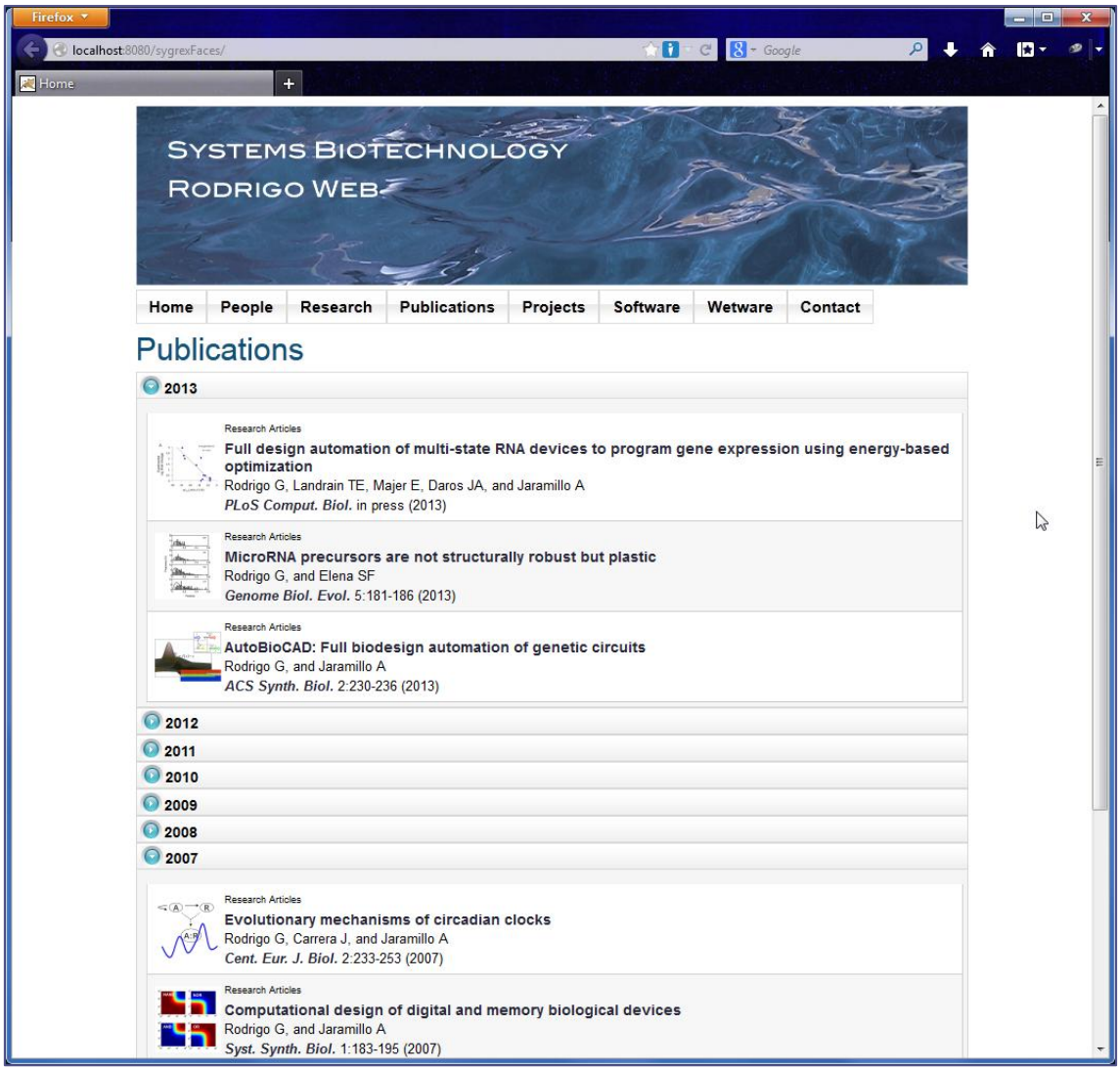

Fig. 6.4. Captura de la pantalla Publications.

Una vez el usuario administrador es validado por la aplicación, tiene la opción de modificar cada pantalla de la aplicación. En este momento, el administrador puede acceder mediante unos iconos de creación, edición y borrado, que son las operaciones permitidas en bases de datos. En las Figs. 6.9 y 6.10 podemos ver un ejemplo de sobre el mantenimiento de dos pantallas diferentes; la 6.9 pertenece a la edición de *Research* y la 6.10 pertenece a la edición de *Publications*.

Cuando el usuario administrador intente dar de baja o eliminar cualquier contenido de la web le aparecerá un mensaje que debe confirmar para hacer efectiva esta operación. En la Fig. 6.11 se muestra este mensaje al intentar eliminar un registro de *Research*.

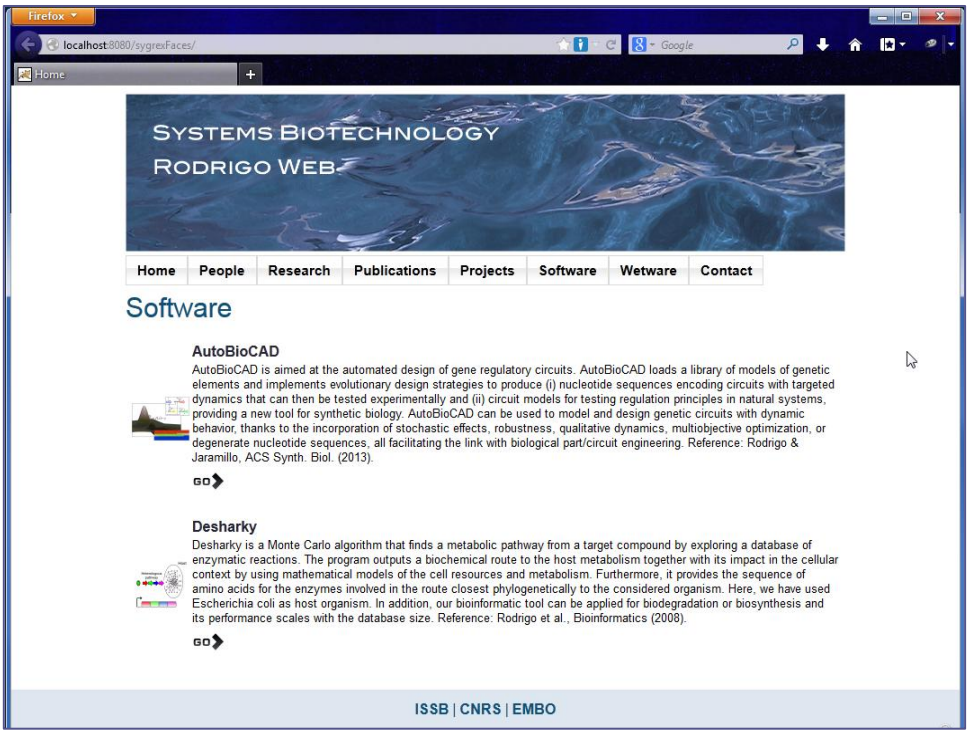

Fig. 6.5.Captura de la pantalla Software.

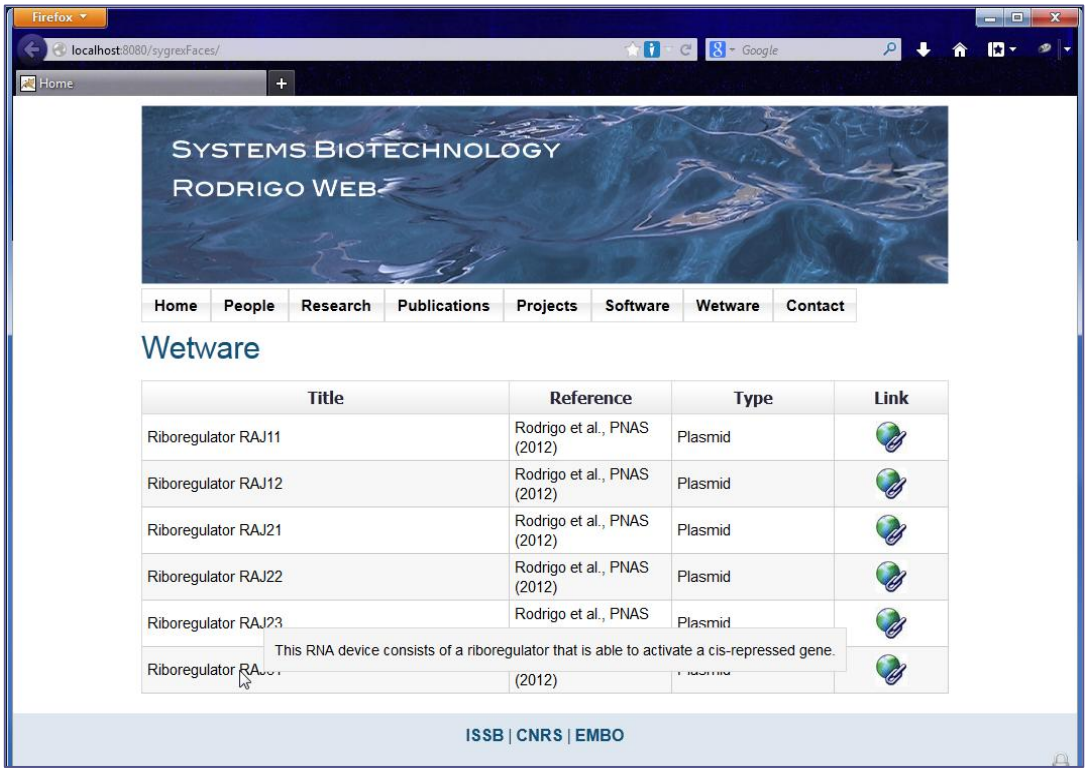

Fig. 6.6. Captura de la pantalla Wetware.

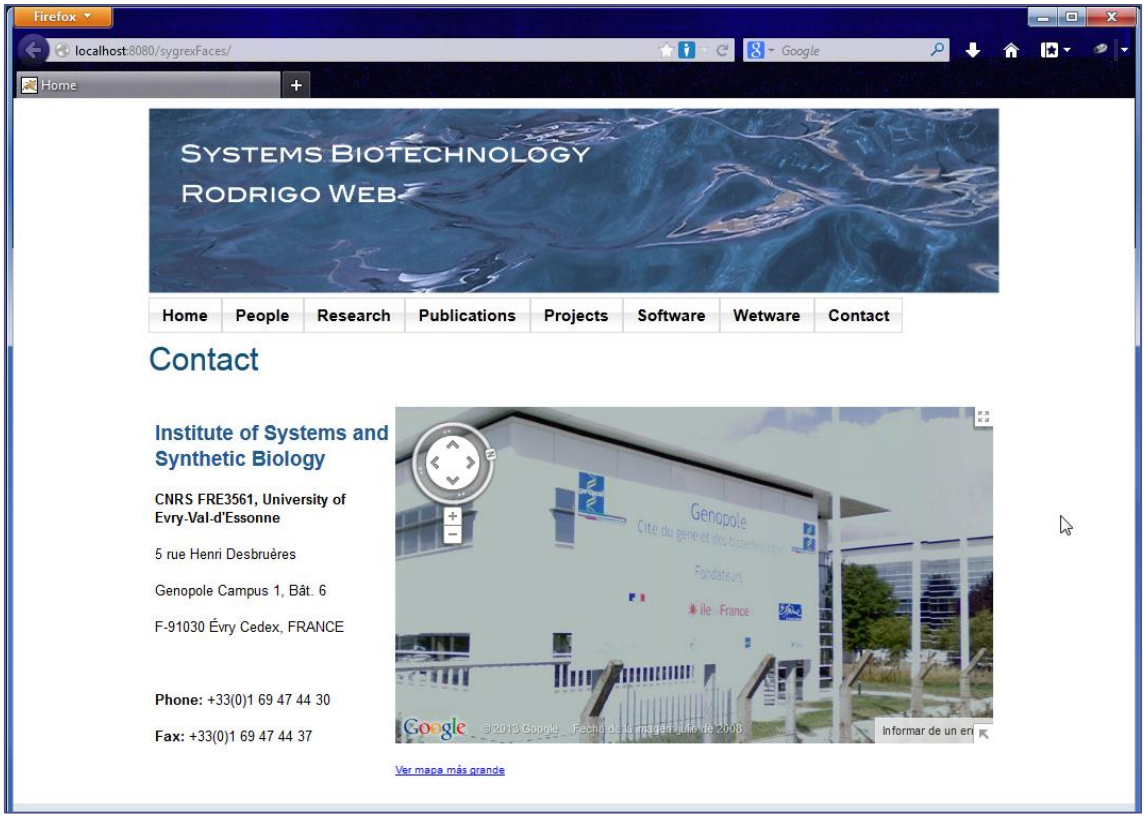

Fig. 6.7.Captura de la pantalla Contact.

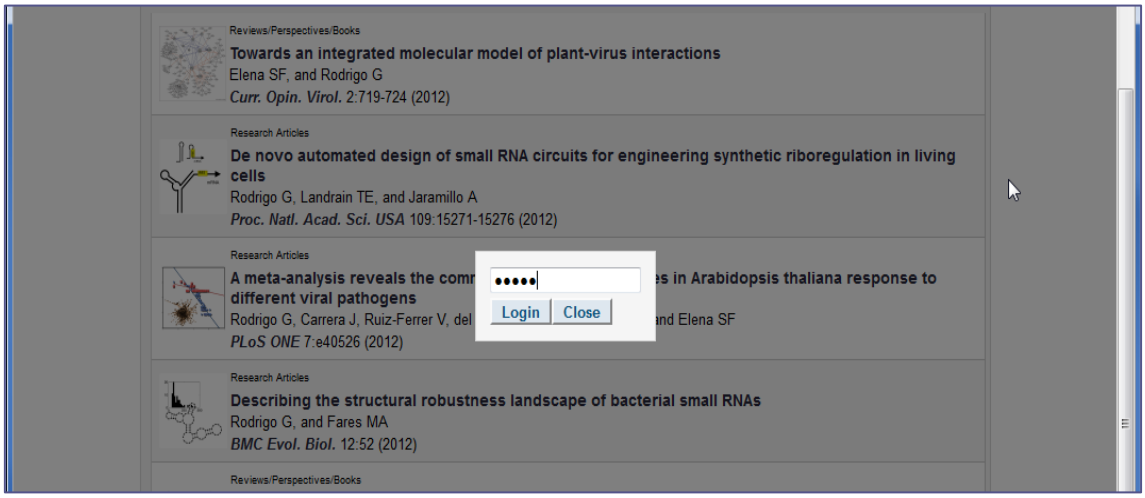

Fig. 6.8. Captura del cuadro para el Login.

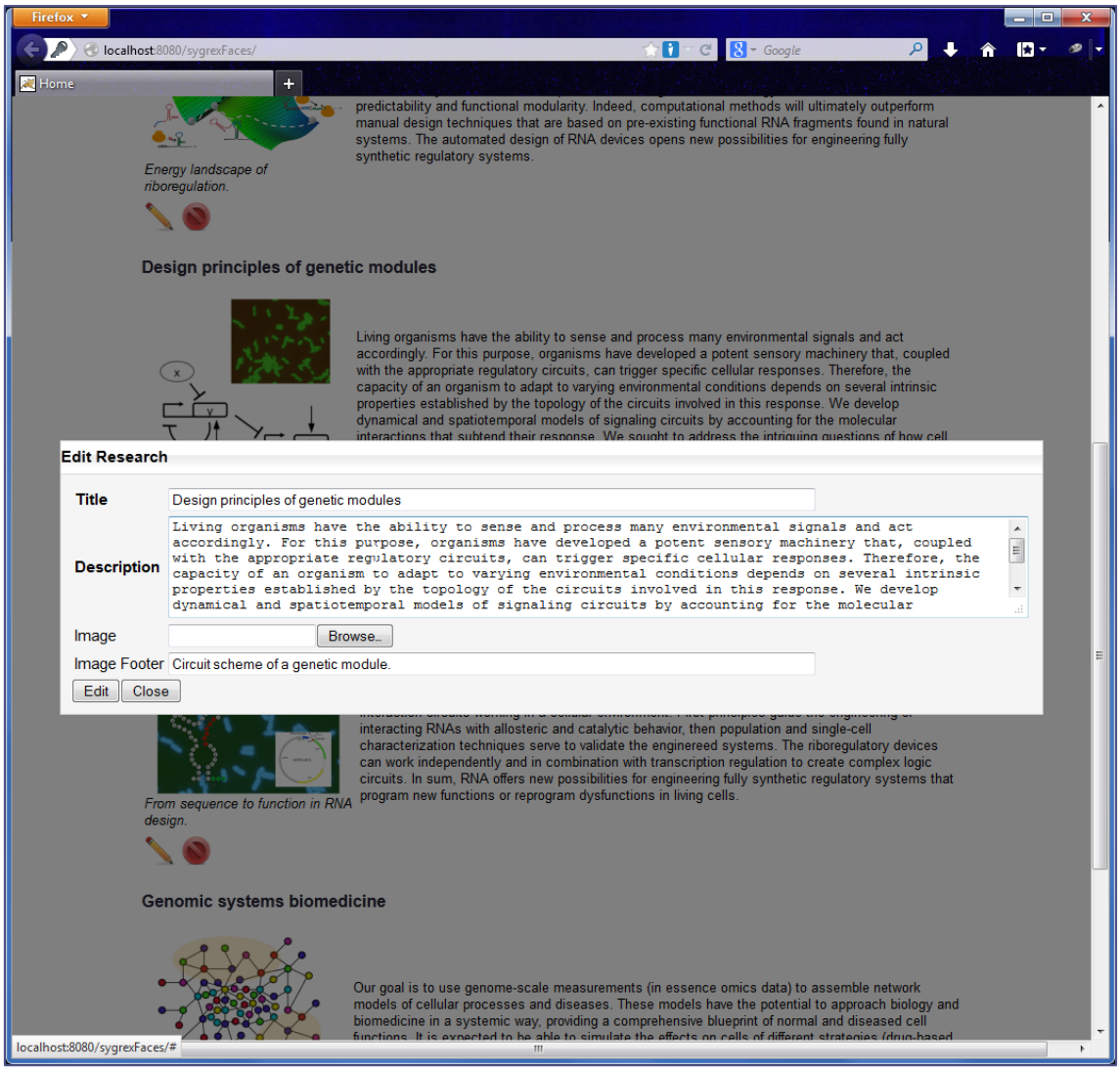

Fig. 6.9. Captura de la pantalla Research mostrando el cuadro de edición.

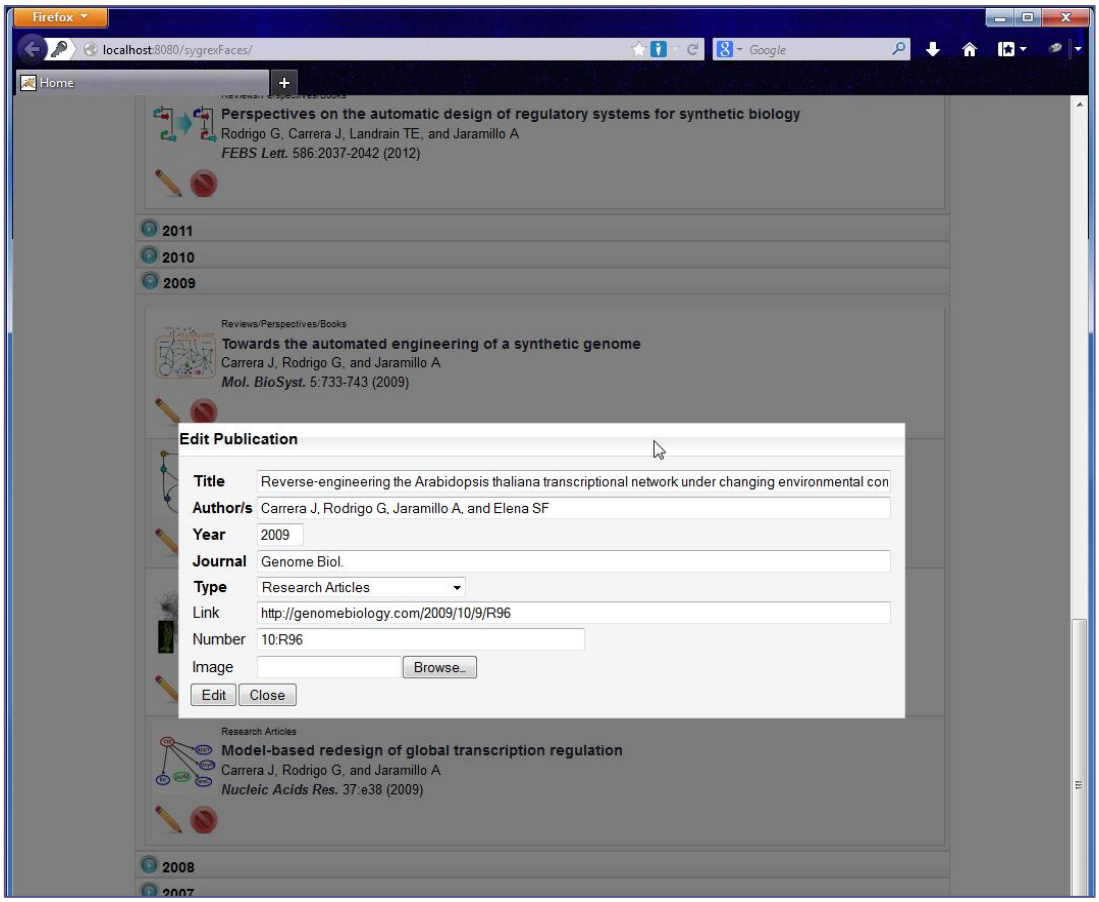

Fig. 6.10. Captura de la pantalla Publications mostrando el cuadro de edición.

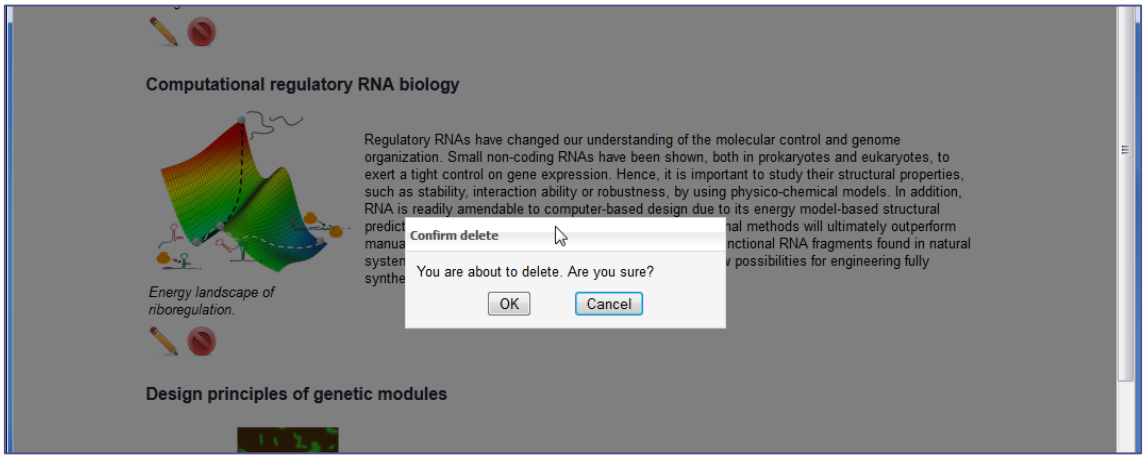

Fig. 6.11. Captura de una pantalla con un cuadro de seguridad de edición al eliminar un elemento. Se pregunta si realmente se desea eliminar.

6.2. Pruebas

Todas las pruebas son dependientes del tipo de funcionalidad que estemos desarrollando y según la complejidad de la aplicación pueden ser más o menos complejas. En desarrollos complejos o de funcionalidad muy amplia es común usar clases de Test y sistemas de pruebas automáticas como JUnit o sistemas de carga del entorno como JMeter. Esta fase de pruebas puede llegar a ser tan compleja que se puede crear un grupo de trabajo cuya finalidad sea detectar errores en el código e informar de ellos, en última instancia, a los desarrolladores que generan el código para su corrección. Sin embargo, esto no ha sido necesario para las pruebas en este proyecto.

Debido a la importancia de esta fase de pruebas hemos tenido que hacer un esfuerzo intentando poner al límite nuestro desarrollo y detectar posibles errores. Todo este tipo de pruebas han sido llevadas a cabo por nosotros como desarrolladores y por el cliente como usuario final. Lo hemos hecho de esta forma para permitir al usuario usar la web y comprobar si le resultaría intuitivo el mantenimiento de la página sin necesidad de grandes explicaciones. Así, el cliente ha podido conocer si se siente cómodo con el proyecto desarrollado, pudiendo también comprobar si la página web reúne todas las especificaciones y necesidades expuestas a lo largo de las diferentes entrevistas que se han llevado a cabo.

Debido a las características de nuestra aplicación, las pruebas han sido de dos tipos. En primer lugar, hemos considerado las pruebas de lógica donde nos hemos asegurado que no hubiera errores no controlados, excepciones que afectaran al uso correcto de la aplicación, fallos en la lógica de la aplicación o posibles puertas traseras al código. Esto es, en este tipo de pruebas hemos tenido en cuenta todo aquello que es responsabilidad directa del desarrollador y que podría llegar a ser un problema si se usara inapropiadamente por el propio usuario de la aplicación, así como por terceras personas. En segundo lugar, las pruebas han sido efectuadas por varias personas que han interpretado el rol de un usuario anónimo (sin permisos) y un usuario administrador (con permisos de modificación). Esto es de utilidad por dos razones: i) para poner a prueba nuestra aplicación con varios usuarios concurrentes y ii)para validar que se han cumplido los requisitos planteados por el cliente.

Por otro lado, es habitual que en esta fase final del proyecto, al cliente se le ocurran nuevas ideas para llevar a cabo y será responsabilidad del jefe del proyecto o del desarrollador del proyecto llevarlas a buen puerto, iniciar una nueva fase de mejoras o descartar cualquier tipo de modificación.

7. Conclusiones

Una vez finalizado este proyecto, vamos a presentar un resumen con conclusiones. En primer lugar, hemos conseguido definirlos requisitos de una página web para un cliente real, y diseñarla desde el principio para ajustarla tanto a las especificaciones iniciales como a las del desarrollo. Hemos implementado el código usando una de las últimas tecnologías Java del mercado e implantado esta web en un servidor cuya localización sita en la Universidad Politécnica de Valencia.

Para el diseño de la web, ha sido necesario el análisis comparativo de tecnologías de programación. Así mismo, hemos hablado sobre diversas herramientas que pueden ser explotadas para el desarrollo de aplicaciones web. En particular, hemos explicado y profundizado en la tecnología JSF con un caso práctico. Así, hemos podido ver cómo resolver algunos problemas que surgen a la hora de implementar una página web. De relevancia, con este proyecto hemos profundizado en el uso de JPA como motor de persistencia y, en concreto, en el uso de la tecnología Hibernate y el lenguaje JPQL tanto para el acceso a los datos como para crear y mantener las diferentes entidades que hemos necesitado al generar nuestra página.

En suma, hemos dado una solución eficiente a un problema real que podría ser reutilizada o tomada como base para futuros desarrollos con unas características similares. Al mismo tiempo, hemos aprendido a llevar a cabo un proyecto completo, y hemos ganado conocimiento de la interacción rica con el cliente.

7.1. Propuestas para ampliar la web

Además, una vez finalizada la implementación actual, el cliente nos ha ofrecido la posibilidad de realizar una segunda fase de desarrollo que dotaría a la web de una mayor funcionalidad y ofrecería más servicios al grupo de investigación, así como a los usuarios que accedan a la página. Éstas son algunas ideas que han surgido tras finalizar con el proyecto:

- Cambio del carrusel de la página inicial por uno más completo y más robusto.
- Añadir un mapamundi con estadísticas sobre los accesos a la aplicación.
- Generar estadísticas de uso sobre las páginas más consultadas y generar gráficas en función del histórico de accesos.
- Envío automático mensual de los correos con las estadísticas de cada página e inclusión de un botón para un envío manual desde el perfil de administrador.
- Generar un sistema propio para administración de usuarios para el Administrador.

- Herramienta para la ejecución online del software desarrollo por el grupo de investigación.

Algunas de estas ideas son interesantes y ya se empiezan a llevar a cabo como proyecto independiente.

8. Bibliografía

[1] García Álvarez de Toledo, J., Fernández Sánchez, R. (2011) Difusión y divulgación científica en internet. Gov. P. Asturias, Cienciatec, Asturias.

[2] Estructuras de datos en JAVA. Compatible con JAVA2. Weiss, M.A. Editorial Pearson. ISBN: (978-)84-7829-035-4.

[3] El Lenguaje de Programación JAVA. K. Arnold, J. Gosling, D. Holmes. Editorial Pearson. ISBN: (978-)84-7829-045-1.

[4] JavaScript. Revisado y actualizado 2004. José Manuel Alarcón. Editorial Anaya Multimedia. ISBN:(978-)84-415-1631-6.

[5] Hibernate: Hibernate Community Documentation © 2007 by Red Hat, Inc. http://docs.jboss.org/hibernate/orm/4.2/manual/en-US/html/

[6] UCSD Biodynamics Laboratory web (headed by Hasty, J.). University of California San Diego, La Jolla CA. http://biodynamics.ucsd.edu/

[7] Desarrollo Web con PHP y MySQL. PHP 5 y MySQL 4.1 y 5. Luke Welling, Laura Thomson. Editorial Anaya Multimedia, ISBN: (978-)84-415-1818-6.

[8] Joomla: What is Joomla? © 2012 by Open Source Matters, Inc. http://www.joomla.org/about-joomla.html

[9] Liferay: Liferay Portal 6.1 - User Guide © 2013 LIFERAY INC. http://www.liferay.com/documentation/liferay-portal/6.1/user-guide/-/ai/what-is-liferay-

Webs consultadas

JSF

http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=ejemploWebIcefaces

http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=IceJboss

JPA

http://courses.coreservlets.com/Course-Materials/hibernate.html

MySQL

http://www.packtpub.com/article/visual-mysql-database-design-in-mysql-workbench

Balsamiq

http://www.balsamiq.com/products/mockups

Icefaces

http://www.icesoft.org/java/projects/ICEfaces/overview.jsf

Thumbnailator

https://code.google.com/p/thumbnailator/

Arquitectura de 3 capas

http://www.eduardoaf.com/blog-frameworks/frameworks-mvc/arquitectura-en-3-capas-arquitectura-mvc-y-la-poo-1/

Otras:

http://www.consultoriajava.com/tools/hibernate.shtml

http://www.maestrosdelweb.com/editorial/los-diferentes-lenguajes-de-programacion-para-la-web/

https://docs.google.com/document/preview?hgd=1&id=1ACvEZgyTjKj_6tqPwbi-gfG_BZr8H7A_fatsUWYpBeA