UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

etsinf

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

PROYECTO FINAL DE CARRERA

INGENIERÍA INFORMÁTICA

SISTEMA DE GESTIÓN DE CONTENIDOS WEB PARA UNA AGENCIA TURÍSTICA DE CERDEÑA

AUTOR

Santiago García Belmonte

DIRECTOR

Ramón Pascual Mollá Vaya

Marzo 2012

A mis padres,

que con su esfuerzo y sacrificio diario me permitieron alcanzar esta meta, que me animaron y apoyaron tanto en los buenos momentos como en los malos, y no me dejaron abandonar bajo ningún concepto.

A mis hermanos,

que ejercieron a la perfección su condición de hermanos mayores, consintiéndome en los momentos necesarios, sirviéndome de ejemplo continuamente, haciéndome mucho más llevadera mi vida de estudiante.

A mi amigo Mauro Casula,

del que tanto he aprendido de esta profesión y con el que comparto en gran medida mi visión de la vida. No cabe duda que sin su ayuda este proyecto no podría haberse finalizado. Espero no dejar de aprender de él durante el tiempo que sea posible.

A Ramón Pascual Mollá Vaya,

quien amablemente me prestó su ayuda para que pudiera presentar este proyecto y aceptó de buena gana ser el director del mismo.

A la Universidad Politécnica de Valencia,

y a la Escuela Técnica Superior de Ingeniería Informática,

por permitirme finalizar mis estudios después de tanto tiempo ausente y por la cantidad de conocimientos, recuerdos, vivencias y anécdotas que me llevo de mi paso por éstas.

Índice

Índice de ilustraciones

Capítulo I. Introducción

1. Introducción

El presente documento constituye la memoria del Proyecto Final de Carrera de Ingeniería Informática, realizado en la Escuela Técnica Superior de Ingeniería Informática de la Universidad Politécnica de Valencia por el alumno Santiago García Belmonte, y dirigido por el profesor Ramón Pascual Mollá Vaya.

Seguidamente se describen en detalle las fases de realización de dicho proyecto, que consiste en la creación de un portal web dinámico o Sistema de Gestión de Contenidos web, para una agencia turística de la isla de Sant'Antioco al suroeste de Cerdeña (Italia).

Con la conclusión del proyecto se pretende cumplir con los objetivos de comunicación, presencia en Internet, y gestión interna básica, que el negocio de dicha agencia requiere y de los que depende en gran medida para su éxito comercial.

2. Motivación y objetivos

Sant'Antioco es una pequeña isla situada al suroeste de la isla de Cerdeña, y conectada a ésta mediante un único puente. Pese a su apariencia inhóspita y aislada, constituye un enclave turístico muy importante en el sur de Italia. Sus aguas y costas vírgenes, protegidas de la masificación urbanística, hacen de Sant'Antioco un valor seguro por el que apostar en lo que a materia de turismo se refiere. No obstante, gran parte de los residentes de Sant'Antioco prefieren alejarse en épocas típicamente vacacionales. Y por otro lado, es habitual que muchas propiedades de la isla sean segundas residencias (deshabitadas gran parte del año).

En el marco anteriormente descrito, surge con naturalidad el conocido negocio del alquiler de propiedades (pisos, apartamentos, casas, villas, etc...) en las temporadas en que éstas se encuentran vacías. Si se añade un intermediario que actúe entre inquilinos y propietarios, surge la figura de la agencia turística.

El objetivo principal de este Proyecto Final de Carrera consiste en el desarrollo de un *Sistema de Gestión de Contenidos web* o *CMS* (de sus siglas en inglés *Content Management System*), adaptado a las necesidades y el tipo de negocio de una agencia turística que opera en el contexto detallado anteriormente. Por tanto, dicho sistema tendrá que cumplir con una serie de principios básicos que ayuden a los propósitos de la agencia turística:

- Permitir la gestión del catálogo de propiedades o inmuebles en alquiler.

  El sistema debe dar la posibilidad de tratar toda la información necesaria de los inmuebles (precios por temporadas, reservas y periodos disponibles, imágenes, características, etc...).

- Permitir la gestión de la estructura y la información del portal web.

  Se debe tener la capacidad de agregar páginas de contenido al portal, cambiar el contenido de dichas páginas (como textos e imágenes), decidir de qué forma se visualiza la información y que contenido se publica.

- Óptima presencia en Internet.

  El portal debe favorecer el posicionamiento natural en los principales buscadores web, mediante el uso de URLs amigables, la generación de código HTML lo más simple posible y el uso de otras técnicas SEO Pasivas. Al mismo tiempo se debe proporcionar una imagen atractiva y clara del portal, que permita a los visitantes orientarse sin problemas.

- Facilitar la búsqueda de inmuebles a los visitantes.

  Se debe proporcionar un sistema de búsqueda intuitivo en el catálogo publicado, que mediante filtros y parámetros adecuados, pueda localizar el elemento óptimo para el visitante interesado en alquilar una propiedad.

- Multi-idioma y comunicación.

  Dado el carácter publicitario del portal, la variedad de nacionalidades de los posibles clientes y la necesidad de llegar al mayor número de usuarios, el sistema debe ser capaz de gestionar múltiples idiomas y la información relacionada con cada idioma.

Como objetivo académico, este proyecto permite familiarizarse con la construcción de aplicaciones web desde cero, así como adquirir experiencia en las diferentes tecnologías utilizadas para la realización del mismo.

3. Contexto

Para simplificar, la solución planteada en este proyecto se puede dividir en dos partes bien definidas: una zona privada (protegida por nombre de usuario y contraseña) para uso exclusivo del administrador del sitio, y una zona pública accesible por cualquier visitante del portal.

En la sección privada, el administrador (previamente autentificado) podrá realizar todas las tareas de gestión requeridas. Desde una barra de menú tendrá acceso a la gestión del catálogo de propiedades en alquiler, a la gestión del contenido del portal y la gestión de características especiales de los inmuebles. Toda esta información se verá reflejada de manera ordenada en la parte pública.

En la sección pública, cualquier visitante podrá navegar por el contenido del portal, acceder al catálogo de propiedades, realizar búsquedas sobre el catálogo y contactar con la agencia turística mediante formularios.

La implementación se apoya en el uso de Symfony (en su versión 1.2), un framework diseñado para optimizar y agilizar el desarrollo de aplicaciones web. Symfony basa su arquitectura en el patrón de diseño MVC, comúnmente utilizado en aplicaciones web, que separa el código en tres capas (Modelo, Vista y Controlador) de acuerdo con su naturaleza. En secciones posteriores de este documento se tratará detenidamente las características de dicho patrón de diseño.

Capítulo II. Especificación de requisitos

1. Introducción

La especificación de requisitos describe las características que el proyecto debe satisfacer para cumplir con las exigencias del cliente, en este caso una agencia turística. Este apartado se ha desarrollado siguiendo el estándar 830 de 1998 del IEEE, el cual presenta un conjunto de directrices o normas para la Especificación de Requisitos Software o ERS, siendo uno de los estándares más utilizados en proyectos web.

1.1. Propósito

La captación y acotamiento de los requisitos de un proyecto es vital para el éxito o el fracaso del mismo. Cuanto antes se consiga definir dichos requisitos y acordarlos con el cliente, antes darán comienzo otras etapas del ciclo de desarrollo y menos interrupciones se ocasionarán en este ciclo.

El propósito principal de esta ERS es reflejar de forma ordenada y clara los requisitos funcionales del proyecto, así como otras posibles características del sistema que puedan ser relevantes para su construcción: condiciones de rendimiento, restricciones tecnológicas, dependencias con otros sistemas, etc. Una buena ERS puede servir de contrato entre desarrollador y cliente, como base para la planificación y estimación de costes, o de punto de referencia para la validación del proyecto.

Esta ERS va dirigida tanto al desarrollador del proyecto como al cliente, así como a cualquier persona interesada en conocer el funcionamiento y las características de la aplicación.

1.2. Ámbito

El producto software que se describe en esta ERS es una aplicación web, un sistema de gestión de contenidos (CMS) que se especializa de manera destacada en el mantenimiento y gestión de un catálogo de propiedades (inmuebles) en alquiler para una agencia turística de nacionalidad italiana.

Una vez concluida la aplicación e implementadas todas las funcionalidades, el administrador (con control total) podrá definir la estructura y el contenido de un portal web público, así como gestionar un catálogo avanzado de inmuebles. Es decir, el

administrador será capaz de añadir páginas al portal, decidir de qué manera se publican, completar información relevante para el posicionamiento pasivo, e incluir contenido en ellas (texto e imágenes) organizado en base a plantillas predefinidas. En cuanto al catálogo de inmuebles, cabe destacar que la información adyacente que se gestionará será muy amplia: se podrá elegir y subir las imágenes de la galería de cada inmueble, definir las temporadas vacacionales y los precios por cada temporada, gestionar la disponibilidad de los inmuebles en función de las reservas, etc. Y sobre todo, cada información almacenada en la base de datos podrá definirse para cada idioma preestablecido, ya que el sistema estará preparado para la gestión de varios idiomas.

A todos los efectos, el visitante del portal público podrá navegar por el mismo y consultar su contenido como en cualquier otro portal web de la WWW, de manera transparente a los procesos que intervienen en la gestión de dicho portal. De la misma forma, se proporcionará al visitante una zona del portal donde podrá visualizar y consultar el catálogo de inmuebles, ver el detalle de los inmuebles, y realizar búsquedas sobre el catálogo en función de: la disponibilidad, la zona o localidad, y el tipo de inmueble.

Pese a que la aplicación no proporcionará una zona privada para el visitante, se facilitará el contacto entre el visitante y la agencia turística (administrador) mediante formularios específicos, especialmente para la realización de reservas.

1.3. Definiciones, acrónimos y abreviaturas

A continuación se definen los términos, acrónimos y abreviaturas utilizadas en la presente ERS, intentando mantener el orden de aparición en el documento o su carácter común.

- **IEEE**: siglas que corresponden a *Institute of Electrical and Electronics Engineers*, asociación técnico-profesional mundial sin ánimo de lucro dedicada, entre otras cosas, a la creación de estándares. Su creación se remonta a 1884.

- **Administrador**: usuario de la aplicación con control total, con permisos para acceder a cualquier funcionalidad de la misma. Este tipo de usuario debe autentificarse en el sistema mediante un nombre de usuario y una contraseña.

- **Visitante**: cualquier usuario de la aplicación no autentificado sin permisos específicos y con acceso restringido, capaz de acceder únicamente a la zona pública de la aplicación.

- **Aplicación web**: aquellas aplicaciones que los usuarios puede utilizar accediendo a un servidor web mediante el uso de un navegador web.

- **Portal (o sitio) web**: conjunto de páginas web relacionadas y comunes a un nombre de dominio en la World Wide Web.

- **CMS**: de sus siglas en inglés *Content Management System*, software que permite crear una estructura de soporte para la creación y administración de contenidos (normalmente web), y que facilita la tarea de su publicación.

- **Buscador web**: también conocido como *motor de búsqueda*, es un sistema informático que busca archivos almacenados en un conjunto de servidores web. Se consultan haciendo uso de palabras clave relacionadas con un tema, y ofrecen el resultado en forma de listado de direcciones web cuyo contenido está relacionado con las palabras clave de la búsqueda realizada.

- **SEO**: del inglés *Search Engine Optimizer* o *Search Engine Optimization*, es el proceso y conjunto de técnicas utilizadas para mejorar la visibilidad o posicionamiento de un sitio web en los diferentes motores de búsqueda (por ejemplo Google o Bing) de forma orgánica (sin pagar por esto al buscador).

- **Posicionamiento pasivo**: conjunto de tareas en el ámbito del SEO que se realizan mayoritariamente durante el proceso de desarrollo de un sitio web y que actúan directamente en la óptima indexación de contenidos en los buscadores web.

- **Base de datos**: conjunto de datos pertenecientes a un mismo contexto y almacenados (de forma electrónica o digital en el contexto de este proyecto) para su uso posterior.

- **WWW**: o la **web**, corresponde a las siglas de *World Wide Web* es un sistema de intercambio de información accesible a través de Internet. Fue creada en 1989 por Tim Berners-Lee y Robert Cailliau durante su trabajo en el CERN.

- **Navegador web**: programa informático que interpreta el código de una página web permitiendo su correcta visualización y la interacción del usuario con la misma.

- **Internet**: conjunto descentralizado y heterogéneo de redes de comunicación de alcance mundial interconectadas que funcionan como una red lógica única. Sus orígenes se remontan a 1969 (ARPANET) y uno de sus servicios más populares es la World Wide Web.

- **Servidor web**: programa informático que procesa una aplicación realizando conexiones con un cliente que recibe la respuesta generada por dicho programa, utilizando el protocolo HTTP para la transmisión de datos. También recibe este nombre la máquina física donde se encuentra instalado este software.

- **Inyección SQL**: es una vulnerabilidad que puede darse en aplicaciones que contengan o bien generen código SQL para el acceso a bases de datos. Suele presentarse por un incorrecto chequeo o filtrado de las variables, normalmente las utilizadas para la entrada de datos. Se dice que ha existido *inyección SQL* cuando, de alguna manera, se inserta o *inyecta* código SQL invasor dentro del código SQL programado, alterando el funcionamiento normal de programa o los datos almacenados en la base de datos.

- **XSS**: del inglés *Cross-site Scripting*, es un tipo de vulnerabilidad que puede darse en aplicaciones cuyo objetivo final es mostrar información en un navegador web. Esta vulnerabilidad permite la ejecución de código de "scripting" malicioso (JavaScript por ejemplo) en el contexto de otro sitio web y, es provocada por no validar de forma adecuada los datos de entrada de la aplicación.

- **CSRF** (o **XSRF**): del inglés *Cross-site Request Forgery*, es una vulnerabilidad de los sitios web por el que comandos no autorizados pueden transmitirse por un

usuario en el que el sitio confía. A diferencia que XSS donde se aprovecha la confianza que el usuario tiene en el sitio web, CSRF se aprovecha de la confianza que un sitio web tiene en el usuario del navegador web.

- **Ataque DoS**: o ataque de denegación de servicio, por sus siglas en inglés *Denial of Service*, es un ataque a un sistema de computadoras o red que causa que un recurso o servicio sea inaccesible. Provoca la pérdida de conectividad de la red por el consumo del acho de banda de la red de la víctima, o por la sobrecarga de los recursos computacionales del sistema de la víctima.

- **HTTP**: corresponde a las siglas en inglés de *Hypertext Transfer Protocol,* es un protocolo de transferencia de datos orientado a transacciones y sin estado que sigue el esquema petición-respuesta entre un cliente y un servidor. Es el protocolo usado en cada transacción de la World Wide Web.

- **JavaScript**: lenguaje de programación interpretado de sintaxis similar a C, usado principalmente del lado del cliente como parte integrada en los navegadores web.

- **CSS**: *Cascading Style Sheets* en inglés, es un lenguaje usado para definir la presentación de un documento estructurado escrito en HTML, XML o XHTML.

- **PHP**: acrónimo que significa *PHP Hypertext Pre-procesor*, es un lenguaje de programación interpretado originalmente creado para la construcción de sitios web dinámicos. Puede ser desplegado en la mayoría de servidores web y en casi todos los sistemas operativos sin coste alguno gracias a su licencia de software libre.

- **PDO**: siglas de *PHP Data Objects*, es una capa de abstracción para el acceso a bases de datos para PHP 5. Con PDO se consigue hacer uso de las mismas funciones para hacer consultas y obtener datos de distintos tipos de bases de datos a través de diferentes manejadores.

- **MySQL**: es un sistema de gestión de base de datos relacional, nacido y desarrollado como software libre, actualmente ofrece dos tipos de licencias: GNU GPL para cualquier proyecto que pueda cumplir con esta licencia, y una licencia de pago para incorporarlo en productos privativos.

- **InnoDB**: tecnología de almacenamiento de datos disponible para MySQL a partir de su versión 4. Se caracteriza por soportar transacciones, bloqueo de registro e integridad referencial (propiedades no disponibles en la tecnología por defecto MyISAM).

- **Apache**: servidor web HTTP de código abierto disponible para las plataformas mayoritarias. Es el servidor más utilizado y más aceptado de la red.

- **TCP/IP**: conjunto de protocolos de red en los que se basa Internet y que permiten la transmisión de datos entre computadores.

1.4. Referencias

- (IEEE STD 830-1998 "Guide to Software Requeriments Specifications")

- (http://www.wikipedia.org: La Wikipedia)

1.5. Visión global

En el resto de esta ERS se describe de forma general los factores que afectan al producto, y por tanto a sus requisitos, desde el punto de vista de su contexto: perspectiva y funciones (a grandes rasgos) del producto, características de los usuarios, restricciones y limitaciones que se imponen al desarrollo, y las suposiciones y dependencias establecidas.

Para finalizar, el documento se centra en detallar los requisitos específicos de la aplicación, de forma que los desarrolladores sean capaces de (1) diseñar el sistema que satisfaga estos requisitos, (2) planificar las pruebas y (3) validar su cumplimiento.

2. Descripción general

En este apartado se ofrece una visión general de la aplicación sin entrar en los detalles específicos de sus requisitos (que se definirán con detalle en el apartado siguiente). Se describe el contexto de los requisitos de manera que sean fácilmente entendibles a lo largo de la ERS.

2.1. Perspectiva del producto

El producto a desarrollar será un sistema que actuará de forma autónoma, independiente y no integrado con otro sistema mayor. Únicamente será necesario emplear un navegador web compatible y disponer de una conexión a Internet para acceder desde cualquier plataforma.

Aunque evidentemente el producto se construirá en base a las tecnologías elegidas, éstas últimas han sido seleccionadas por ser soportadas en las plataformas más importantes. Por tanto, será necesario alojar dicha aplicación en un servidor que soporte las tecnologías empleadas, así como un nombre de dominio para su direccionamiento y acceso a través de Internet.

2.2. Funciones del producto

Las funciones del producto se clasifican en el siguiente listado según el perfil de usuario que utiliza la aplicación:

- Administrador

  - **Insertar/Modificar/Borrar/Listar Zonas**: permite la gestión de las zonas con las que los inmuebles serán agrupados por localización. Servirán para definir filtros de búsqueda en el catálogo de inmuebles.

  - **Insertar/Modificar/Borrar/Listar Temporadas**: permite la gestión de periodos de tiempo para la definición de temporadas vacacionales. Se utilizarán para asignar precios a los inmuebles según estas temporadas.

  - **Insertar/Modificar/Borrar/Listar/Filtrar Tipos de precio**: permite la gestión de modalidades de precio para el alquiler de inmuebles (por persona, por semana, por mes, etc.). Junto con las temporadas, se definen las variantes de precio de un inmueble.

  - **Insertar/Modificar/Borrar/Listar Tipos de inmueble**: permite la gestión de los tipos de inmueble presentes en el catálogo. Servirán para definir filtros de búsqueda en el catálogo de inmuebles.

  - **Insertar/Modificar/Borrar/Listar/Filtrar Opciones**: permite la gestión de las opciones o servicios asociados a los inmuebles (piscina, terraza, número de camas dobles, internet, etc.).

  - **Insertar/Modificar/Borrar/Listar/Filtrar Inmuebles**: permite la gestión de los inmuebles del catálogo.

  - **Publicar/Ordenar Inmuebles**: permite definir la forma en que son vistos los inmuebles en el portal (por orden definido, destacado o last minute).

  - **Administrar galería de imágenes de Inmuebles**: permite la gestión de las imágenes y fotos de los inmuebles.

  - **Administrar reservas de Inmuebles**: permite una gestión básica de las reservas realizadas para los inmuebles.

  - **Asignar precios de Inmuebles**: permite definir los precios de alquiler de inmuebles por temporada y tipo de precio.

  - **Insertar/Modificar/Borrar/Ordenar Páginas**: permite la gestión de las páginas de contenido del portal y su estructura.

  - **Insertar/Modificar/Borrar Contenido:** permite la gestión del contenido de las páginas del portal.

  - **Login/Logout**: permite la autentificación del administrador y el cierre de sesión de administración.

- Visitante

  - **Navegar Portal**: permite al usuario visitar todas las páginas del portal y visualizar su contenido.

  - **Cambiar Idioma**: permite al usuario cambiar el idioma del portal visualizando la información específica para ese idioma.

  - **Listar/Buscar Inmuebles**: permite visualizar el catálogo de inmuebles según los filtros de búsqueda activos.

  - **Ver detalle de Inmuebles:** permite visualizar toda la información relativa a un inmueble concreto.

  - **Contactar**: permite al usuario enviar e-mails de contacto al administrador.

2.3. Características del usuario

En lo que a usuarios de la aplicación se refiere, la aplicación dará lugar a dos tipos de perfiles, que se diferencian en base a las funcionalidades que éstos serán capaces de utilizar. Éstos son el usuario *Visitante* y el usuario *Administrador*.

El usuario Visitante se caracteriza por ser un tipo de usuario anónimo, por lo que no requerirá ningún tipo de autentificación. El Visitante es el tipo de usuario más habitual de la aplicación, y representa al conjunto de usuarios que navegará por el portal en busca de información acerca de la isla de Sant'Antioco (o cualquier otro tipo de información disponible en el sitio web). En especial, el Visitante podrá consultar el catálogo de inmuebles, buscar inmuebles de su interés y acceder a la información específica de cada inmueble. En caso de desearlo, el Visitante podrá ponerse en contacto con la agencia para realizar una reserva, dejando sus datos personales según los formularios habilitados en el portal.

El usuario Administrador es un tipo especial de usuario, con control total sobre la aplicación, es decir, capaz de utilizar todas las funcionalidades disponibles. El usuario Administrador, para actuar como tal, debe autentificarse previamente en el sistema, facilitando un nombre de usuario y contraseña predefinidos en la aplicación. Como Administrador, tendrá disponible una zona privada para realizar sus tareas de gestión, las cuales se pueden separar en dos bloques:

- **Gestión del contenido del sitio**: donde se especifica la estructura del sitio web público y se insertan los contenidos que resulten de interés.

- **Gestión del catálogo de inmuebles**: donde se mantiene el listado de inmuebles en alquiler, así como toda la información relativa a cada inmueble.

Pese a que no es necesario poseer ningún tipo de conocimiento específico para el uso de la aplicación, es recomendable algún tipo de experiencia previa en la navegación de páginas web o conocimiento mínimo en informática. En caso del usuario Administrador, se hace indispensable un conocimiento más profundo del negocio de la agencia turística.

2.4. Restricciones

Para la utilización de la aplicación se requerirá de cualquier hardware o dispositivo con conexión a Internet (ordenador personal, Smartphone, tableta, PDA, etc.) que además disponga de un navegador web compatible para su acceso.

Deben adoptarse medidas para evitar problemas de seguridad críticos en este tipo de aplicaciones como la inyección SQL y los ataques XSS y CSRF. Aunque debido a su naturaleza, la aplicación puede verse afectada por otros problemas típicos asociados a Internet, por ejemplo cortes o caídas en el servicio, o ataques DoS o de denegación de servicio.

2.5. Supuestos y dependencias

La aplicación será accesible mediante la utilización de un navegador web (sin especificar ninguno en concreto) compatible con en protocolo HTTP 1.1, que sea capaz de procesar y tenga activo el JavaScript, y que pueda interpretar directivas CSS 2 para su correcta visualización. Por lo general, cualquier navegador mayoritario en la actualidad no debería experimentar ningún problema.

El servidor donde se aloje la aplicación tendrá que soportar las tecnologías utilizadas para su implementación, en concreto:

- PHP 5.2 o superior, con las extensiones PDO instaladas y activadas para el acceso a base de datos.
- MySQL 5 o superior como sistema de gestión de base de datos, con capacidad para crear tablas InnoDB para la integridad referencial.
- Apache 2.0 o superior como servidor HTTP, con el módulo de reescritura de direcciones activado.

3. Requisitos específicos

3.1. Interfaces externas

3.1.1. Interfaces de usuario

Como ya se ha visto antes, la aplicación distingue dos tipos de usuario: Administrador y Visitante. Según el tipo de usuario destinado a utilizar la aplicación se distinguen también diferentes interfaces de usuario. A continuación se detallan cada una de estas interfaces:

- Administrador

  El Administrador, para tener acceso a su zona de trabajo, debe autentificarse previamente en el sistema mediante un nombre de usuario y una contraseña. Por tanto, la aplicación debe proporcionar una pantalla donde introducir dichos datos. Esta pantalla se compone básicamente de un formulario centrado en la pantalla y sólo será visible en caso de que el administrador no esté aún autentificado. Podemos ver un boceto en la siguiente imagen.

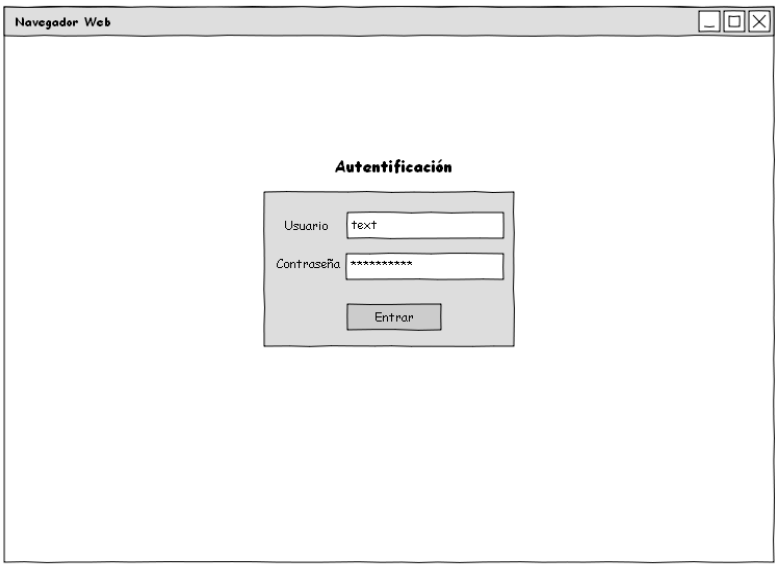


Ilustración 1: Pantalla de autentificación o "login"

Una vez autentificado, el Administrador tiene acceso a su interfaz privada de trabajo. Ésta se compone de dos zonas: una barra de menú superior con opciones desplegables y, un cuerpo central que puede variar en función del tipo de operación que se esté desempeñando.

En la barra de menú se tendrán disponibles las operaciones de gestión de la aplicación agrupadas por categorías. Cada categoría desplegará un menú con un conjunto de operaciones asociadas a dicha categoría. En las siguientes imágenes podemos apreciar estos detalles.

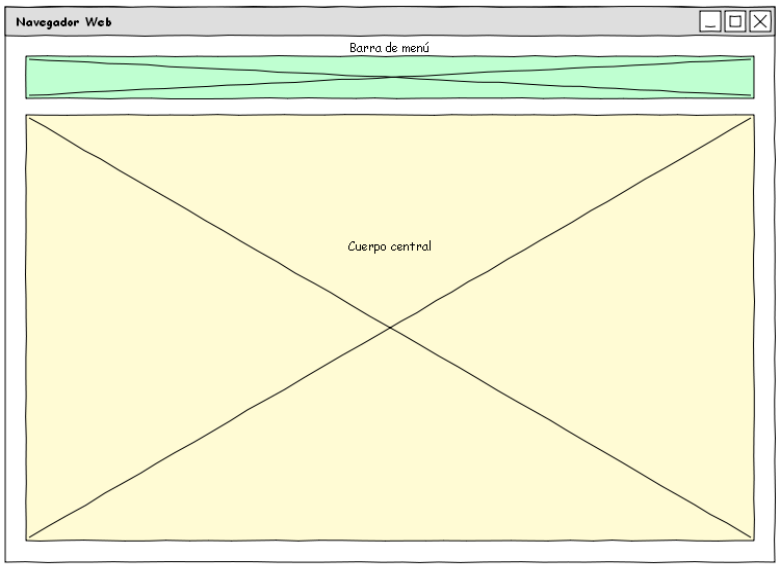


Ilustración 2: Estructura principal (“layout”) de la interfaz privada

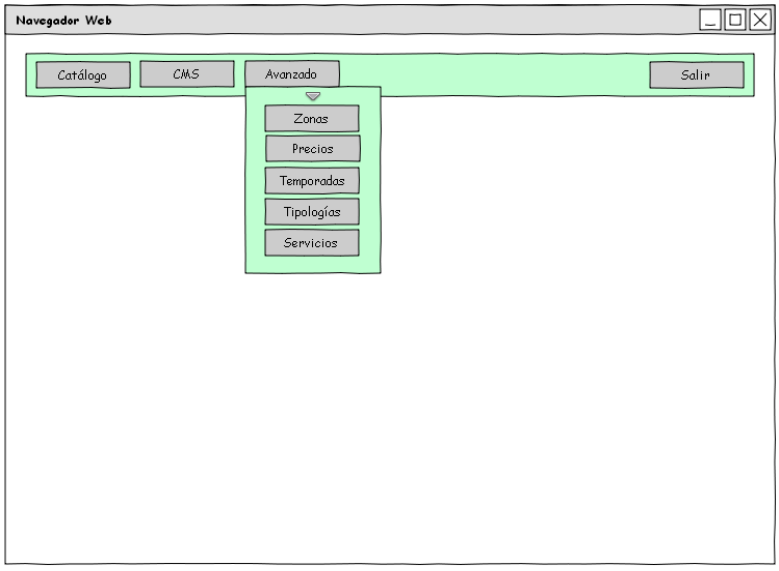


Ilustración 3: Barra de menú

El cuerpo central se organizará de dos formas diferentes en función de si la operación actual es del tipo Listado-Búsqueda o si es del tipo Creación-Edición.

Para las operaciones del tipo Listado-Búsqueda el cuerpo se compone de un listado de resultados en la parte izquierda, y una zona reservada a la derecha para introducir y/o activar filtros que alteren el listado de resultados. Por cada línea del listado puede haber accesos directos a operaciones sobre esa línea de resultados en particular. También al final del listado aparecerán los controles de la paginación de resultados, y pueden aparecer accesos a

operaciones sobre una selección de múltiples líneas del resultado. La siguiente figura sirve de aclaración.

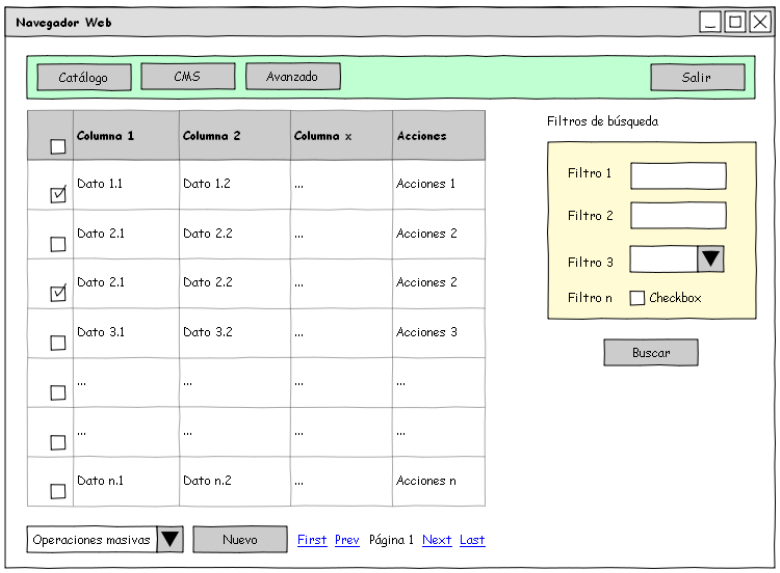


Ilustración 4: Interfaz para operaciones Listado-Búsqueda

Para las operaciones del tipo Creación/Edición el cuerpo central mostrará un formulario de introducción de datos, con las operaciones típicas de "guardar, cancelar, borrar" y otras al final del formulario. El formulario aparecerá en blanco para operaciones de Creación, mientras que éste aparecerá con los campos precargados para operaciones de Edición. Los campos del formulario pueden aparecer agrupados en bloques por categorías o sueltos (véase la siguiente figura).

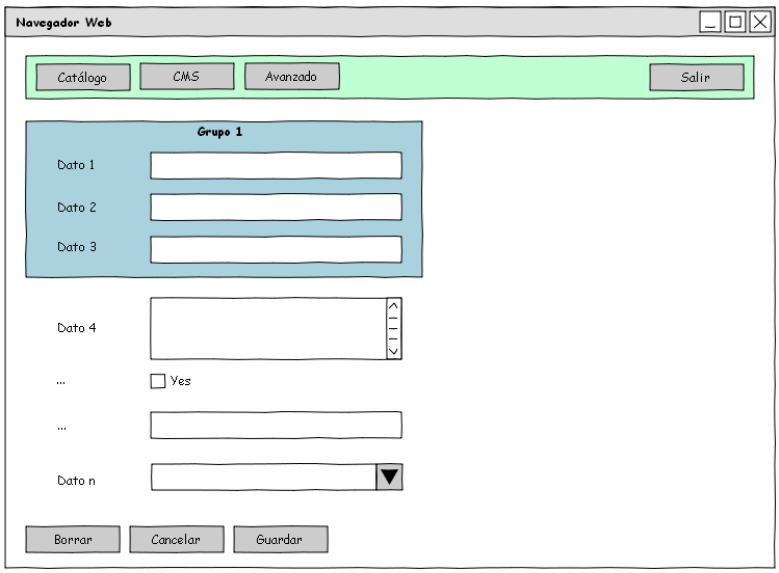


Ilustración 5: Interfaz para operaciones Creación-Edición

Para la edición del contenido del portal la interfaz será un poco más avanzada ya que la edición será "inline". Esto significa que el contenido se podrá modificar directamente desde la vista del portal, como si estuviéramos navegando por él cambiando "in situ" el contenido. Los bloques de texto e imágenes editables se resaltarán al situar el ratón encima de ellos. Al pulsar con el ratón sobre un bloque resaltado aparecerá un menú contextual que permitirá acceder a las opciones de edición específica del contenido.

- Visitante

Para el Visitante, las páginas del portal seguirán una estructura básica bastante común en la web: una cabecera con una barra de menú horizontal, un cuerpo central y un pie. El menú se construirá en función de la estructura del portal definida por el administrador, pero además se añadirán un buscador rápido y la opción del cambio de idioma. El pie de página simplemente contendrá información estática y algunos enlaces de interés predefinidos. El cuerpo central contendrá el contenido de cada página que se dispondrá en función de la plantilla elegida por el Administrador para cada página.

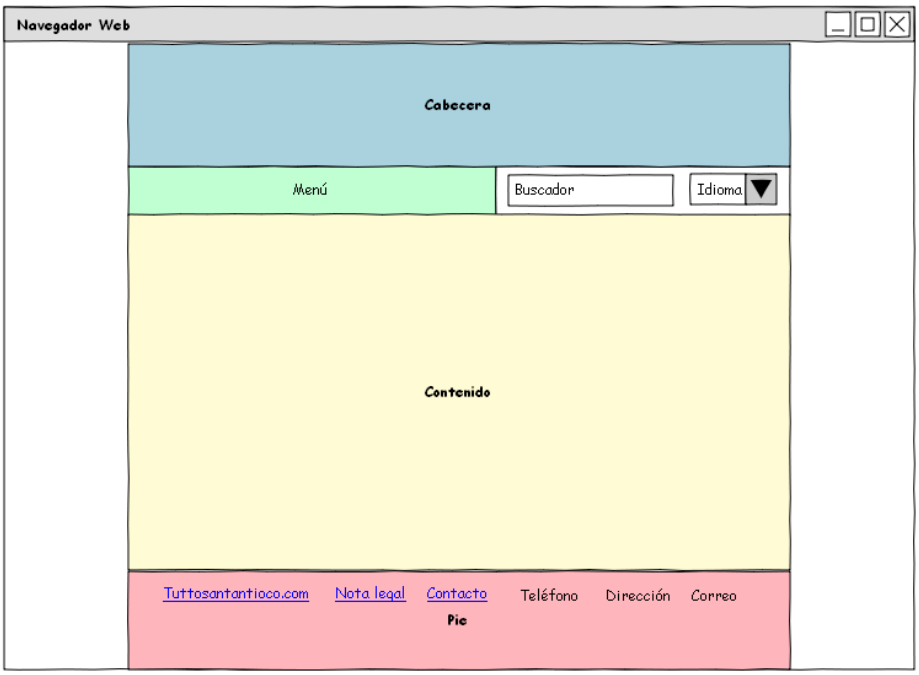


Ilustración 6: Estructura principal del portal público

El cuerpo adopta una estructura más especializada para la parte del portal donde el visitante puede consultar el catálogo de inmuebles (búsqueda, listado y detalles de inmuebles). Esta estructura se compone de: una columna a la izquierda que alberga los filtros de búsqueda de inmuebles, y el resto del espacio que se destina a visualizar el listado típico de inmuebles o los detalles de un inmueble. La columna de filtros se divide en tres partes: (1) filtros por disponibilidad, (2) filtros por localización y (3) filtros por tipo de inmueble. Véase la siguiente figura para entender la disposición de estos elementos.

Ilustración 7: Estructura del portal para la vista del catálogo

La vista que muestra en detalle la información de un inmueble será accesible desde el listado de inmuebles mediante un simple "clic" de ratón. Esta vista también se compondrá de una manera determinada: (1) título y nombre del inmueble, (2) descripción del inmueble, (3) galería fotográfica, (4) características y servicios del inmueble, (5) lista de precios y (6) disponibilidad del inmueble.

3.1.2. Interfaces Hardware

La aplicación se basa en el modelo cliente-servidor, por lo que se diferencian dos tipos de configuraciones hardware en función del rol que desempeñan.

Para la parte del cliente únicamente será necesario disponer de un dispositivo con conexión a Internet capaz de ejecutar un navegador web. Sería recomendable que el terminal cliente permitiera una resolución de pantalla de 1024x768, aunque no es indispensable puesto que la aplicación es accesible con resoluciones menores. Aún así, no se recomiendan resoluciones de pantalla menores de 800x600.

Para el servidor se requiere de una máquina conectada a Internet, con recursos suficientes para albergar y ejecutar los servicios asociados a la aplicación (Apache, MySQL y el intérprete PHP).

3.1.3. Interfaces Software

Para la persistencia de datos, la aplicación hará uso de MySQL (www.mysql.com) como sistema de gestión de base de datos a través de conexiones directas al puerto por defecto. El servidor donde se aloje la aplicación tendrá que disponer de una versión igual o superior a MySQL 5.0.

También, para la correcta interpretación del código fuente, el servidor tendrá que disponer de PHP (www.php.net), en versiones iguales o superiores a la 5.2.

Se requiere de Apache 2.0 (www.apache.org) o superior como servidor HTTP. Además de tener activo el procesamiento de ficheros de código PHP, también es necesaria la activación de su módulo de reescritura de direcciones.

Estas tecnologías están disponibles en las plataformas mayoritarias, por lo que no se impone ningún requisito o restricción para la elección del Sistema Operativo del servidor.

3.1.4. Interfaces de comunicación

Puesto que la aplicación es accesible a través de Internet (mediante el uso de un navegador web), el interfaz de comunicación es el modelo estándar TCP/IP, así como el protocolo HTTP (asociado a la World Wide Web) a nivel de aplicación.

3.2. Funcionales

A continuación se describen en detalle los requisitos funcionales de la aplicación clasificados por el tipo de usuario que interviene en su uso, tal y como se ha estado haciendo en el resto de esta ERS.

- **Insertar Zonas**
  - Actor: Administrador.
  - Introducción: El administrador debe ser capaz de dar de alta zonas o localizaciones donde estarán situados los inmuebles del catálogo.
  - Entradas: Debe indicarse el tipo de zona, la zona padre si la tuviera (sólo se permite una profundidad de dos hijos ó 3 niveles: ciudad, zona y semi-zona), y para cada idioma definido: el nombre, la descripción, unas palabras clave y un texto para la búsqueda rápida de inmuebles. El tipo de zona y el nombre son datos obligatorios.
  - Proceso: Se solicita la introducción de los datos de la zona. La nueva zona se almacena en la base de datos en caso de que la validación de los datos sea positiva.

- Salidas: Se muestra un mensaje indicando que la nueva zona ha sido guardada. En caso de no indicar algún campo requerido se muestra un mensaje de error indicando el campo obligatorio.

- **Modificar Zonas**

  - Actor: Administrador.

  - Introducción: El administrador debe ser capaz de modificar los datos de las zonas o localizaciones donde estarán situados los inmuebles del catálogo.

  - Entradas: Debe indicarse el identificador de la zona, el tipo de zona, la zona padre si la tuviera (sólo se permite una profundidad de dos hijos ó 3 niveles), y para cada idioma definido: el nombre, la descripción, unas palabras clave y un texto para la búsqueda rápida de inmuebles. El identificador, el tipo de zona y el nombre son datos obligatorios.

  - Proceso: Se visualizan los datos de la zona indicada y se solicita la introducción de los nuevos. Los nuevos datos de la zona se almacena en la base de datos en caso de que la validación de los mismos sea positiva. No es posible cambiar la zona padre de una zona con zonas “hijas”.

  - Salidas: Se muestra un mensaje indicando que la zona ha sido modificada. En caso de no indicar algún campo requerido se muestra un mensaje de error indicando el campo obligatorio.

- **Borrar Zonas**

  - Actor: Administrador.

  - Introducción: El administrador debe ser capaz de eliminar zonas donde estarán situados los inmuebles del catálogo.

  - Entradas: Debe indicarse el identificador de la zona (obligatorio).

  - Proceso: Se requerirá de confirmación de la operación. Como resultado la zona se elimina de la base de datos (sólo si ninguna propiedad está asociada a dicha zona). Si la zona borrada tiene zonas “hijas”, como resultado estas zonas también son eliminadas.

  - Salidas: Se muestra un mensaje indicando que la zona ha sido eliminada. Si la zona está asignada a una propiedad, se mostrará un mensaje indicando que la zona no puede borrarse.

- **Listar Zonas**

  - Actor: Administrador.

  - Introducción: El administrador debe ser capaz de listar las zonas donde estarán situados los inmuebles del catálogo que están dados de alta en el sistema.

  - Entradas: Ninguna.

  - Proceso: Se seleccionan de la base de datos todas las zonas dadas de alta.

- Salidas: Se muestra un listado en forma de árbol ordenado alfabéticamente.

- **Insertar Temporadas**

  - Actor: Administrador.

  - Introducción: El administrador debe ser capaz de dar de alta temporadas para los precios.

  - Entradas: Debe indicarse la fecha de inicio de temporada y la fecha de fin, y el título de la temporada para cada idioma definido. Todos los datos son obligatorios.

  - Proceso: Se solicita la introducción de los datos de la temporada. La nueva temporada se almacena en la base de datos en caso de que la validación de los datos sea positiva.

  - Salidas: Se muestra un mensaje indicando que la nueva temporada ha sido guardada. En caso de no indicar algún campo requerido se muestra un mensaje de error indicando el campo obligatorio. En caso que la fecha de inicio sea posterior a la de fin de temporada se muestra un mensaje indicando dicho error.

- **Modificar Temporadas**

  - Actor: Administrador.

  - Introducción: El administrador debe ser capaz de modificar los datos de las temporadas.

  - Entradas: Debe indicarse el identificador de la temporada, la fecha de inicio de temporada y la fecha de fin, y el título de la temporada para cada idioma definido. Todos los datos son obligatorios.

  - Proceso: Se visualizan los datos de la temporada indicada y se solicita la introducción de los nuevos. Los nuevos datos de la temporada se almacenan en la base de datos en caso de que la validación de los mismos sea positiva.

  - Salidas: Se muestra un mensaje indicando que la temporada ha sido modificada. En caso de no indicar algún campo requerido se muestra un mensaje de error indicando el campo obligatorio. En caso que la fecha de inicio sea posterior a la de fin de temporada se muestra un mensaje indicando dicho error.

- **Borrar Temporadas**

  - Actor: Administrador.

  - Introducción: El administrador debe ser capaz de eliminar temporadas de precios.

  - Entradas: Debe indicarse el identificador de la temporada (obligatorio).

  - Proceso: Se requerirá de confirmación de la operación. Como resultado la temporada se elimina de la base de datos (sólo si no se utiliza para asignar precios a inmuebles).

- Salidas: Se muestra un mensaje indicando que la temporada ha sido eliminada. Si la temporada se usa para asignar precios a algún inmueble se muestra un mensaje indicando que no es posible eliminar dicha temporada.

- **Listar Temporadas**

  - Actor: Administrador.

  - Introducción: El administrador debe ser capaz de listar las temporadas de precios que están dadas de alta en el sistema.

  - Entradas: Puede indicarse el campo de fecha por el que se ordenará el listado, y su sentido ascendente o descendente.

  - Proceso: Se seleccionan de la base de datos todas las temporadas dadas de alta.

  - Salidas: Se muestra un listado de las temporadas (ordenable por fechas) detallando su fecha de inicio y de fin.

- **Insertar Tipos de Precio**

  - Actor: Administrador.

  - Introducción: El administrador debe ser capaz de dar de alta categorías o tipos de precio.

  - Entradas: Debe indicarse el título del tipo de precio para cada idioma definido. Todos los datos son obligatorios.

  - Proceso: Se solicita la introducción de los datos del tipo de precio. El nuevo tipo se almacena en la base de datos en caso de que la validación de los datos sea positiva.

  - Salidas: Se muestra un mensaje indicando que el nuevo tipo de precio ha sido guardado. En caso de no indicar algún campo requerido se muestra un mensaje de error indicando el campo obligatorio.

- **Modificar Tipos de Precio**

  - Actor: Administrador.

  - Introducción: El administrador debe ser capaz de modificar los datos de los tipos de precio.

  - Entradas: Debe indicarse el identificador del tipo de precio, y el título del tipo de precio para cada idioma definido. Todos los datos son obligatorios.

  - Proceso: Se visualizan los datos del tipo de precio indicado y se solicita la introducción de los nuevos. Los nuevos datos del tipo se almacenan en la base de datos en caso de que la validación de los mismos sea positiva.

  - Salidas: Se muestra un mensaje indicando que el tipo de precio ha sido modificado. En caso de no indicar algún campo requerido se muestra un mensaje de error indicando el campo obligatorio.

- **Borrar Tipos de Precio**
  - Actor: Administrador.
  - Introducción: El administrador debe ser capaz de eliminar tipos de precios.
  - Entradas: Debe indicarse el identificador del tipo de precio (obligatorio).
  - Proceso: Se requerirá de confirmación de la operación. Como resultado el tipo de precio se elimina de la base de datos (sólo si no se está utilizando para asignar precio a algún inmueble del catálogo).
  - Salidas: Se muestra un mensaje indicando que el tipo ha sido eliminado. En caso de que se esté usando para asignar un precio de inmueble se muestra un mensaje indicando la imposibilidad de borrar el tipo de precio.
- **Listar Tipos de Precio**
  - Actor: Administrador.
  - Introducción: El administrador debe ser capaz de listar los tipos de precio que están dados de alta en el sistema.
  - Entradas: Pueden indicarse intervalos de fechas de creación y última modificación para filtrar los resultados.
  - Proceso: Se seleccionan de la base de datos todos los tipos de precio dados de alta en función de los intervalos de fechas introducidos.
  - Salidas: Se muestra un listado de los tipos de precio dados de alta que satisfacen los filtros utilizados.
- **Insertar Tipos de Inmueble**
  - Actor: Administrador.
  - Introducción: El administrador debe ser capaz de dar de alta tipos de inmuebles.
  - Entradas: Debe indicarse el título y la descripción del tipo de inmueble para cada idioma definido. Todos los datos son obligatorios.
  - Proceso: Se solicita la introducción de los datos del tipo de inmueble. El nuevo tipo se almacena en la base de datos en caso de que la validación de los datos sea positiva.
  - Salidas: Se muestra un mensaje indicando que el nuevo tipo de inmueble ha sido guardado. En caso de no indicar algún campo requerido se muestra un mensaje de error indicando el campo obligatorio.
- **Modificar Tipos de Inmueble**
  - Actor: Administrador.
  - Introducción: El administrador debe ser capaz de modificar los datos de los tipos de inmuebles.

- Entradas: Debe indicarse el identificador, y el título y descripción del tipo de inmueble para cada idioma definido. Todos los datos son obligatorios.

- Proceso: Se visualizan los datos del tipo de inmueble indicado y se solicita la introducción de los nuevos. Los nuevos datos del tipo se almacenan en la base de datos en caso de que la validación de los mismos sea positiva.

- Salidas: Se muestra un mensaje indicando que el tipo de inmueble ha sido modificado. En caso de no indicar algún campo requerido se muestra un mensaje de error indicando el campo obligatorio.

- **Borrar Tipos de Inmueble**

  - Actor: Administrador.

  - Introducción: El administrador debe ser capaz de eliminar tipos de inmueble.

  - Entradas: Debe indicarse el identificador del tipo de inmueble (obligatorio).

  - Proceso: Se requerirá de confirmación de la operación. Como resultado el tipo de inmueble se elimina de la base de datos (sólo si no existen inmuebles del tipo de inmueble en el catálogo del sistema).

  - Salidas: Se muestra un mensaje indicando que el tipo de inmueble ha sido eliminado. En caso de existir algún inmueble del tipo de inmueble indicado se notifica mediante un mensaje que el inmueble no puede ser eliminado.

- **Listar Tipos de Inmueble**

  - Actor: Administrador.

  - Introducción: El administrador debe ser capaz de listar los tipos de inmueble que están dados de alta en el sistema.

  - Entradas: Ninguna.

  - Proceso: Se seleccionan de la base de datos todos los tipos de inmueble dados de alta.

  - Salidas: Se muestra un listado ordenado alfabéticamente por el título.

- **Insertar Opciones**

  - Actor: Administrador.

  - Introducción: El administrador debe ser capaz de dar de alta opciones o servicios asociados a los inmuebles.

  - Entradas: Debe indicarse el grupo de la opción (interior, exterior o servicio), el tipo, si debe mostrarse siempre en el detalle del inmueble, el valor por defecto y para cada idioma definido: el nombre y la descripción. El grupo, el tipo, si es visible y el nombre son datos obligatorios.

  - Proceso: Se solicita la introducción de los datos de la opción. La nueva opción adicional se almacena en la base de datos en caso de que la validación de los datos sea positiva.

- Salidas: Se muestra un mensaje indicando que la nueva opción ha sido guardada. En caso de no indicar algún campo requerido se muestra un mensaje de error indicando el campo obligatorio.

- **Modificar Opciones**

  - Actor: Administrador.

  - Introducción: El administrador debe ser capaz de modificar los datos de opciones o servicios asociados a los inmuebles.

  - Entradas: Debe indicarse el identificador de la opción, el grupo de la opción (interior, exterior o servicio), el tipo, si debe mostrarse siempre, y el valor por defecto y para cada idioma definido: el nombre y la descripción. El identificador, el grupo, el tipo, si es visible y el nombre son datos obligatorios.

  - Proceso: Se visualizan los datos de la opción indicada y se solicita la introducción de los nuevos. Los nuevos datos de la opción adicional se almacena en la base de datos en caso de que la validación de los datos sea positiva.

  - Salidas: Se muestra un mensaje indicando que la opción ha sido modificada. En caso de no indicar algún campo requerido se muestra un mensaje de error indicando el campo obligatorio.

- **Borrar Opciones**

  - Actor: Administrador.

  - Introducción: El administrador debe ser capaz de eliminar opciones o servicios asociados a los inmuebles.

  - Entradas: Debe indicarse el identificador de la opción (obligatorio).

  - Proceso: Se requerirá de confirmación de la operación. Como resultado la opción se elimina de la base de datos. No se permite el borrado de la opción “número de plazas”, puesto que es utilizada para filtrar el listado.

  - Salidas: Se muestra un mensaje indicando que la opción ha sido eliminada. En caso de intentar borrar la opción “número de plazas” se muestra un mensaje indicando que no es posible.

- **Listar Opciones**

  - Actor: Administrador.

  - Introducción: El administrador debe ser capaz de listar las opciones o servicios asociados a los inmuebles que están dados de alta en el sistema.

  - Entradas: Pueden indicarse valores para el grupo, el valor por defecto, el tipo y si son siempre visibles para filtrar los resultados. También puede indicarse si se desea ordenar los resultados por grupo, tipo o visibilidad, en su sentido ascendente o descendente.

  - Proceso: Se seleccionan de la base de datos todas las opciones dadas de alta en función de los filtros introducidos.

- Salidas: Se muestra un listado de las opciones dadas de alta que satisfacen los filtros utilizados.

- **Insertar Inmuebles**

  - Actor: Administrador.

  - Introducción: El administrador debe ser capaz de dar de alta inmuebles del catálogo.

  - Entradas: Debe indicarse un código identificador, el tipo de inmueble, la zona donde está, si debe aparecer publicado, y para cada idioma definido: el título, una descripción corta, la descripción completa y un texto para búsqueda rápida de inmuebles. También habrá que indicar para cada opción adicional de alta en la base de datos su valor específico para el inmueble y si queremos que la opción sea siempre visible para el inmueble concreto. El tipo, la zona, el título y la descripción corta son datos obligatorios.

  - Proceso: Se solicita la introducción de los datos del inmueble. El nuevo inmueble se almacena en la base de datos en caso de que la validación de los datos sea positiva.

  - Salidas: Se muestra un mensaje indicando que el nuevo inmueble ha sido guardado. En caso de no indicar algún campo requerido se muestra un mensaje de error indicando el campo obligatorio.

- **Modificar Inmuebles**

  - Actor: Administrador.

  - Introducción: El administrador debe ser capaz de modificar los datos de los inmuebles del catálogo.

  - Entradas: Debe indicarse el identificador del inmueble, un código identificador, el tipo de inmueble, la zona donde está, si debe aparecer publicado, y para cada idioma definido: el título, una descripción corta, la descripción completa y un texto para búsqueda rápida de inmuebles. También habrá que indicar para cada opción adicional de alta en la base de datos su valor específico para el inmueble y si queremos que la opción sea siempre visible para el inmueble concreto. El identificador, el tipo, la zona, el título y la descripción corta son datos obligatorios.

  - Proceso: Se visualizan los datos del inmueble indicado y se solicita la introducción de los nuevos. Los nuevos datos del inmueble se almacenan en la base de datos en caso de que la validación de los datos sea positiva.

  - Salidas: Se muestra un mensaje indicando que el inmueble ha sido modificado. En caso de no indicar algún campo requerido se muestra un mensaje de error indicando el campo obligatorio.

- **Borrar Inmuebles**

  - Actor: Administrador.

- Introducción: El administrador debe ser capaz de eliminar inmuebles del catálogo.

- Entradas: Debe indicarse el identificador del inmueble (obligatorio).

- Proceso: Se requerirá de confirmación de la operación. Como resultado el inmueble se elimina de la base de datos.

- Salidas: Se muestra un mensaje indicando que el inmueble ha sido eliminado.

- **Listar Inmuebles**

  - Actor: Administrador.

  - Introducción: El administrador debe ser capaz de listar los inmuebles que están dados de alta en el sistema.

  - Entradas: Pueden indicarse valores para el código, el tipo, la zona y si están publicados para filtrar los resultados. De forma especial, también puede indicarse como filtro el número de plazas del inmueble y si se admiten animales. Por último, también puede indicarse si se desea ordenar los resultados por código o publicación, en su sentido ascendente o descendente.

  - Proceso: Se seleccionan de la base de datos todos los inmuebles dados de alta en función de los filtros introducidos.

  - Salidas: Se muestra un listado de los inmuebles dados de alta que satisfacen los filtros utilizados.

- **Ordenar Inmuebles**

  - Actor: Administrador.

  - Introducción: El administrador debe ser capaz de dar prioridad a los inmuebles del catálogo para ordenarlos dentro de los listados de resultados por importancia.

  - Entradas: Debe indicarse el identificador del inmueble, y si se quiere subir o bajar su prioridad (en magnitud de una unidad) o situarlo como el más prioritario.

  - Proceso: A partir del listado de inmuebles, el administrador indica para un inmueble si desea subirlo, bajarlo una posición o situarlo en la cima del listado.

  - Salidas: El listado de resultados aparece reordenado con el inmueble situado en su nueva posición.

- **Agregar imágenes a la galería de imágenes del Inmueble**

  - Actor: Administrador.

  - Introducción: El administrador debe ser capaz de agregar imágenes a la colección de imágenes de un inmueble del catálogo.

- Entradas: Debe indicarse las imágenes almacenadas en disco que quieren agregarse a la colección. Los ficheros seleccionados deben tener un formato válido de imagen y no pueden superar un tamaño de 2 MB.

- Proceso: Se muestra una lista de imágenes incluidas en la colección. El Administrador a través de una ventana de exploración de ficheros del sistema operativo selecciona las imágenes que desea agregar. Las imágenes son almacenadas en el sistema.

- Salidas: El listado de imágenes de la colección se renueva reflejando las nuevas imágenes agregadas. En caso que alguna de las imágenes no sean válidas se muestra un mensaje de error indicando el problema.

- **Eliminar imágenes de la galería de imágenes del Inmueble**

  - Actor: Administrador.

  - Introducción: El administrador debe ser capaz de eliminar imágenes a la colección de imágenes de un inmueble del catálogo.

  - Entradas: El identificador del inmueble y los identificadores de las imágenes de la galería que deben borrarse (obligatorios).

  - Proceso: Se muestra una lista de imágenes incluidas en la colección. El Administrador selecciona las imágenes de la lista que desea eliminar. Las imágenes son eliminadas del sistema tras confirmar la operación.

  - Salidas: El listado de imágenes de la colección se renueva sin las imágenes eliminadas.

- **Ocupar Inmueble**

  - Actor: Administrador.

  - Introducción: El administrador debe ser capaz de indicar la fecha que un inmueble tiene una reserva y va a ser ocupado.

  - Entradas: Debe indicarse la fecha de inicio de la ocupación, la fecha de fin y el identificador del inmueble (obligatorios). También se puede indicar un nombre y un comentario de la ocupación y si la reserva está confirmada.

  - Proceso: Se consultan las fechas de ocupación del inmueble y se muestra un calendario con estas fechas destacadas. El usuario selecciona la fecha de inicio y de fin y confirma la operación. Las fechas deben ser ininterrumpidas. Tras indicar la fechas, si se desea se pueden indicar el resto de datos.

  - Salidas: Se muestra el calendario con el nuevo rango de ocupación actualizado. En caso de errores se muestra un mensaje de error.

- **Desocupar Inmueble**

  - Actor: Administrador.

  - Introducción: El administrador debe ser capaz de eliminar fechas en que un inmueble tiene una reserva y va a ser ocupado.

- Entradas: Debe indicarse el identificador de la ocupación y el identificador del inmueble (obligatorios).

- Proceso: Se consultan las fechas de ocupación del inmueble y se muestra un calendario con estas fechas destacadas. El usuario selecciona el rango deseado y confirma la operación.

- Salidas: Se muestra el calendario con el nuevo rango de ocupación eliminado. En caso de errores se muestra un mensaje de error.

- **Agregar precio de Inmueble**

  - Actor: Administrador.

  - Introducción: El administrador debe ser capaz de dar de alta los precios para inmuebles específicos del catálogo.

  - Entradas: Debe indicarse el identificador del inmueble, una temporada, un tipo de precio, y un valor. Todos los datos son obligatorios, además en conjunto deben ser únicos para un inmueble dado.

  - Proceso: Se solicita la introducción de los datos del precio. El nuevo precio se almacena en la base de datos en caso de que la validación de los datos sea positiva.

  - Salidas: Se muestra un mensaje indicando que el nuevo precio ha sido guardado. En caso de no indicar algún campo requerido se muestra un mensaje de error indicando el campo obligatorio. En caso de existir ya en el sistema un precio para el conjunto de datos se muestra un error indicando la previa existencia del precio.

- **Modificar precios de Inmueble**

  - Actor: Administrador.

  - Introducción: El administrador debe ser capaz de modificar los datos de los precios de un inmueble del catálogo.

  - Entradas: El identificador del inmueble, el identificador del precio, la temporada, el tipo de precio y el valor. Todos los datos son obligatorios, además en conjunto deben ser únicos para un inmueble dado.

  - Proceso: Se visualizan los datos del precio indicado y se solicita la introducción de los nuevos. Los nuevos datos del precio se almacenan en la base de datos en caso de que la validación de los datos sea positiva.

  - Salidas: Se muestra un mensaje indicando que los datos del precio han sido modificados. En caso de no indicar algún campo requerido se muestra un mensaje de error indicando el campo obligatorio. En caso de existir ya en el sistema un precio para el conjunto de datos se muestra un error indicando la previa existencia del precio.

- **Eliminar precio de Inmueble**

  - Actor: Administrador.

- Introducción: El administrador debe ser capaz de eliminar los precios establecidos para inmuebles específicos del catálogo.

- Entradas: El identificador del inmueble, y el identificador del precio a eliminar. Todos los datos son obligatorios.

- Proceso: Se requerirá de confirmación de la operación. Como resultado el precio del inmueble se elimina de la base de datos.

- Salidas: Se muestra un mensaje indicando que el precio del inmueble ha sido eliminado.

- **Listar precios de Inmueble**

  - Actor: Administrador.

  - Introducción: El administrador debe ser capaz de listar los precios de los inmuebles que están dados de alta en el sistema.

  - Entradas: El identificador del inmueble.

  - Proceso: Se seleccionan de la base de datos todos los precios del inmueble dados de alta.

  - Salidas: Se muestra un listado de los precios del inmueble.

- **Insertar Página**

  - Actor: Administrador.

  - Introducción: El administrador debe ser capaz de dar de alta páginas de contenido del portal.

  - Entradas: Debe indicarse la página padre si la hubiera, la plantilla a utilizar para el contenido (obligatorio), si se va a publicar, si debe aparecer en el menú principal del portal, y para cada idioma definido: el título (obligatorio), la descripción y las palabras clave.

  - Proceso: Se solicita la introducción de los datos de la página. La nueva página se almacena en la base de datos en caso de que la validación de los datos sea positiva.

  - Salidas: Se muestra un mensaje indicando que la nueva página ha sido guardada. En caso de no indicar algún campo requerido se muestra un mensaje de error indicando el campo obligatorio.

- **Modificar Página**

  - Actor: Administrador.

  - Introducción: El administrador debe ser capaz de modificar los datos de las páginas de contenido del portal.

  - Entradas: Debe indicarse el identificador de la página, la página padre si la hubiera, la plantilla a utilizar para el contenido (obligatorio), si se va a publicar,

si debe aparecer en el menú principal del portal, y para cada idioma definido: el título (obligatorio), la descripción y las palabras clave.

- Proceso: Se visualizan los datos de la página indicada y se solicita la introducción de los nuevos. Los nuevos datos de la página se almacenan en la base de datos en caso de que la validación de los datos sea positiva.

- Salidas: Se muestra un mensaje indicando que los datos de la página han sido modificados. En caso de no indicar algún campo requerido se muestra un mensaje de error indicando el campo obligatorio.

- **Borrar Página**

  - Introducción: El administrador debe ser capaz de eliminar páginas del portal.

  - Entradas: Debe indicarse el identificador de la página (obligatorio).

  - Proceso: Se requerirá de confirmación de la operación. Como resultado la página se elimina de la base de datos.

  - Salidas: Se muestra un mensaje indicando que la página ha sido eliminada.

- **Ordenar Página**

  - Actor: Administrador.

  - Introducción: El administrador debe ser capaz de ordenar las páginas del menú del portal para que se visualice a su elección.

  - Entradas: El identificador de la página a mover, el identificador de la página respecto a la cual moveremos la anterior y la posición con respecto a la última.

  - Proceso: Se muestra un árbol representativo del menú. El Administrador seleccionará la página a mover y la arrastrara hasta una posición anterior o posterior de otra de las páginas existentes.

  - Salidas: Se muestra el árbol con las entradas reordenadas.

- **Insertar Texto de Página**

  - Actor: Administrador.

  - Introducción: El administrador debe ser capaz de insertar texto como contenido de una página del portal.

  - Entradas: El identificador de la página, el identificador del hueco de texto definido por la plantilla de la página y el texto a incluir.

  - Proceso: Se mostrará la página de contenido con los huecos de contenido disponibles según la plantilla definida. Se activará la edición de texto de un hueco de texto mediante un "clic" de ratón y se introducirá el texto elegido. El texto queda almacenado en la base de datos tras confirmar la operación.

  - Salidas: La página se muestra con el nuevo texto en su posición definida por la plantilla.

- **Modificar Texto de Página**

  - Actor: Administrador.

  - Introducción: El administrador debe ser capaz de modificar el texto de una página del portal.

  - Entradas: El identificador de la página, el identificador del hueco de texto definido por la plantilla de la página y el texto a modificar.

  - Proceso: Se mostrará la página de contenido con los huecos y textos de contenido disponibles según la plantilla definida. Se activará la edición de texto de un hueco o texto mediante un "clic" de ratón y se modificará el texto elegido. El texto queda modificado en la base de datos tras confirmar la operación.

  - Salidas: La página se muestra con el texto modificado en su posición definida por la plantilla.

- **Borrar Texto de Página**

  - Actor: Administrador.

  - Introducción: El administrador debe ser capaz de eliminar el texto de una página del portal.

  - Entradas: El identificador de la página, el identificador del hueco de texto definido por la plantilla de la página.

  - Proceso: Se mostrará la página de contenido con los huecos y textos de contenido disponibles según la plantilla definida. Se activará la edición de texto de un hueco o texto mediante un "clic" de ratón y se eliminará el texto elegido. El texto queda eliminado en la base de datos tras confirmar la operación.

  - Salidas: La página se muestra con el texto eliminado y su hueco libre en su posición definida por la plantilla.

- **Insertar Imagen de Página**

  - Actor: Administrador.

  - Introducción: El administrador debe ser capaz de insertar imágenes como contenido de una página del portal.

  - Entradas: El identificador de la página, el identificador del hueco de imagen definido por la plantilla de la página y la imagen a incluir. El fichero de imagen deber ser de un formato de imagen válido.

  - Proceso: Se mostrará la página de contenido con los huecos de contenido disponibles según la plantilla definida. Se activará la edición de imagen de un hueco de imagen mediante un "clic" de ratón, se seleccionará "cargar imagen" y se seleccionará la imagen a subir. La imagen se almacena en el servidor.

- Salidas: La página se muestra con la nueva imagen en su posición definida por la plantilla. En caso de problemas con la subida de la imagen se muestra un mensaje de error.

- **Retocar Imagen de Página**

  - Actor: Administrador.

  - Introducción: El administrador debe ser capaz de realizar retoques básicos de las imágenes de contenido de una página del portal.

  - Entradas: El identificador de la página, el identificador del hueco de imagen definido por la plantilla de la página y el identificador de la imagen.

  - Proceso: Se mostrará la página de contenido con los huecos de contenido disponibles según la plantilla definida. Se activará la edición de imagen de un hueco de imagen mediante un "clic" de ratón y se seleccionará "editar imagen". Se mostrará un editor con el que se podrá rotar, escalar y cambiar el color del fondo de la imagen. Tras realizar alguna de estas operaciones y confirmar la operación la imagen queda modificada en el servidor.

  - Salidas: La página se muestra con la imagen retocada en su posición definida por la plantilla.

- **Autentificarse como Administrador**

  - Actor: Administrador.

  - Introducción: El Administrador debe poder autentificarse en el sistema para tener acceso al área privada de administración del portal.

  - Entradas: El nombre de usuario y la contraseña de acceso.

  - Proceso: El sistema contrasta los datos de acceso con la base de datos. El sistema almacena los datos necesarios en el sistema de sesiones del servidor.

  - Salidas: Se muestra la interfaz de administración abierta en la página por defecto. Se muestra un mensaje de error en caso que los datos no se correspondan con los datos de acceso válidos.

- **Cerrar sesión de Administrador**

  - Actor: Administrador.

  - Introducción: El Administrador debe ser capaz de cerrar su sesión de administración.

  - Entradas: Ninguna.

  - Proceso: La sesión iniciada por el Administrador se borra del sistema de sesiones del servidor.

  - Salidas: Se muestra la pantalla de autentificación.

- **Navegar Portal**

- Actor: Visitante.
- Introducción: El Visitante debe ser capaz de visualizar las diferentes páginas del portal a través de los hipervínculos del mismo.
- Entradas: El identificador de la página visitada y el idioma preferido.
- Proceso: El sistema consulta el contenido de la página solicitada y construye el HTML asociado.
- Salidas: Se muestra la página solicitada en un formato válido para su visualización en un navegador.

- **Cambiar Idioma del Portal**
  - Actor: Visitante.
  - Introducción: El Visitante debe ser capaz de cambiar el idioma actual del portal.
  - Entradas: El identificador del idioma a cambiar.
  - Proceso: Se consultan los idiomas disponibles del portal y se almacena en sesión de servidor la preferencia del nuevo idioma.
  - Salidas: Se muestra la página actual con el contenido en el idioma seleccionado.

- **Listar Inmuebles del Portal**
  - Actor: Visitante.
  - Introducción: El Visitante debe ser capaz de visualizar los inmuebles que están dados de alta en el sistema.
  - Entradas: Pueden indicarse fechas de inicio y fin de ocupación, el tipo y la zona para filtrar los resultados.
  - Proceso: Se seleccionan de la base de datos todos los inmuebles dados de alta y con la publicación activa, en función de los filtros introducidos.
  - Salidas: Se muestra un listado de los inmuebles dados de alta que satisfacen los filtros utilizados.

- **Búsqueda rápida de Inmuebles**
  - Actor: Visitante.
  - Introducción: El Visitante debe ser capaz de realizar una búsqueda rápida de inmuebles por palabras clave.
  - Entradas: Palabras clave de búsqueda
  - Proceso: El sistema construye una cadena de texto de cada inmueble con todos los datos disponibles, y selecciona aquellos donde aparezcan las palabras clave introducidas.

- Salidas: Se muestra un listado de los inmuebles dados de alta que satisfacen el filtro de búsqueda por palabras clave.

- **Ver detalle de Inmueble**

  - Actor: Visitante.

  - Introducción: El Visitante debe ser capaz de visualizar toda la información de un inmueble de forma detallada.

  - Entradas: El identificador del inmueble.

  - Proceso: Se consulta de la base de datos toda la información asociada al inmueble seleccionado.

  - Salidas: Se muestra una página con los detalles del inmueble: descripciones, galerías, servicios, ocupación, etc.

- **Contactar**

  - Actor: Visitante.

  - Introducción: El Visitante debe ser capaz de contactar con el Administrador del portal en relación con los inmuebles.

  - Entradas: Datos personales del visitante, datos relativos a un alquiler concreto: fechas de ocupación, número de adultos, número de niños, habitaciones necesarias, zona de interés, etc.

  - Proceso: Con los datos introducidos por el Visitante se construye y envía un e-mail al Administrador.

  - Salidas: Se muestra un mensaje de información que confirma el envío del e-mail al Administrador.

3.3. Rendimiento

Pese a que no se imponen valores específicos en lo que al tiempo de respuesta se refiere, la carga y visualización de las páginas del portal han de ser lo suficientemente fluidas como para no entorpecer la experiencia del usuario. En general un 80% de las páginas visitadas deben cargarse en menos de 1 segundo.

Se impone el uso de un sistema de caché que permita agilizar el acceso a la base de datos y el envío del contenido de las páginas vistas.

3.4. Atributos

3.4.1. Seguridad

A continuación se especifican los factores que protegen al software del uso accidental o malicioso que provoca el acceso, la modificación, la destrucción o la revelación de datos privados o protegidos:

1. La contraseña de Administrador se almacena cifrada en la base de datos mediante un algoritmo no reversible (MD5), y se complementa con una "semilla" también cifrada (SHA1).

2. Sólo se contempla la existencia de un único usuario Administrador para evitar el acceso concurrente a la gestión de la información.

3. Los ficheros de configuración existentes (como por ejemplo el archivo que almacena la conexión con la base de datos) no son accesibles, bien porque se encuentran fuera del directorio raíz de la web, o porque están protegidos por el servidor web y no son facilitados por éste.

4. Todos los datos de entrada son escapados y validados previamente a su almacenamiento para evitar ataques, entre otros, ataques de inyección SQL.

5. Los formularios de entrada de datos se protegen con un "token", un campo de datos específico de la aplicación para evitar ataques CSRF.y XSS.

6. El código de la aplicación se separa en dos módulos diferenciados: un "frontend" que representa la parte pública del portal, y un "backend" que representa la zona privada de gestión. De esta forma el acceso al código o el uso de diferentes funciones está supeditado al módulo al que pertenecen.

3.4.2. Mantenimiento

Una vez puesta en marcha la aplicación, cualquier cambio en la información de la misma podrá ser realizado por el usuario Administrador desde la propia funcionalidad implementada, con la excepción del cambio del usuario y contraseña del usuario Administrador (que tendrá que hacerse directamente en la base de datos por la persona con permisos y privilegios para hacerlo).

Cualquier cambio que implique modificaciones en la estructura de base de datos o en el diseño de la aplicación, o la incorporación de nuevas funcionalidades deberán realizarse por el personal cualificado para tal fin. Se recomienda una previa familiarización con el sistema tras la lectura detenida del conjunto de este documento.

3.4.3. Portabilidad

En general no existen requisitos de portabilidad. Se deben utilizar tecnologías disponibles en una amplia gama de plataformas mayoritarias. Esto evita la dependencia de la aplicación hacia un sistema concreto, permitiendo que sea soportada por cualquiera de estas plataformas.

Capítulo III. Análisis

1. Introducción

En el capítulo anterior se han captado a partir del cliente las necesidades, funciones y atributos que la aplicación debe satisfacer. Esto permite acotar el proyecto en el ámbito de su contexto, y sirve como base fundamental para el desarrollo y validación de la aplicación.

En este capítulo se da un paso más hacia adelante, iniciando el ciclo de desarrollo para la conclusión del proyecto. En la fase de análisis se construye un modelo conceptual del sistema a implementar desde el punto de vista de la orientación a objetos. De manera gráfica se describen e identifican artefactos, funcionalidades y procesos que intervendrán en el sistema.

Para dicha descripción del sistema se utiliza el lenguaje de modelado UML (de sus siglas en inglés *Unified Modeling Language* ó *Lenguaje de Modelado Unificado*), que sin duda es uno de los recursos más utilizados en la actualidad. UML es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema, en ningún caso define un proceso o metodología de desarrollo de software. UML ofrece un estándar para describir un modelo del sistema proporcionando una serie de diagramas, de los cuales, se han usado tres para esta fase del desarrollo:

- Diagrama de casos de uso: ilustran la relación de los actores con los requisitos funcionales del sistema.

- Diagramas de actividad: permiten comprender qué acciones deben ocurrir y cuáles son las dependencias de comportamiento en los casos de uso.

- Diagrama de clases: describe la estructura de información del sistema y la relación entre sus entidades.

2. Diagramas de Casos de uso

A continuación se muestran los diferentes diagramas de casos de uso que describen la funcionalidad presente en el sistema implementado. Además, los diagramas de casos de uso representan las relaciones que existen entre actores (personajes, entidades u otros sistemas que participan en una funcionalidad) y funcionalidades del sistema, describiendo estas interacciones desde el punto de vista de los usuarios del mismo.

En el siguiente diagrama se pone de manifiesto la relación que existe entre los diferentes tipos de usuario del sistema.

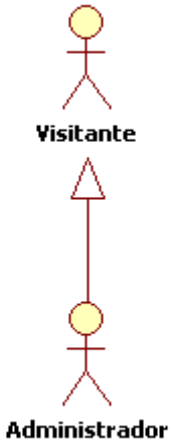


Ilustración 8: Relación entre Actores del sistema

Como puede observarse a partir de la figura anterior, el usuario Administrador es en realidad una especialización del usuario Visitante, ya que podrá realizar todas las acciones asociadas a éste último como si de un usuario Visitante se tratara. El Administrador posee funcionalidades específicas que lo caracterizan como un usuario privilegiado. Para entender mejor esta situación, en adelante definiremos los diagramas de casos de uso para cada uno de estos usuarios.

2.1. Casos de uso de Actor Visitante

El usuario Visitante es un usuario sin privilegios especiales dentro de la aplicación. Sus funciones básicamente se componen de aquellas que son necesarias para la correcta visualización y navegación del portal. Como Visitante, un usuario sólo necesita poder consultar a su antojo aquella información del portal que le resulte de su interés. Por tanto, en el siguiente diagrama podemos observar que acciones podrá llevar a cabo un usuario de este tipo.

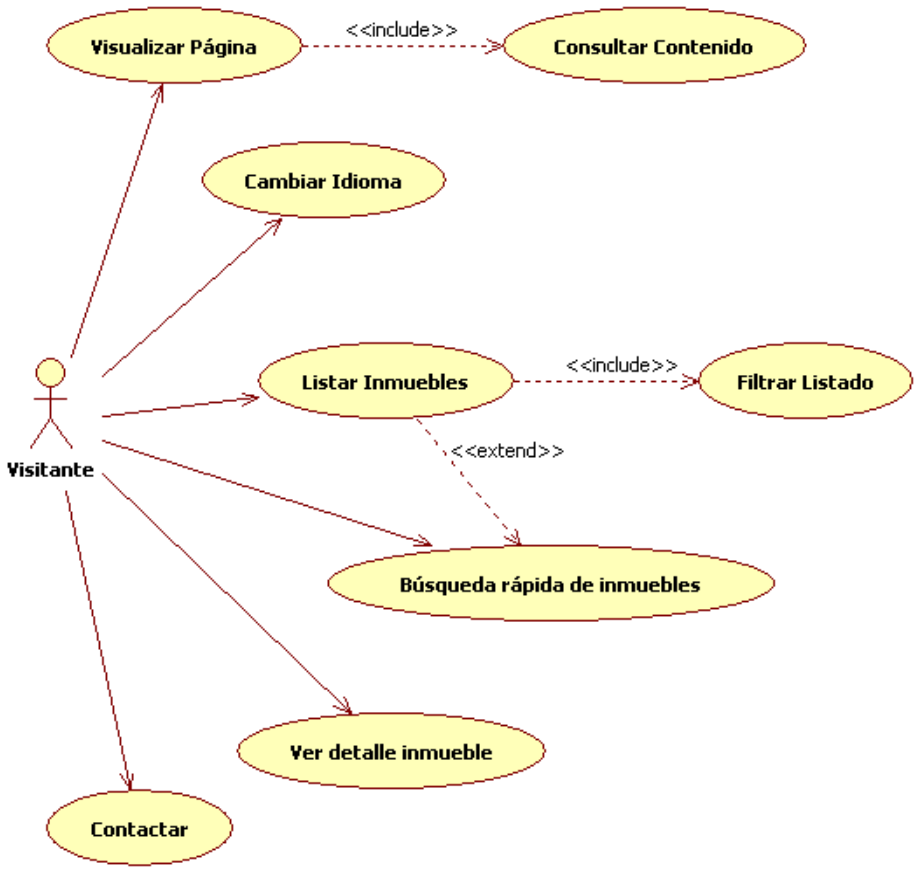


Ilustración 9: Casos de uso de Actor Visitante

2.2. Casos de uso de Actor Administrador

El usuario Administrador es el usuario de más alto nivel de la aplicación. Este usuario posee cualquier tipo de privilegio en la aplicación, además es el encargado del mantenimiento de toda la información gestionada por ésta. Como ya se ha comentado anteriormente, el usuario administrador será capaz de realizar cualquier acción de la que es capaz el Visitante, por lo que en el siguiente diagrama sólo se han expresado las acciones específicas de este usuario.

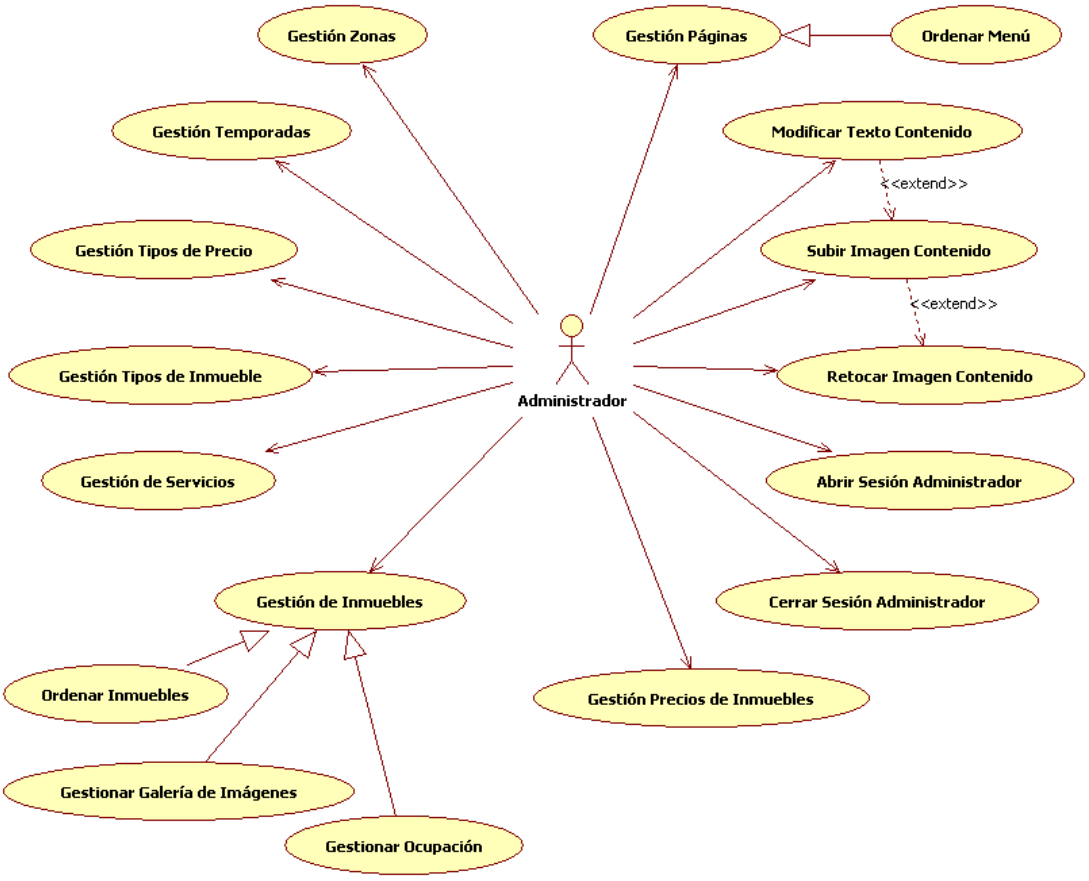


Ilustración 10: Casos de uso de Actor Administrador

El diagrama anterior se ha simplificado omitiendo los casos de uso habituales en las tareas de gestión de entidades. Es decir, se han omitido los casos de uso que representan el alta, la baja, la modificación y el listado de entidades. Por tanto, cada uno de los casos de uso de gestión, asociados al Administrador en el diagrama anterior, posee especializaciones de los casos de uso típicos. El diagrama de casos de uso del Administrador estaría completo con la expansión de sus tareas de gestión (en el siguiente diagrama puede verse la forma de realizar dicha expansión).

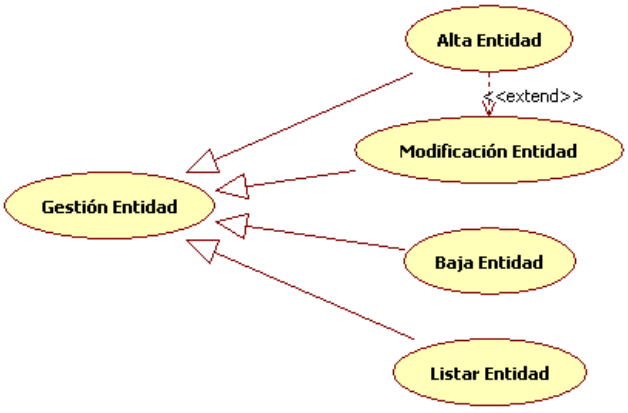


Ilustración 11: Expansión de casos de uso de gestión

3. Diagramas de Actividad

El diagrama de Actividad es una especialización del diagrama de Estados, organizado respecto de las acciones y utilizado para especificar métodos, procesos de negocios, y en el caso de este análisis, casos de uso. El diagrama de Actividad permite aumentar la expresividad del diagrama de Casos de uso, ayuda a comprender su complejidad interna mostrando las acciones y comportamientos que intervienen en una cierta funcionalidad.

Seguidamente se exponen los diferentes diagramas de Actividad para los casos de uso principales o más relevantes mostrados en el apartado anterior. En el caso del Administrador, se han creado diagramas de actividad genéricos para definir las tareas de gestión más comunes (alta, baja, modificación y listado) de las entidades presentes debido a su similitud de proceso.

3.1. Visualizar Página

En este diagrama puede observarse que antes de poder mostrar una página de contenido es necesario decorar la plantilla asociada a una página. Esto supone consultar el contenido en función del idioma de cada uno de los huecos definidos por la plantilla. Además puede observarse que es necesario descomponer la URL solicitada para obtener los datos necesarios de visualización de cualquier página de contenido.

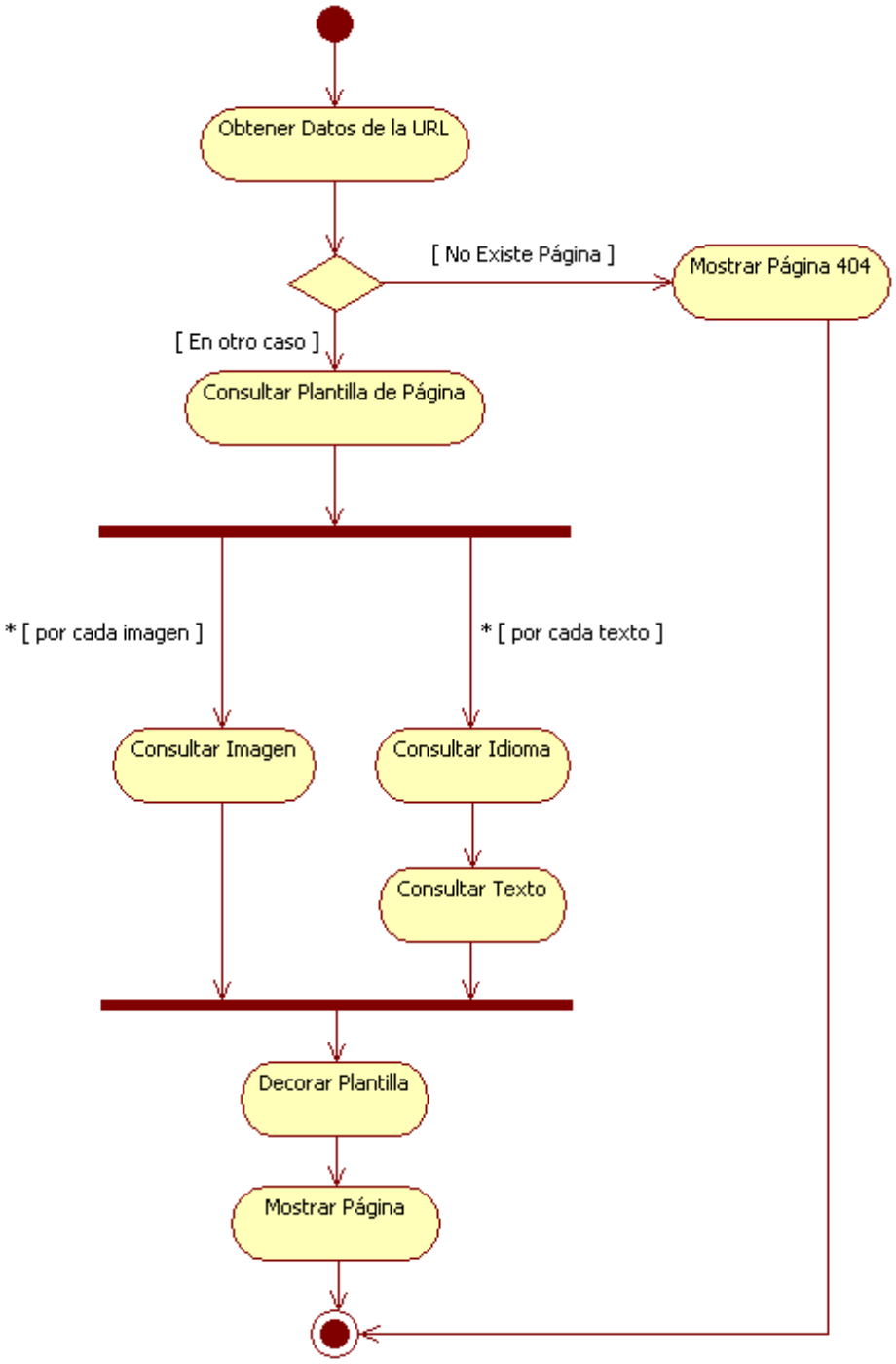


Ilustración 12: Diagrama de actividad de Visualizar Página

3.2. Cambiar Idioma

Cambiar el idioma es una acción simple, que almacena en sesión el valor del idioma preferido por el usuario (siempre y cuando se haya definido este idioma en el sistema).

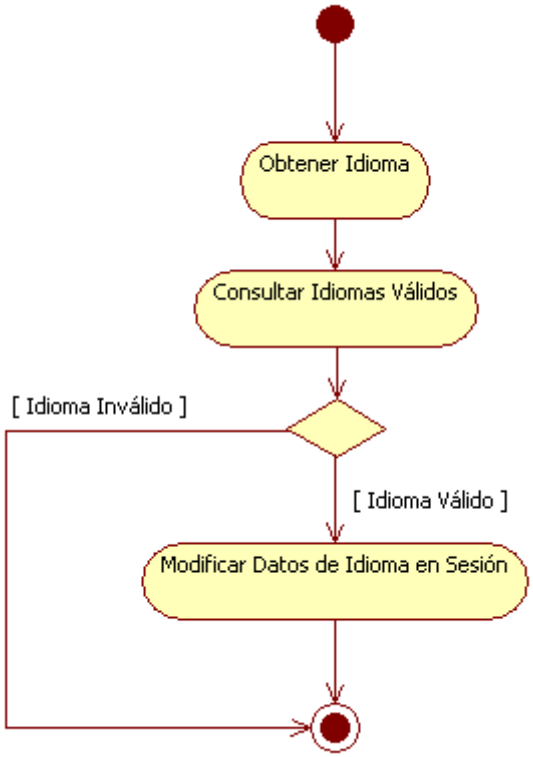


Ilustración 13: Diagrama de actividad de Cambiar Idioma

3.3. Listar Inmuebles (Visitante)

Para listar los inmuebles es necesario descomponer la URL solicitada para obtener el valor de los filtros utilizados en la búsqueda de resultados. Puede observarse, que para mostrar la página final, no sólo es necesario consultar el catálogo de inmuebles para crear el listado, sino que además debe construirse la sección de filtros de forma acorde a los filtros usados.

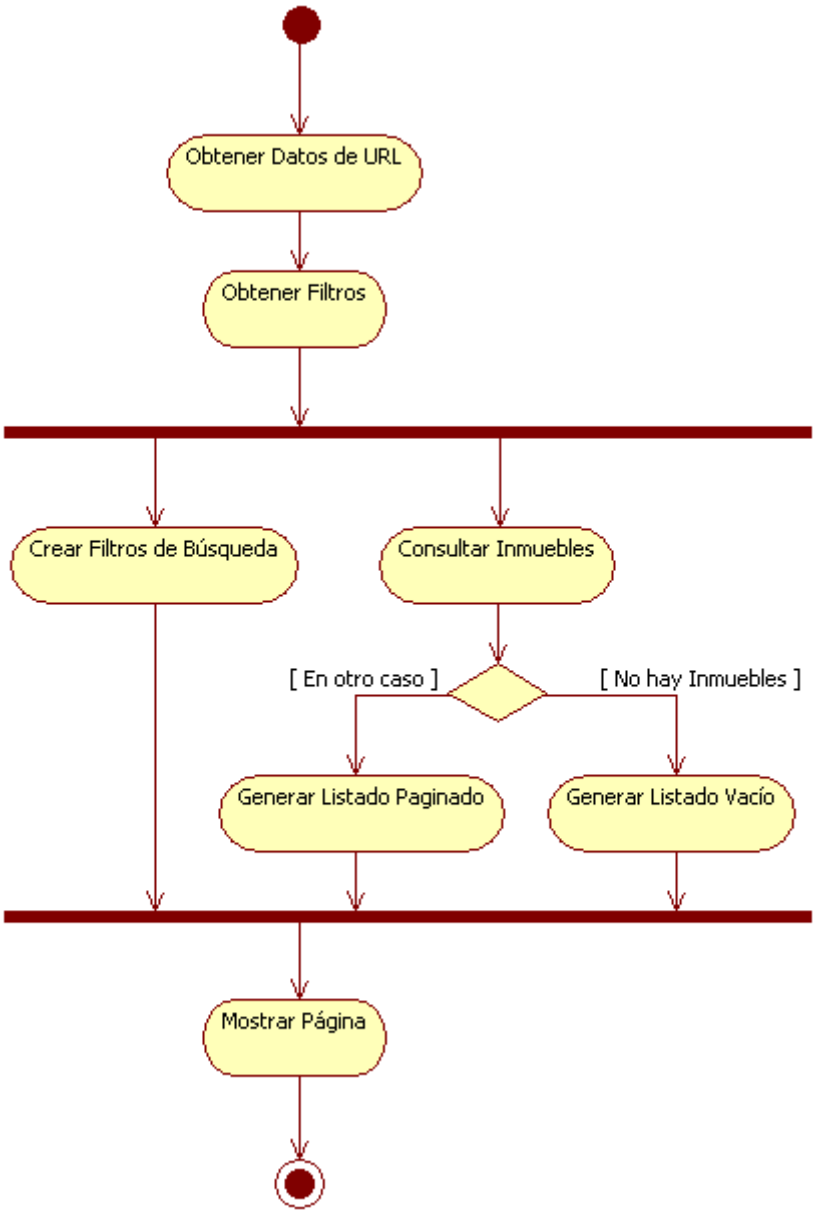


Ilustración 14: Diagrama de actividad de Listar Inmuebles (Visitante)

3.4. Búsqueda Rápida de Inmuebles

La búsqueda rápida de inmuebles es una especialización o extensión del caso de uso anterior. La búsqueda rápida añade un filtro extra para la creación del listado a partir de las palabras clave obtenidas de la llamada HTTP.

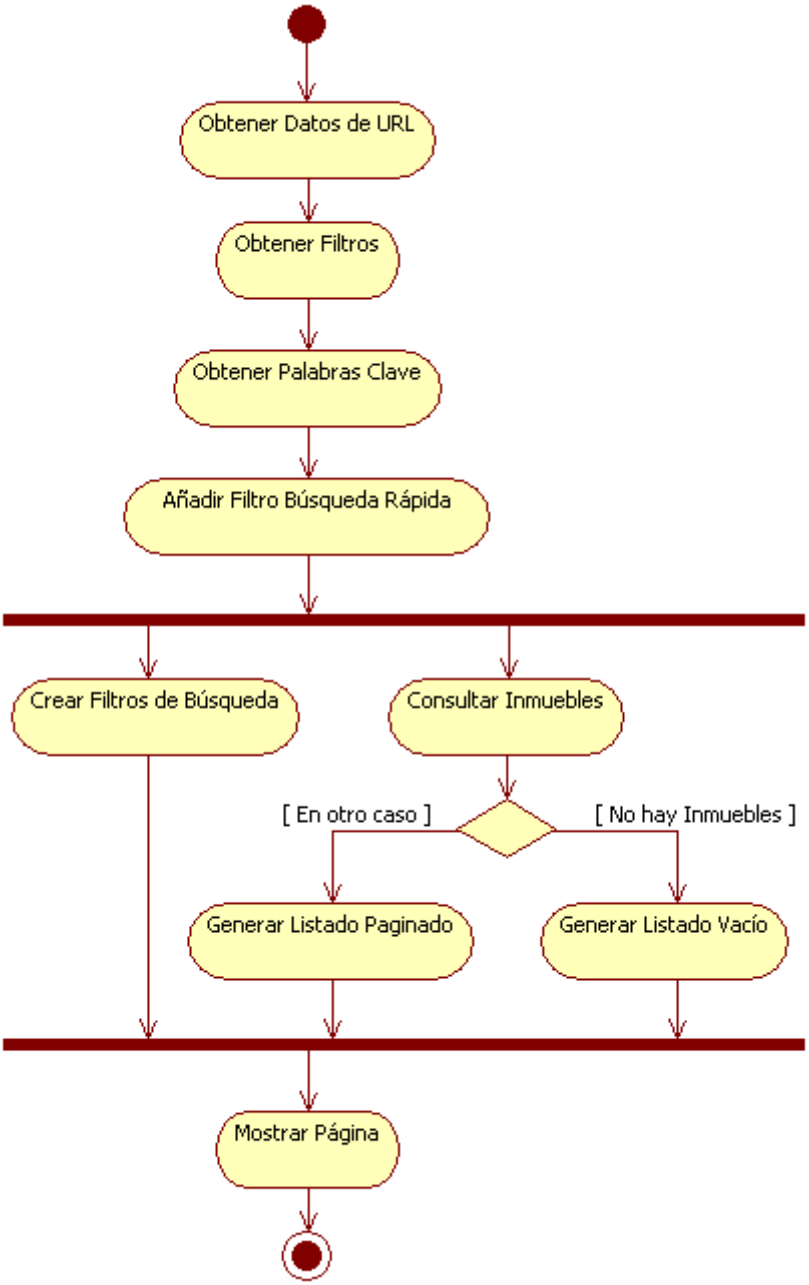


Ilustración 15: Diagrama de actividad de Búsqueda Rápida de Inmuebles

3.5. Ver detalle de Inmueble

En este diagrama puede observarse, como hemos visto antes, que es necesario descomponet la URL solicitada para validar la existencia del inmueble deseado y consultar su información. Además se pone de manifiesto que para mostrar la página también es necesario construir la sección de filtros de búsqueda.

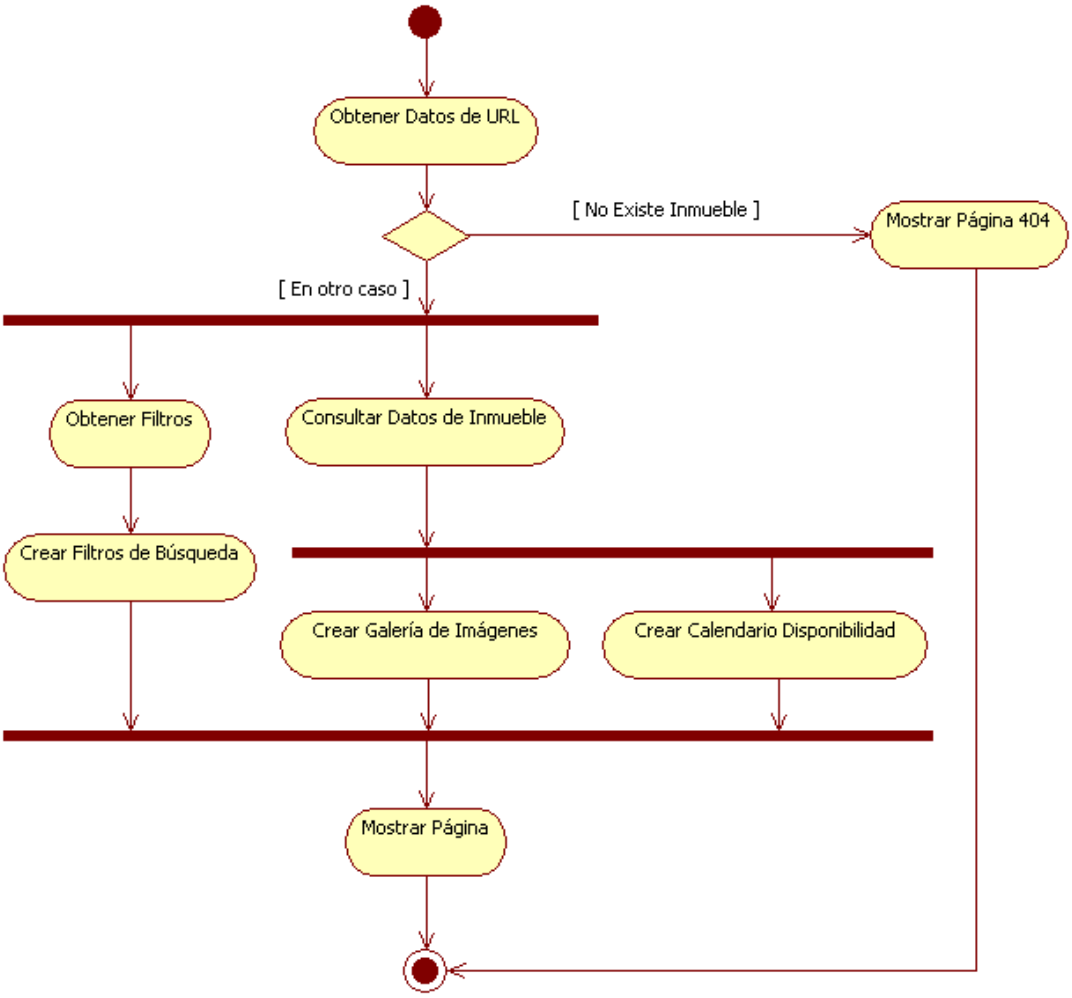


Ilustración 16: Diagrama de actividad de Ver detalle de Inmueble

3.6. Contactar

Contactar supone el envío de un e-mail a partir de los datos de un formulario. Este diagrama pone de manifiesto esta circunstancia además de su proceso de validación de los datos de entrada.

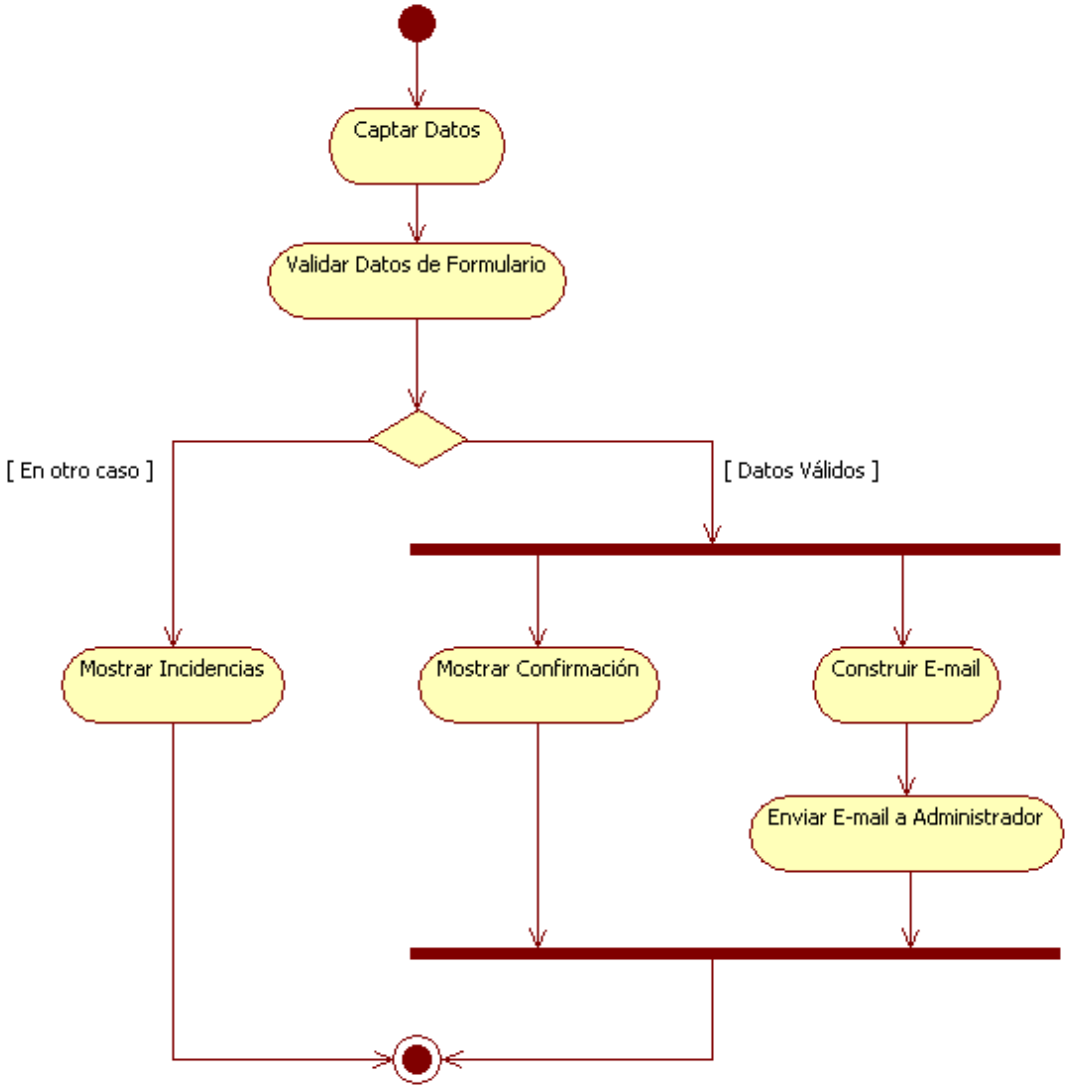


Ilustración 17: Diagrama de actividad de Contactar

3.7. Alta de Entidades (Genérico)

En este diagrama se observa como la creación de entidades es un proceso sencillo, que consiste en la captación y validación de los datos de entrada del formulario de alta.

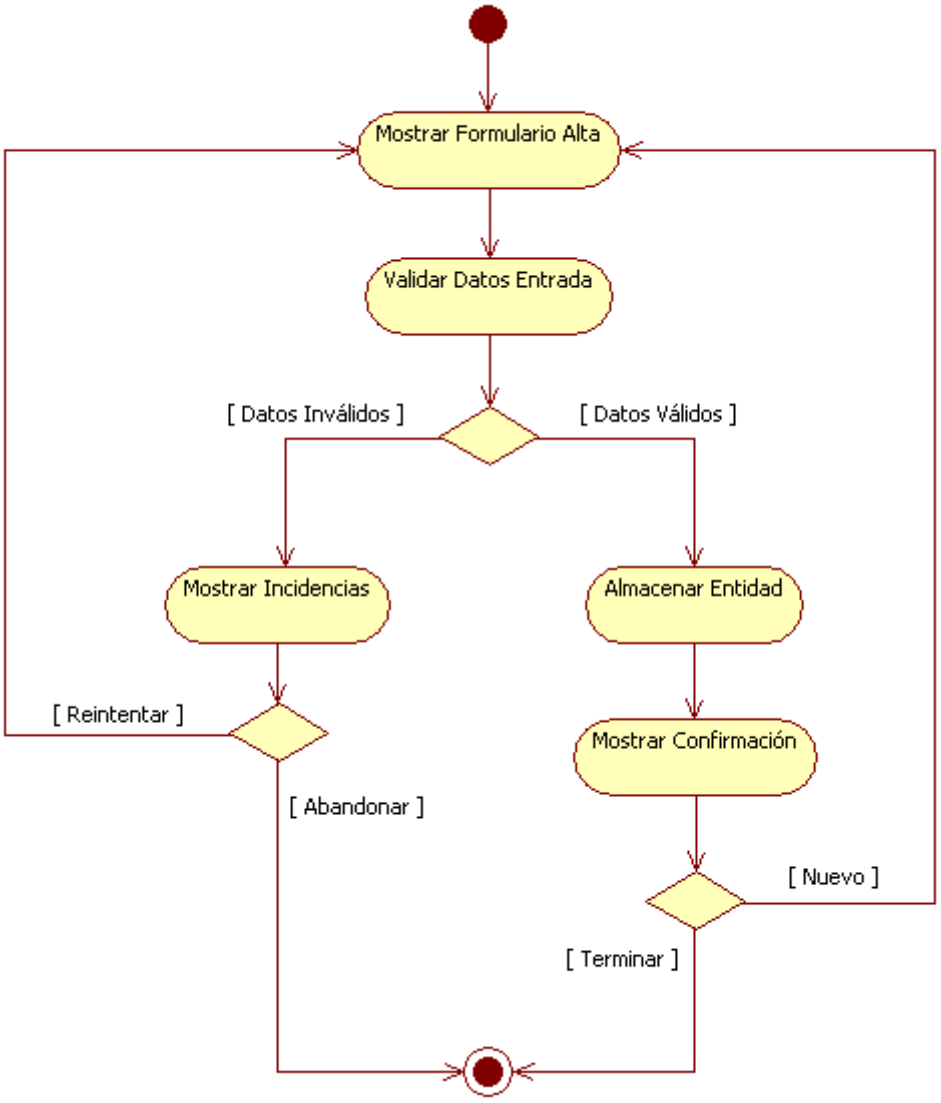


Ilustración 18: Diagrama de actividad de Alta de Entidades (Genérico)

3.8. Baja de Entidades (Genérico)

El proceso de baja de una entidad es muy simple, constando únicamente de una confirmación previa para evitar borrados accidentales.

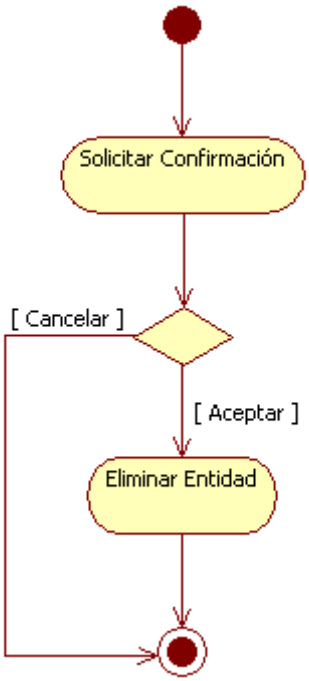


Ilustración 19: Diagrama de actividad de Baja de Entidades (Genérico)

3.9. Modificación de Entidades (Genérico)

La modificación de entidades es un proceso análogo al alta, sólo que en este caso no se produce la creación de una nueva entidad, sino que se modifican los datos de una entidad existente.

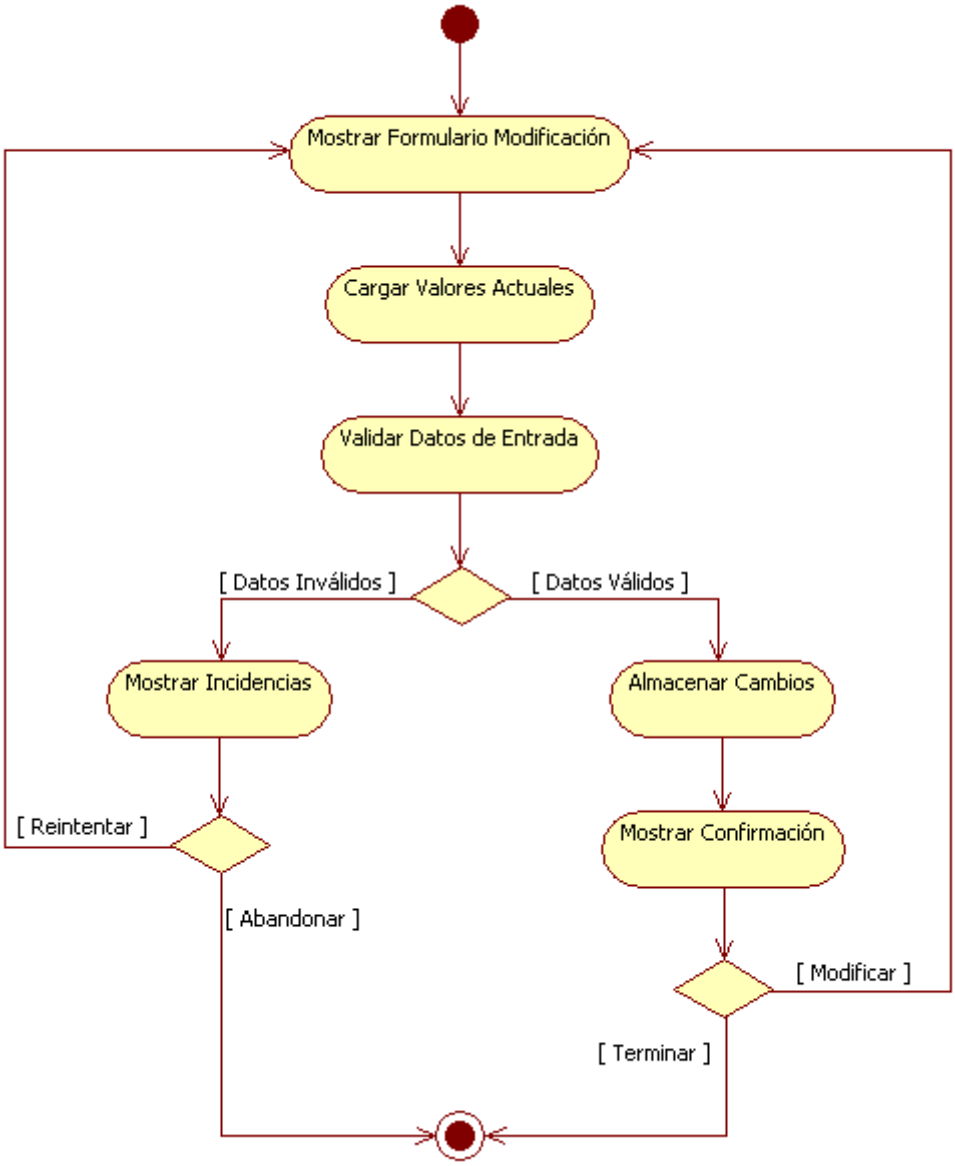


Ilustración 20: Diagrama de Modificación de Entidades (Genérico)

3.10. Listar Entidades (Genérico)

El proceso de listar entidades comporta el filtrar los resultados en caso de existir parámetros de búsqueda. Además de mostrar el listado correctamente, se debe visualizar la zona de filtros que pueden utilizarse, lo que queda reflejado en el siguiente diagrama.

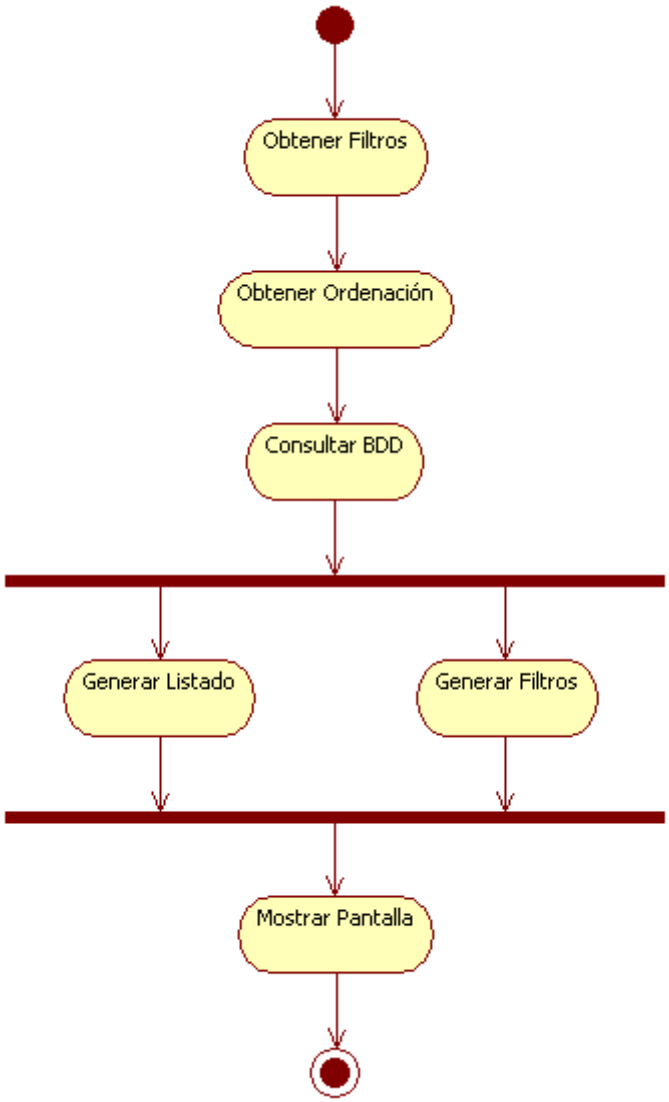


Ilustración 21: Diagrama de actividad de Listar Entidades (Genérico)

3.11. Abrir Sesión Administrador

En este diagrama puede observarse la necesidad de autentificar los datos del Administrador para dar acceso a la zona de administración, así como la activación del usuario privilegiado en la sesión del servidor.

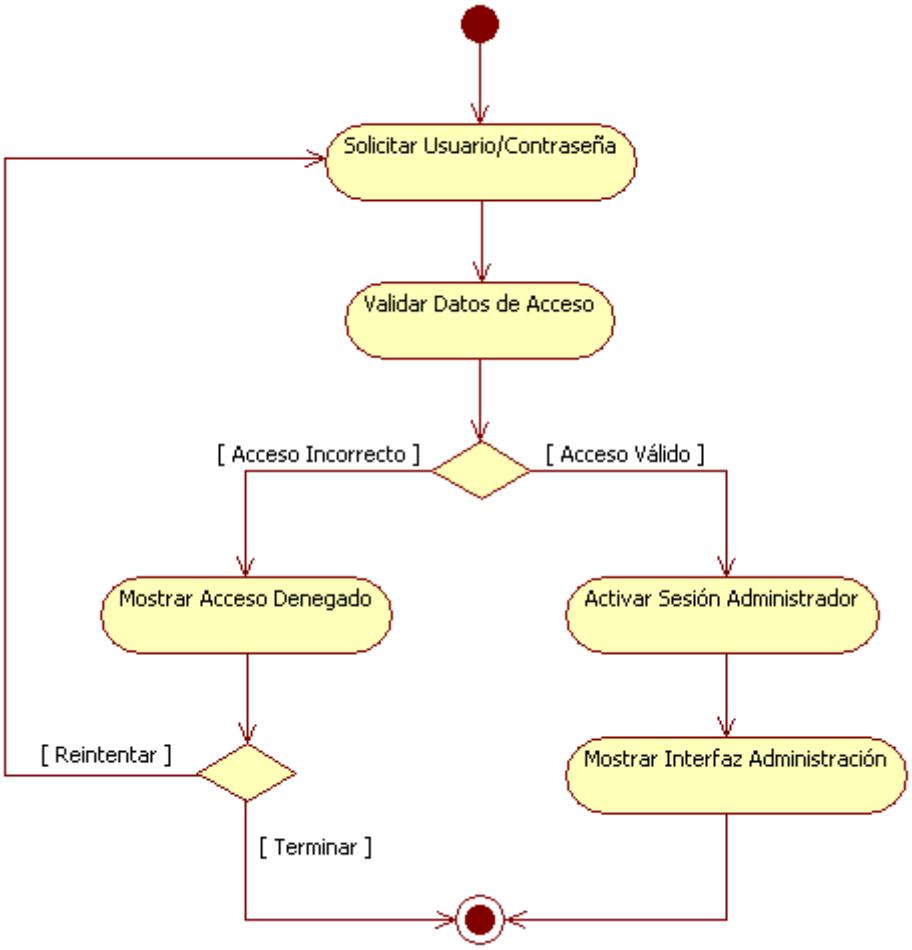


Ilustración 22: Diagrama de actividad de Abrir Sesión Administrador

3.12. Modificar Texto de Contenido de Página

Los contenidos de una página del portal se modifican insitu, durante la misma visualización de la página. Esto significa que el administrador puede modificar los contenidos y visualizarlos de inmediato mientras navega por el portal. El siguiente diagrama intenta reflejar esta situación, mostrando la existencia de una interfaz específica para estas acciones que se activa unitariamente por cada elemento de contenido.

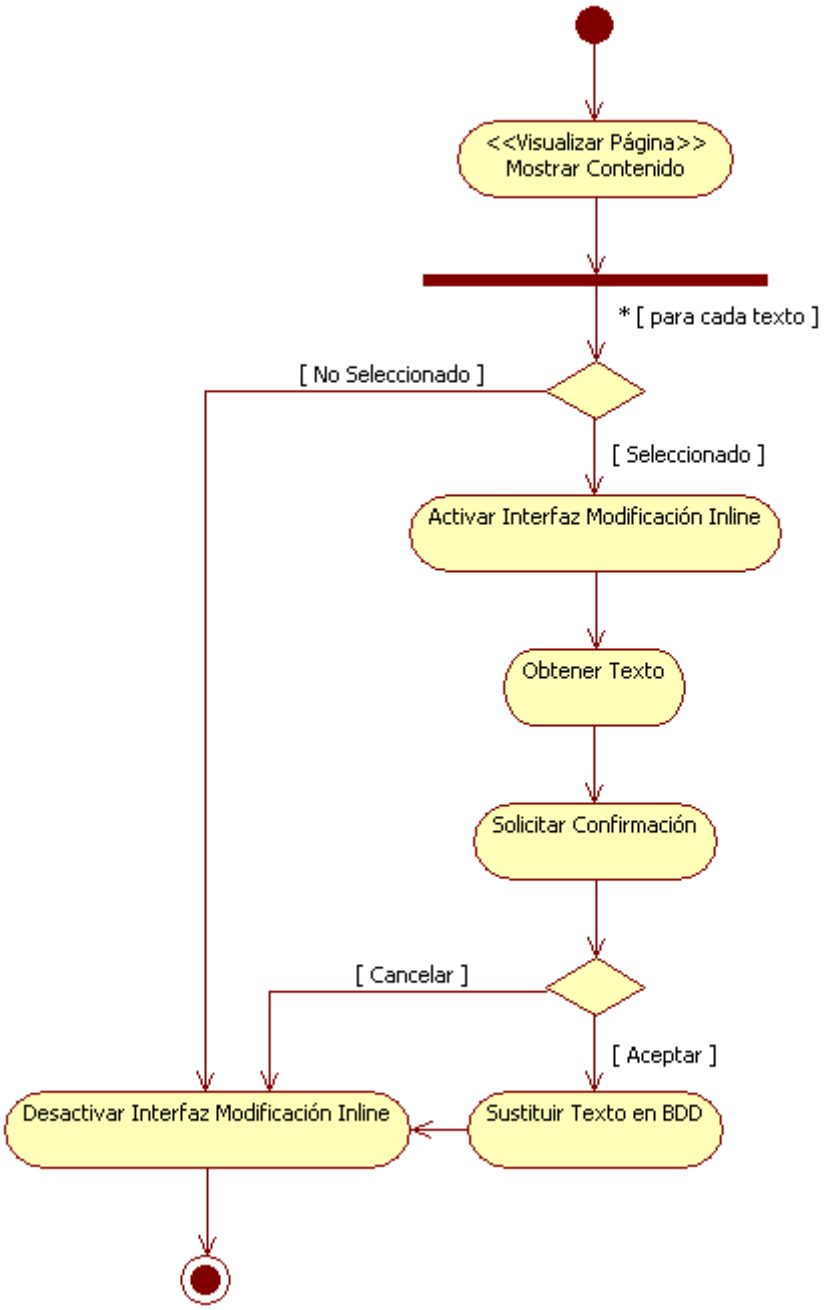


Ilustración 23: Diagrama de Modificar Texto de Contenido de Página

3.13. Subir Imagen de Contenido de Página

Como en el caso anterior, la carga de imágenes es un tipo de función que se realiza insitu con su interfaz propia. Pese a sus similitudes, este diagrama aporta las diferencias en el proceso debido a la naturaleza diferente del contenido.

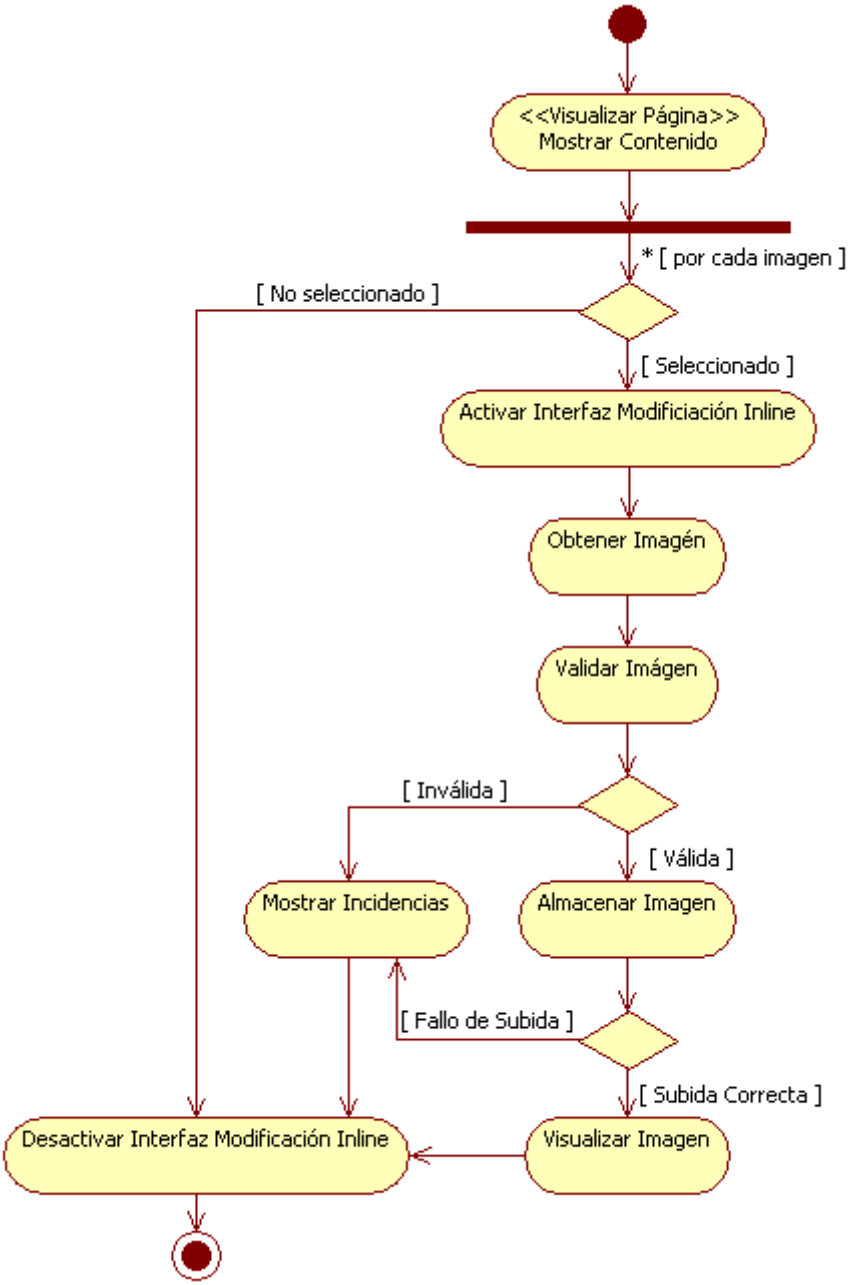


Ilustración 24: Diagrama de Subir Imagen de Contenido de Página

3.14. Retocar Imagen de Contenido de Página

Retocar una imagen permite la optimización y la aplicación de transformaciones a las imágenes del contenido. Como puede observarse en el siguiente diagrama, se construye una ventana de interfaz especial para realizar dichas acciones. Por lo demás el proceso es muy similar a las funcionalidades anteriores de modificación in situ.

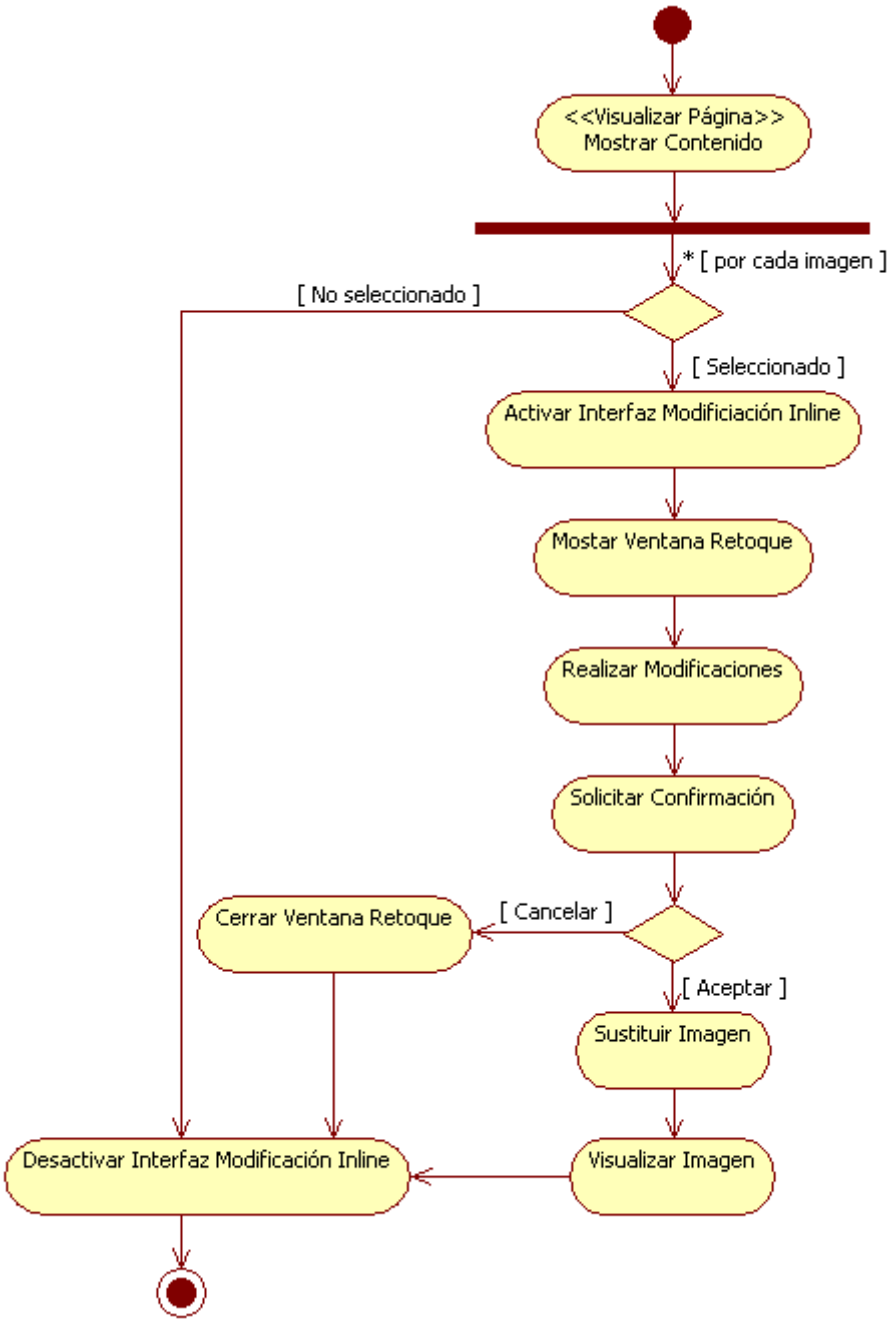


Ilustración 25: Diagrama de Retocar Imagen de Contenido de Página

3.15. Gestionar Ocupación de Inmueble

Como se puede inferir del siguiente diagrama, la gestión de la ocupación es un proceso que libera u ocupa un inmueble en cierto intervalo de tiempo. Es necesario indicar intervalos de fecha ininterrumpidos, es decir no puede haber fechas con ocupación y sin ocupación en la elección de los intervalos a gestionar.

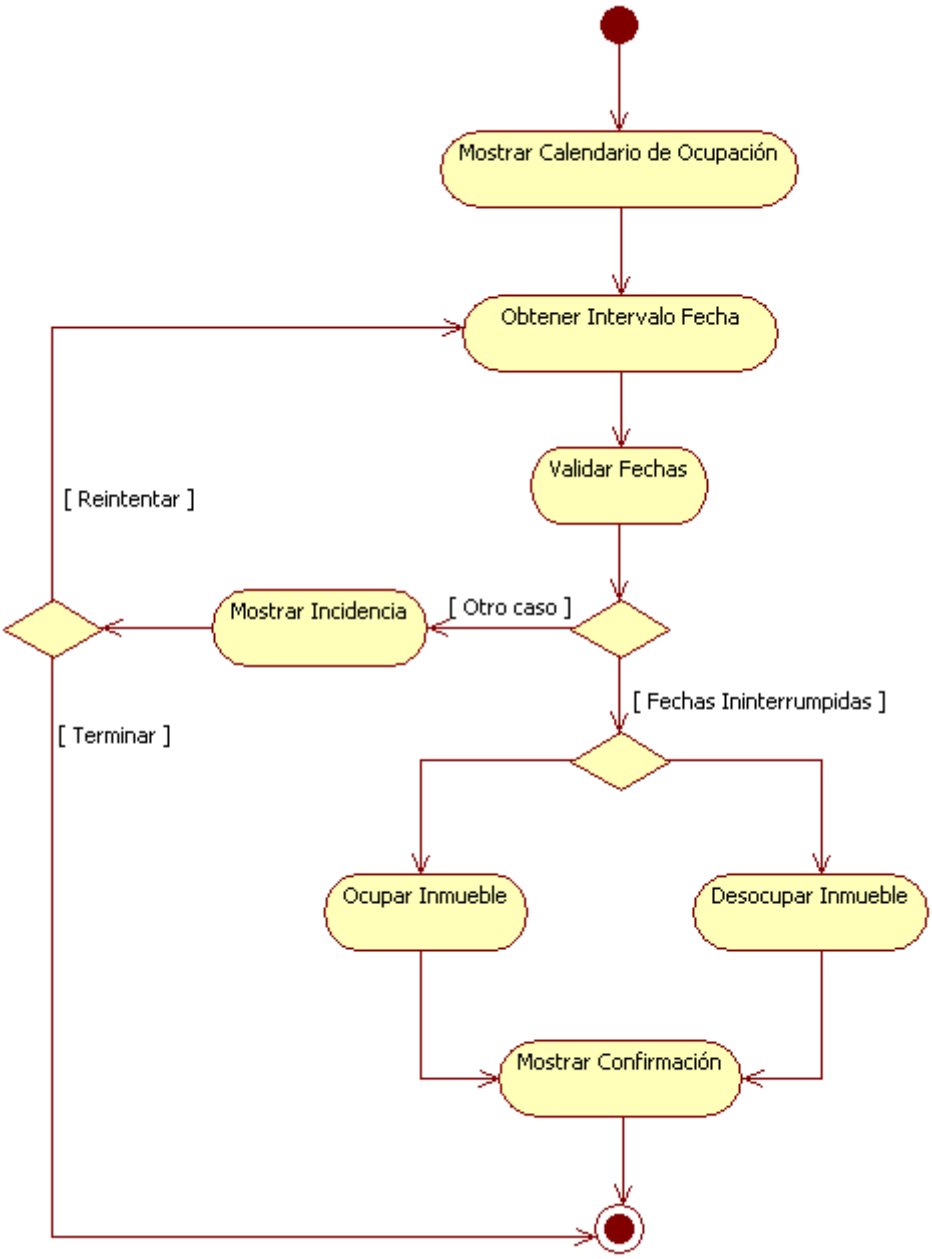


Ilustración 26: Diagrama de actividad de Gestionar Ocupación de Inmuebles

4. Diagrama de Clases

Mediante el uso del diagrama de clases se describe la estructura estática de un sistema mostrando sus clases, atributos y las relaciones que existen entre ellos. Los sistemas se diseñan sobre la abstracción de los objetos reales que intervienen en esos sistemas, en consecuencia las clases del diagrama de clases tratan de identificar dichos objetos de la realidad junto con sus características o atributos.

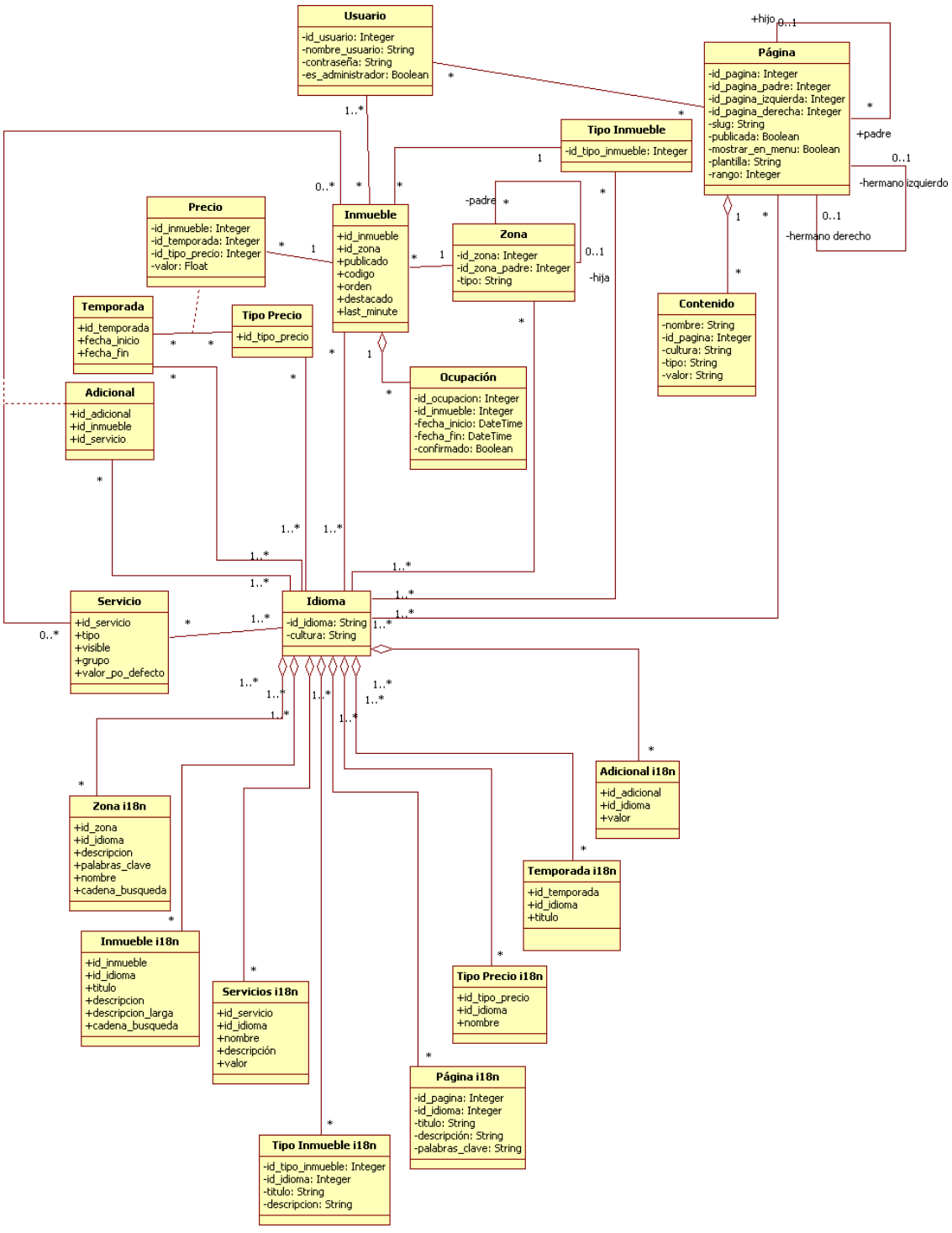


Ilustración 27: Diagrama de clases

El diagrama anterior muestra las entidades que intervienen en el sistema implementado para este proyecto. Puede observarse la importancia que acarrea el uso de múltiples idiomas para el modelado del sistema. Toda entidad que necesita que ciertos atributos estén adaptados a los diferentes idiomas definidos en el sistema, tienen su entidad complementaria "i18n" (acrónimo del término en inglés "internationalization").

Aunque podría haberse decidido representar las entidades relacionadas con el idioma como clases de asociación (en las asociaciones entre entidades principales y el idioma), en pro de una mejor visibilidad y compresión de este diagrama, dichas entidades se han representado como agregados del idioma. Si bien el sistema es el mismo a nivel conceptual, mediante esta representación podemos aislar las entidades complementarias de las entidades que conforman el núcleo funcional de la aplicación.

Capítulo IV. Diseño

1. Introducción

El diseño de aplicaciones orientadas a objetos es difícil, y todavía más difícil es diseñarlas de forma que éstas sean reutilizables. El diseño debe ser específico al problema que se maneja, pero al mismo tiempo debe ser lo bastante general como para soportar futuros problemas o requisitos, evitando en lo posible el tener que rediseñar.

En la creación de software, es bien sabido que en lugar de afrontar cada problema desde sus principios, es preferible reutilizar soluciones que han funcionado anteriormente. Cuando se encuentra una buena solución, ésta se utiliza una y otra vez.

Los patrones de diseño describen problemas que ocurren continuamente en la programación orientada a objetos, y detallan la base de la solución a dichos problemas de forma que puedan ser utilizados multitud de ocasiones. Los patrones de diseño aportan a los diseños orientados a objetos más flexibilidad, elegancia y reusabilidad.

En este capítulo se describe el patrón *MVC* (*Modelo-Vista-Controlador*) como solución de diseño de este proyecto. El patrón de diseño MVC es utilizado frecuentemente en aplicaciones web, por lo que proporciona una base bien establecida y de eficacia probada.

2. Patrón de diseño MVC

MVC viene de las siglas en inglés de *Model View Controller* o *Modelo Vista Controlador* en castellano. Es un método probado a lo largo del tiempo para separar la interfaz de usuario de una aplicación de su lógica de negocio (fue descrito por primera vez en 1979 e inicialmente usado para la creación de interfaces gráficas de usuario para Smalltalk).

El objetivo principal del patrón MVC es aislar cambios en la interfaz de usuario y prevenirlos de requerir cambios en la lógica de negocio de la aplicación.

La razón principal de esta división es que la interfaz de usuario y la lógica de dominio tienen diferentes motivos para el cambio y unas tasas de cambio diferentes. Al hacer esta separación, se puede cambiar la interfaz de usuario, sin tocar la lógica de negocio y viceversa.

MVC divide una aplicación en tres conceptos (véase Ilustración 28):

- **Modelo**: encapsula los datos básicos de la aplicación y la funcionalidad de la lógica de negocio.
- **Vista**: obtiene datos del modelo y la presenta al usuario.
- **Controlador**: recibe y traduce las acciones del usuario en peticiones al modelo o la vista.

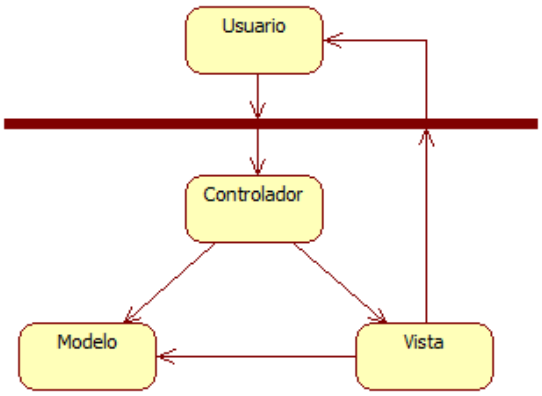


Ilustración 28: Estructura MVC

El siguiente diagrama representa de forma genérica el flujo de control de MVC.

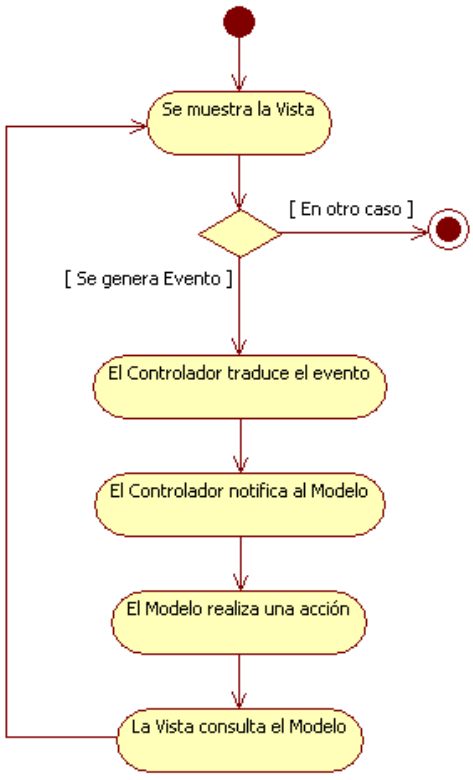


Ilustración 29: Flujo de control MVC

Como puede observarse en el diagrama anterior:

1. El usuario interactúa con la interfaz de usuario de alguna manera (por ejemplo, pulsando un botón del ratón).

2. El controlador gestiona el evento de entrada de la interfaz de usuario, y convierte el evento en una acción de usuario adecuada y comprensible para el modelo.

3. El controlador notifica al modelo de la acción del usuario, posiblemente resultando en un cambio de estado del modelo. (Por ejemplo, cambiando el código postal de una dirección).

4. Una vista consulta el modelo con el fin de generar una interfaz de usuario adecuada (por ejemplo una lista de los artículos del carrito de compra). La vista obtiene sus datos propios del modelo.

5. La interfaz de usuario espera a más interacciones del usuario, lo que se reinicia el ciclo de control de flujo.

MVC es visto frecuentemente en aplicaciones web donde la Vista es el código HTML o XHTML generado por la aplicación. El controlador recibe datos de entrada por peticiones GET y POST y decide qué hacer con ellos, los entrega a los objetos de la lógica de negocio (es decir, el modelo) que contienen las reglas de negocio y saben cómo llevar a cabo tareas específicas, y dejan el control en manos de componentes de generación de (X) HTML.

El modelo no es necesariamente sólo una base de datos, el "Modelo" en el MVC es a la vez los datos y la lógica de negocio necesarios para manipular los datos de la aplicación. Muchas aplicaciones utilizan un mecanismo de almacenamiento persistente como una base de datos para almacenar datos, pero MVC no menciona específicamente la capa de acceso a datos, ya que se entiende que está por debajo o encapsulado en el modelo. Los modelos no son objetos de acceso a datos, sin embargo, en aplicaciones muy simples que tienen poca lógica de negocio no puede hacerse una distinción real.

MVC describe una arquitectura y cada una de las partes del patrón MVC puede describirse mediante otros patrones de diseño más específicos o concretos, pero el estudio de estos patrones queda fuera del alcance de este proyecto. Para conocer más detalles de este patrón y de otros patrones complementarios puede consultarse (Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides "Design Patterns: Elements of Reusable Object-Oriented Software").

3. Modelo

El modelo encapsula el núcleo funcional de una aplicación, es decir, su lógica de negocio. El objetivo del MVC es hacer que el modelo sea independiente de la vista y el controlador, que en conjunto forman la interfaz de usuario de la aplicación. Por tanto, un objeto puede actuar como modelo para más de un trío MVC a la vez.

Dado que el modelo debe ser independiente, no puede referirse a las partes de la vista o al controlador de la aplicación. El modelo no puede mantener variables directas que refieran a la vista o el controlador. De manera pasiva presta sus servicios y datos a las otras capas de la aplicación.

El modelo pasivo es utilizado comúnmente en MVC para web. Con un modelo pasivo, los objetos utilizados en el modelo desconocen por completo que están siendo utilizados en la forma de MVC. Es decir, el controlador notifica a la vista cuando se ejecuta una operación en el modelo que requerirá la actualización de la vista. La vista siempre se reconstruye por completo en cada ciclo, independientemente de los cambios.

Como se ha comentado anteriormente, el modelo es a la vez los datos y la lógica de negocio y encapsula la capa de acceso a datos. Para este proyecto además de utilizar un sistema de gestión de base de datos relacional para la persistencia de datos, se añade una capa de abstracción para el mapeo de dichos datos con los objetos de nuestro modelo. Esto es lo que comúnmente se conoce como ORM (del inglés Object-Relational Mapping). Puede observarse esta relación en el siguiente diagrama.

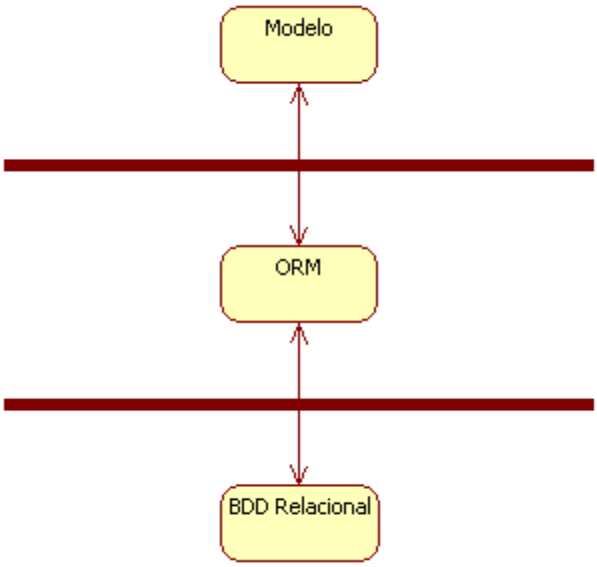


Ilustración 30: Componentes del Modelo

A continuación, como parte del modelo de la aplicación, se muestra la estructura de base de datos utilizada. En general, la mayoría de las tablas representan entidades detectadas anteriormente en la etapa de análisis, aunque algunas otras son utilizadas como configuración o se han añadido en previsión de nuevas funcionalidades de la aplicación.

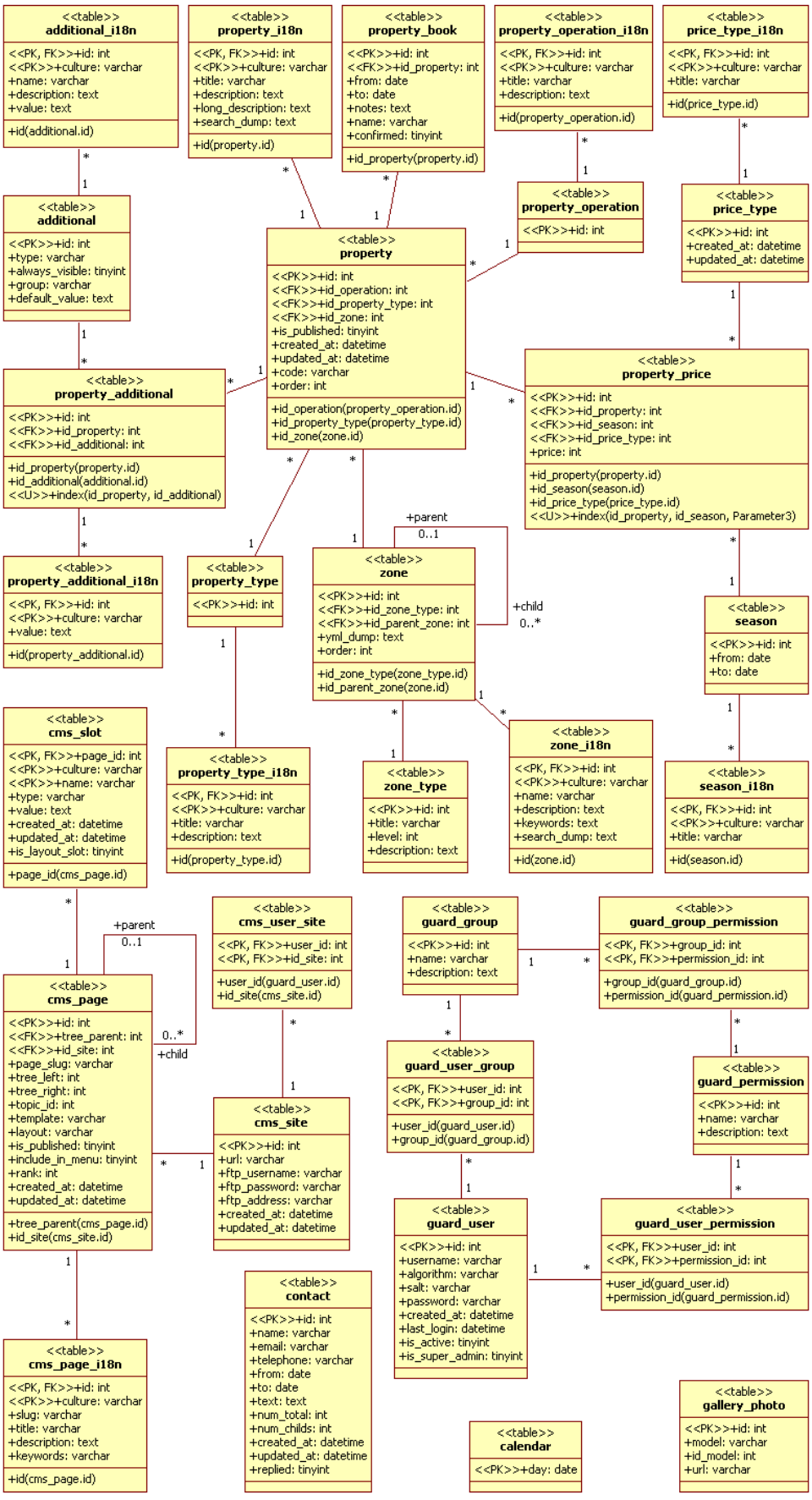


Ilustración 31: Estructura de base de datos

Se ha utilizado el diagrama de clases de UML para representar el esquema de tablas de la base de datos. Mediante el uso de estereotipos se identifican las claves primarias, las claves ajenas e índices únicos ("PK", "FK" y "U" respectivamente). También mediante operaciones de clase identificamos los campos relacionados por claves ajenas y la composición de índices múltiples.

4. Vista

La vista obtiene los datos del modelo y la presenta al usuario. La vista representa la salida de la aplicación.

La vista generalmente tiene libre acceso al modelo, pero no debe cambiar el estado del modelo. Las vistas son representaciones de sólo lectura del estado del modelo. La vista lee los datos del modelo utilizando métodos de consulta provistos por el modelo.

Para este proyecto se construirá la vista del patrón MVC a partir de un sistema de plantillas. Esta estrategia de implementación de la vista es conocida como *Template View*.

Un "template" o plantilla es un documento típicamente HTML con marcadores incrustados en su código que son remplazados, manipulados o evaluados mediante el API de un motor de plantillas para producir un documento de salida. El propósito de una plantilla es aislar el HTML del código del lenguaje de programación utilizado para implementar la lógica de la aplicación, en este caso PHP.

El controlador, tras notificar al modelo, comunicará a la vista que debe recargarse para mostrar los cambios en el modelo. La vista proporcionará la plantilla adecuada para mostrar el modelo, sustituirá los marcadores por los datos apropiados del modelo, y devolverá al usuario la salida final. Véase la siguiente figura.

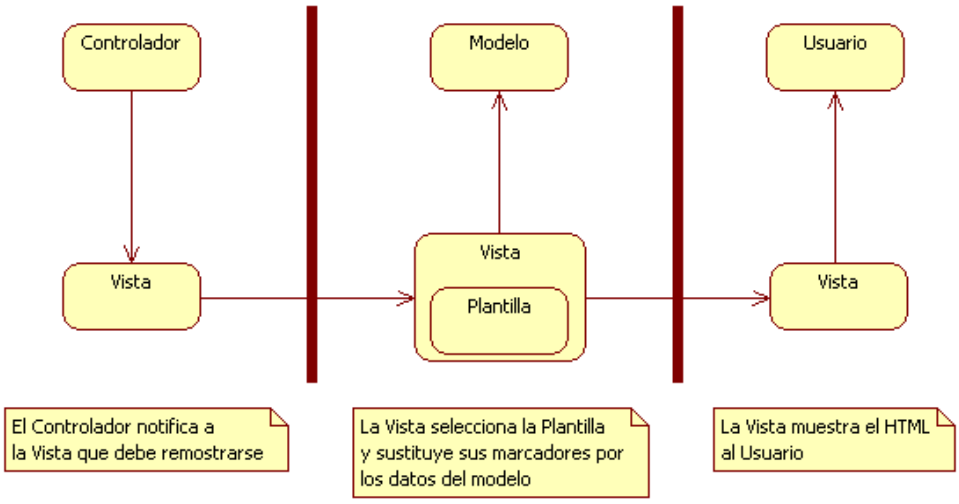


Ilustración 32: Comportamiento de la Vista

Las plantillas para este proyecto se construirán en forma de archivos PHP para aprovechar construcciones propias del lenguaje como iteradores o estructuras condicionales (de esta forma se pueden construir plantillas complejas como listados que necesitan cierta evaluación que el puro HTML no proporciona). Los marcadores de estas plantillas serán también código PHP, ya sean simples variables o complejas sentencias. De esta forma, las plantillas estarán formadas sólo por código HTML y PHP, y podrán ser evaluadas sin problemas por el intérprete PHP del servidor.

5. Controlador

El controlador recibe y traduce la entrada en solicitudes al modelo o la vista. Los controladores suelen ser responsables de llamar a los métodos en el modelo que cambian su estado. En el modelo pasivo típico de la web, el controlador se encarga de decirle a la vista cuando debe actualizarse.

La palabra "controlador" se ha sobrecargado con diferentes significados en distintos patrones. Existen varios de estos patrones para implementar el controlador, pero para este proyecto se utilizará el patrón conocido como *Controlador Frontal* (o *Front Controller* en inglés).

El controlador frontal se caracteriza por presentar un solo punto de entrada en el servidor web para interactuar con una aplicación web, es decir un solo punto de envío de peticiones HTTP.

Debido a la gran cantidad de peticiones que abordan los controladores frontales, éstos pueden ser propensos a problemas de rendimiento. Pero por contra, el controlador frontal aporta un control extraordinario de las peticiones realizadas a la aplicación, ayuda a eliminar duplicidades en el código y facilita la configuración de la aplicación en el servidor aumentando la portabilidad de la misma.

Un controlador frontal, por lo general, consiste en un administrador de peticiones (o sistema de enrutamiento) y un diccionario de comandos. El administrador de peticiones recibe las solicitudes del servidor web y envía la petición a un comando en su diccionario que se encargará de ella. Ver Ilustración 33.

Las aplicaciones web que utilizan este patrón pueden tener más de un controlador frontal. En concreto para este proyecto se van a utilizar dos: uno para la parte pública de la aplicación y otro para resolver las peticiones de la zona de administración. La siguiente figura pretende aclarar el funcionamiento de estos controladores frontales.

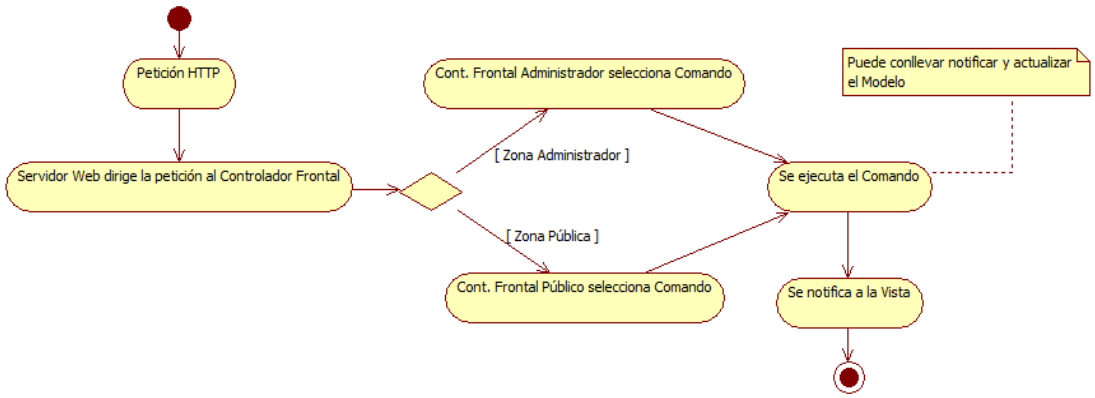


Ilustración 33: Comportamiento del Controlador Frontal

Capítulo V. Implementación

1. Introducción

En capítulos anteriores se han identificado los requisitos de la aplicación, se han localizado las entidades lógicas que la modelizan, se han definido los procesos involucrados, y se ha diseñado una arquitectura basada en el patrón MVC.

Este capítulo se basa en describir las tecnologías y recursos utilizados para construir la aplicación. En especial se centra en comentar las características del framework Symfony como solución de implementación.

No es fácil describir qué es un framework, a simple vista y por su traducción al castellano, no es más que un marco de trabajo. Pero aunque esta definición es correcta, se queda muy corta en lo que a framework se entiende en el desarrollo de software.

Un framework es un conjunto de librerías, herramientas y convenciones utilizado como base para la creación de software, y pueden definir esqueletos para aplicaciones completas o simplemente para tareas concretas. En general, un framework conforma un estándar que facilita la colaboración entre desarrolladores y ayuda al desarrollo rápido de aplicaciones.

Symfony es un framework para el desarrollo de aplicaciones web basado en el patrón MVC para PHP 5. Aunque a fecha de la escritura de esta memoria existen versiones estables de su versión 1.4 y están apareciendo las primeras versiones beta de Symfony 2, se ha obtado por una versión más estable, en concreto Symfony 1.2. Las razones para la utilización de esta versión ha sido la predilección de sus opciones de configuración por defecto (en especial adaptadas al ORM Propel).

Al igual que en el capítulo anterior, este capítulo se divide en los tres niveles del patrón MVC (modelo, vista y controlador) para comentar los destalles de implementación capa por capa, y los recursos que Symfony utiliza para cada nivel.

En el siguiente apartado no se pretende explicar en detalle todos los aspectos del framework Symfony. Para este fin, online puede consultarse (http://www.symfony-project.org: Página Oficial de Symfony Framework), la cual alberga una gran cantidad de información, documentación y tutoriales en varios idiomas. Por contra, servirá para dar a conocer dicho framework y cómo Symfony se estructura para dar cabida al patrón MVC. El siguiente apartado servirá de punto de partida para no perderse en los detalles de implementación de los apartados posteriores.

2. Symfony Framework

Symfony es un framework completamente diseñado para optimizar el desarrollo de aplicaciones web. Contiene numerosas herramientas y clases dirigidas a reducir el tiempo de desarrollo de aplicaciones web complejas. Adicionalmente, automatiza tareas comunes para que el desarrollador pueda enfocarse por completo en las características específicas de la aplicación.

Symfony está escrito enteramente en PHP 5. Ha sido probado exhaustivamente en varios proyectos reales y está siendo usado en sitios web de comercio electrónico con una demanda muy elevada. Symfony es compatible con la mayoría de motores de bases de datos disponibles, incluyendo MySQL, PostgreSQL, Oracle, y Microsoft SQL Server. Además funciona bajo plataformas tanto tipo Unix como Windows.

La primera versión de Symfony fue lanzada en Octubre de 2005 por su creador Fabien Potencier, CEO de Sensio, una agencia web francesa reconocida por su visión innovadora en el desarrollo web. Con el nacimiento de PHP 5, Fabien decidió que las herramientas disponibles habían alcanzado la madurez suficiente para ser integradas en un framework completo, y pasó un año implementando su núcleo. Symfony fue construido originalmente para ser usado en proyectos de Sensio, pero tras algunos casos de éxito, Fabien decidió lanzarlo bajo una licencia de código abierto y donarlo a la comunidad de desarrolladores.

Organización de código

Symfony organiza el código en una estructura de proyecto y coloca los ficheros del proyecto en una estructura de árbol estándar. En Symfony, un proyecto es un conjunto de servicios y operaciones disponibles bajo un nombre de dominio dado, que comparten el mismo modelo de objetos.

Dentro de un proyecto, las operaciones se agrupan lógicamente en aplicaciones. Una aplicación pude correr normalmente de forma independiente a otra aplicación del mismo proyecto. En muchos casos, un proyecto contendrá dos aplicaciones: una para el "front-office" y otra para el "back-office", compartiendo la misma base de datos (como en este proyecto que existe una aplicación para el portal público, y otra para la zona de administración). Pero pueden existir proyectos que contengan mini-sitios, cada uno como una aplicación diferente.

Cada aplicación es un conjunto de uno o más módulos. Un módulo normalmente representa una página o grupo de páginas con un propósito similar.

Los módulos albergan acciones, que representan el conjunto de acciones que pueden solicitarse en un módulo. Generalmente, las acciones pueden ser descritas con un verbo. Tratar con acciones es casi como tratar con páginas en una aplicación web clásica, aunque dos acciones pueden resultar en la misma página.

Symfony proporciona una estructura de directorios estándar para organizar todo este contenido de una manera lógica, consistente con las características de la arquitectura (MVC y el agrupamiento en proyecto/aplicación/modulo). La siguiente tabla muestra la estructura que es creada automáticamente cuando se inicializa cada proyecto, aplicación, o módulo. Por supuesto, Symfony da la posibilidad de

personalizarla por completo, para reorganizar los archivos y directorios a la conveniencia del desarrollador para adecuarla a los requisitos del cliente.

Árbol Raíz	Árbol de Aplicación	Árbol de Módulo

apps/
  [application name]/
cache/
config/
data/
  sql/
doc/
lib/
  model/
log/
plugins/
test/
  bootstrap/
  unit/
  functional/
web/
  css/
  images/
  js/
  uploads/
	[application name]/
	  config/
	  i18n/
	  lib/
	  modules/
	  [module name]
	  templates/
	  layout.php
		[module name]/
		  actions/
		  actions.class.php
		  config/
		  lib/
		  templates/
		  indexSuccess.php

Configuración en cascada

La configuración de los proyectos Symfony se distribuye en varios archivos. Estos archivos contienen definiciones de parámetros o ajustes. Algunos de estos parámetros pueden ser sobreescritos a varios niveles (proyecto, aplicación y módulo), algunos son específicos de un cierto nivel.

El mismo ajuste puede definirse más de una vez, en sitios diferentes. De hecho, hay varios niveles de configuración en Symfony:

- Niveles de granularidad:
  - La configuración por defecto propia del framework.
  - La configuración global para todo el proyecto (en "miproyecto/config").
  - La configuración local para una aplicación del proyecto (en "miproyecto/apps/ miaplicación/config").
  - La configuración local restringida a un módulo (en "miproyecto/apps/ miaplicación/modules/mimodulo/config").
- Niveles de entorno:
  - Específica a un entorno.
  - Para todos los entornos.

De todas las propiedades que pueden configurarse, muchas son dependientes del entorno. Por tanto, muchos archivos de configuración se dividen por entorno, más una sección especial para todos los entornos por defecto.

Como se ha dicho, esto significa que una propiedad puede definirse varias veces, y el valor real resulta de la definición en cascada. Es decir, una definición de un parámetro en un entorno dado tiene precedencia sobre la misma definición de parámetro para todos los entornos, el cual tiene precedencia sobre la definición en la configuración por defecto. Un parámetro definido a nivel de módulo tiene precedencia sobre el mismo parámetro definido a nivel de aplicación, el cual tiene precedencia sobre el parámetro definido a nivel de proyecto. En consecuencia, a continuación se expone la lista de prioridades de configuración que se aplican en Symfony:

1. Módulo.
2. Aplicación.
3. Proyecto.
4. Específico al entorno.
5. Todos los entornos.
6. Por defecto del framework.

Autoloading

Cuando se usa un método de clase o se crea un objeto en PHP, es necesario incluir la definición de la clase primero. En proyectos grandes con muchas clases y una estructura de directorios profunda, es difícil hacer el seguimiento de todas estas clases para incluirlas correctamente.

Symfony hace innecesario el uso de directivas "include" y puede instanciarse directamente cualquier clase definida. Symfony busca las definiciones de clase necesarias consultando los archivos con extensión ".php" en todos los directorios "lib" del proyecto. Si la definición de la clase es encontrada, ésta se incluye automáticamente. Esta es la razón por la que los proyectos Symfony no contienen ninguna directiva "include" o "require" en su código de aplicación.

3. Modelo

Con anterioridad se ha comentado que el Modelo está compuesto por los datos de la aplicación y por la lógica de negocio. Al mismo tiempo, la lógica de negocio de una aplicación web recae en gran medida en el modelo de datos.

Propel ORM

Para la persistencia de datos, Symfony es capaz de trabajar con los sistemas de gestión de base de datos mayoritarios, y en concreto para este proyecto se ha utilizado MySQL con el motor InnoDB capaz de gestionar la integridad referencial entre tablas. Para la lógica de negocio, Symfony se basa en una capa de mapeo objeto-relacional u *ORM* conocido con el nombre de *Propel* (http://www.propelorm.org: Página Oficial de Propel ORM). Este componente es el utilizado por defecto en la versión 1.2 de Symfony, pero también es capaz de trabajar con otro ORM llamado Doctrine (http://www.doctrine-project.org: Página Oficial de Doctrine ORM), que es el ORM por defecto a partir de la versión 1.4. Propel también hace uso de una capa de abstracción para la base de datos conocida por las siglas *PDO* o *PHP Data Objects*, por lo que siguiendo el esquema de la Ilustración 30 del capítulo de diseño, la relación existente entre todos los elementos del Modelo de la aplicación Symfony de este proyecto puede quedar representada mediante la siguiente figura.

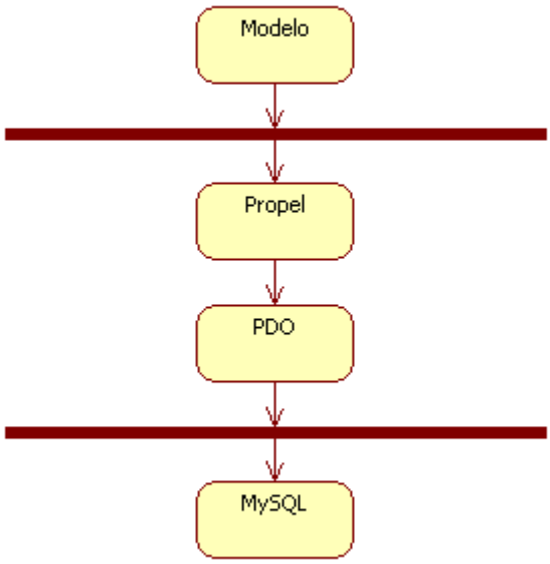


Ilustración 34: Estructura del Modelo en Symfony

La explicación del porqué de esta división en capas del Modelo es muy clara, la base de datos (MySQL) es relacional mientras que PHP 5 (http://www.php.net: Página Oficial de PHP) y Symfony son orientados a objetos. Para acceder a la base de datos de la forma más efectiva en un contexto orientado a objetos, se necesita una interfaz para traducir la lógica de objetos a la lógica relacional y viceversa. Además, la utilización de este tipo de tecnología aporta una serie de ventajas muy favorables:

- Reusabilidad, permitiendo a varias partes de la aplicación acceder a métodos del modelo, incluso por aplicaciones distintas.

- La encapsulación de la lógica de negocio, evitando la modificación de todo el modelo ante un cambio de las reglas de negocio.

- Permite añadir métodos de acceso a los objetos que no tienen por qué corresponder con columnas específicas de la base de datos.

- Añade portabilidad en cuanto al sistema de gestión de base de datos, al utilizar una sintaxis propia e independiente para la construcción de consultas SQL.

El esquema y la conexión con la base de datos

Para crear el modelo de objetos de datos que Symfony usará en la aplicación, es necesario traducir el modelo relacional de la base de datos a un modelo de objetos. Propel necesita una descripción del modelo relacional para realizar el mapeo, lo que se conoce como "esquema". En el esquema se definen las tablas del modelo relacional, sus relaciones, y las características de sus columnas.

Symfony almacena el esquema del modelo en un archivo llamado "schema.yml" (en formato *YAML*) en el directorio "config" en el nivel de aplicación. Symfony permite al desarrollador construir el esquema, y a través de la conexión con la base de datos, generar el modelo relacional en el sistema de base de datos. Pero también permite realizar la operación opuesta, es decir, hacer introspección de una base de datos y generar el esquema del modelo.

El modelo de datos en Symfony es independiente del sistema de gestión de base de datos utilizado, pero al final es necesario utilizar uno, en este caso MySQL. La información mínima que Symfony requiere para el envío de peticiones a la base de datos es el nombre, las credenciales (usuario y contraseña) y el tipo de base de datos. Esta información se almacena en el archivo "databases.yml", que se encuentra en la carpeta "config" del directorio raíz(al igual que el "eschema.yml").

Se puede consultar más sobre el formato *YAML*, la configuración de la conexión de la base de datos y la forma de crear el esquema en la documentación disponible en la página (http://www.symfony-project.org: Página Oficial de Symfony Framework).

Las clases del modelo

Symfony autogenera las clases necesarias para el modelo (y por tanto para la capa del ORM) a partir del esquema. En concreto para cada tabla del esquema se generarán 4 clases diferentes en directorios diferentes:

- En el directorio "lib/model/om" se generan las clases "base" del modelo: BaseNombreTabla.php y BaseNombreTablaPeer.php.

- En el directorio "lib/model" se generan las clases del modelo: NombreTabla.php y NombreTablaPeer.php.

Estas clases se clasifican en directorios diferentes porque las clases "base" se regeneran cada vez que realizamos un cambio en el esquema, mientras que las otras permanecen inalteradas. Por tanto, si queremos ampliar o extender el modelo añadiendo métodos propios a las clases, siempre se tendrán que hacer los cambios pertinentes en las clases del modelo, nunca en las clases base ya que perderíamos las modificaciones en caso de cambiar el esquema. Las clases del modelo son las utilizadas por Symfony para acceder a los datos, y éstas heredan directamente de sus clases base.

También se hace distinción entre clases de objeto y clases "peer". Las clases objeto son las que representan un registro de la base de datos. Estas clases dan acceso a las columnas de un registro y a sus registros relacionados. Las clases "peer" son clases que contienen métodos estáticos para operar sobre las tablas. Estas clases proveen de los mecanismos para obtener registros de las tablas. Sus métodos normalmente retornan un objeto o una colección de objetos del tipo de la clase específica.

4. Vista

La vista es responsable de mostrar la salida correspondiente a una acción particular. En Symfony, la vista consiste en varias partes, con cada parte diseñada para ser fácilmente modificada por la persona que normalmente trabaja con ella.

- Los diseñadores web generalmente trabajan sobre plantillas (la presentación de los datos de la acción actual) y sobre el *layout* (que contiene el código común a todas las páginas). Estas partes están escritas en HTML con pequeños trozos de código PHP embebidos.

- Por reusabilidad, los desarrolladores usualmente empaquetan fragmentos de plantillas en *parciales* y *componentes*. También usan *slots* para modificar zonas del *layout*. Los diseñadores también pueden trabajar en esta área.

- Los desarrolladores se centran en el archivo YAML de configuración de la vista (definiendo propiedades de la respuesta y de otros elementos de la interfaz) y el objeto de la respuesta.

Layout y plantillas

El fichero que contiene el layout de una página se llama "layout.php", y se almacena en la carpeta "templates" bajo el directorio de cada aplicación. Este archivo, también llamado *plantilla global*, almacena el código HTML que es común a todas las páginas de la aplicación para evitar su repetición en cada plantilla. El contenido de una plantilla es integrado dentro del layout, o desde otro punto de vista, se puede decir que el layout "decora" la plantilla. Esto es una aplicación del patrón de diseño *Decorador* como se muestra en la siguiente figura.

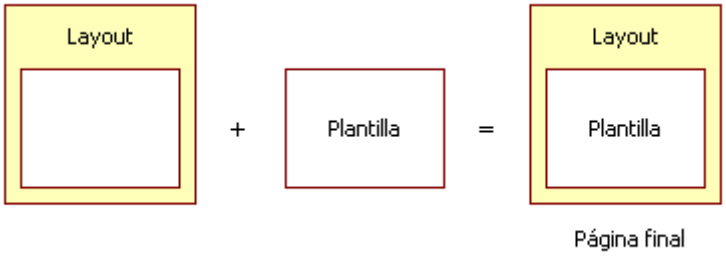


Ilustración 35: Esquema del patrón Decorador

En realidad, este es el layout utilizado por defecto por Symfony. Pero Symfony permite el uso de cuantos ficheros de layout definidos por el desarrollador se desee, ya que podrá configurarse qué layout debe utilizarse por cada acción (incluso la no utilización de un layout).

Las plantillas de cada acción se almacenan en el directorio "templates" al nivel de cada módulo de la aplicación. Por convención, Symfony necesita que los nombres de los archivos que contienen el código de las plantillas sean de la forma "nombreAcciónNombrePlantilla.php", donde "NombrePlantilla" suele ser por defecto las palabras "Success" o "Error" (para cuando una acción ha sido exitosa o fallida respectivamente), aunque también puede ser cualquier nombre definido por el desarrollador.

Fragmentos

En ocasiones es necesario incluir algo de código común HTML o PHP en varias páginas. Para evitar la repetición del código, la directiva "include()" de PHP puede ser suficiente la mayoría de los casos. Pero Symfony pone a disposición tres alternativas de inclusión de código más "inteligentes":

- Si la lógica es ligera, y simplemente se requiere incluir un trozo de plantilla al que se le pasa algún dato. En este caso se hace uso de *parciales*.

- Si la lógica es más pesada (por ejemplo, se necesita tener acceso al modelo y/o modificar el contenido en función de la sesión), donde es preferible separar la presentación de la lógica. En este caso se hace uso de *componentes*.

- Si el fragmento tiene como objetivo remplazar una parte específica del layout. En este caso se usa un *slot*.

Un parcial es un trozo de código de plantilla reutilizable. Al igual que las plantillas, los parciales son archivos localizados en el directorio "templates", y contienen código HTML con código PHP embebido. El nombre de un parcial siempre empieza con el carácter de guión bajo (_), lo que ayuda a distinguirlos de las plantillas, ya que los archivos están localizados en los mismos directorios "templates".

Una plantilla puede incluir parciales de su mismo módulo, parciales de otro módulo, o parciales del directorio "templates" global. Los parciales se incluyen en el código utilizando la función "include_partial()" de Symfony.

Del mismo modo que el patrón MVC se aplica a acciones y plantillas, se puede necesitar separar un parcial en una parte de lógica y otra de presentación. Un componente es como una acción, pero mucho más rápido. La lógca de un componente se mantiene en una clase que hereda de la clase de Symfony "sfComponents", localizada en el fichero "actions/components.class.php". Su presentación se mantiene en un archivo parcial. Los métodos de la clase "sfComponents" empiezan con la palabra "execute", como las acciones, y pueden pasar variables a la presentación de forma análoga a como las acciones pasan variables a las plantillas.

Los parciales que sirven de presentación para componentes son nombrados por el componente (sin el "execute" inicial, pero con un subguión en su lugar). Pero los componentes no hacen comprobaciones de seguridad o validaciones, no pueden ser llamados desde Internet (sólo por la propia aplicación), y no tienen la posibilidad de retornar varias posibilidades de presentación. Estas son las razones por las que un componente es más rápido de ejecutar que una acción.

Los componentes se incluyen en el código de una plantilla utilizando la función "include_component()" de Symfony. En la siguiente tabla se observan las similitudes entre acciones y componentes.

Convención	Actions	Components
Archivo de lógica	`actions.class.php`	`components.class.php`
Clase padre	`sfActions`	`sfComponents`
Nombre de los métodos	`executeMyAction()`	`executeMyComponent()`
Nombre archivo de presentación	`myActionSuccess.php`	`_myComponent.php`

Parciales y componentes son muy útiles para aumentar la reusabilidad de la aplicación. Pero en muchos casos, es necesario rellenar el "layout" de una aplicación que contiene varias zonas dinámicas. Básicamente, un "slot" es un contenedor que puede ponerse en cualquiera de los elementos de la vista (el layout, una plantilla o un parcial). Rellenar este contenedor es como asignarle un valor a una variable. El código de relleno se almacena de forma global en la respuesta, por lo que puede definirse en cualquier lugar (en el layout, la plantilla o parcial). Sólo hay que asegurarse de definirlo antes de incluirlo, y recordar que el layout se ejecuta después de la plantilla (por el proceso de decoración), y los parciales son ejecutados cuando son llamados en la plantillas.

Para incluir un "slot", se usa la función de Symfony "include_slot()". La función "has_slot()" retorna "true" si el slot ha sido definido previamente, lo que proporciona un mecanismo de recuperación ante fallos. Los slots son muy útiles para definir zonas destinadas a mostrar contenido dependiente del contexto. También se usan para añadir HTML al layout para ciertas acciones.

Archivo de configuración de la Vista

En Symfony, una vista consiste en dos partes diferenciadas:

- El HTML de la presentación del resultado de una acción, almacenado en la plantilla, el layout y en fragmentos de plantillas.

- Y todo lo demás, incluyendo lo siguiente:

  - Declaraciones de meta-contenido: palabras clave, descripción de página, o duración de caché, etc.

  - El título de la página: que no sólo ayuda a los usuarios a localizar una página entre todas las ventanas del navegador abiertas, sino que es muy importante para la indexación y el posicionamiento de una página en los buscadores.

  - Inclusión de archivos: como JavaScript y hojas de estilo en cascada.

  - Selección del layout específico: algunas acciones requieren un layout personalizado (pop-ups, o anuncios publicitarios, etc) o la no utilización de layout (como las acciones Ajax).

En la vista, todo lo que no es HTML se llama configuración de la vista, y Symfony a pone a disposición del desarrollador dos formas de manipularla. La forma usual es a través del archivo de configuración "view.yml" localizado en los directorios "config". Este archivo puede usarse para configurar la vista cuando los valores no dependen del contexto o en consultas a la base de datos. Cuando es necesario asignar valores de forma dinámica, el método alternativo es utilizar directamente los métodos que proporciona la clase "sfResponse" desde la acción.

Cada módulo de la aplicación puede tener su propio archivo "view.yml" definiendo las características de sus vistas. Esto permite configurar la vista para todo un módulo o por cada acción desde un sólo archivo. La configuración por defecto para todas las vistas de la aplicación se definen en el archivo "view.yml" del directorio "config" de la aplicación. La configuración final de la vista para una acción vendrá marcada por el principio de configuración en cascada explicada anteriormente, y por el que una configuración a nivel específico sobreescribe las configuraciones del nivel superior más genérico.

5. Controlador

En Symfony, la capa del controlador, la cual contiene el código que enlaza la lógica de negocio y la presentación de la información, está separada en varios componentes que se usan para diferentes propósitos:

- El controlador frontal es el único punto de entrada a la aplicación, carga la configuración y determina que acción debe ejecutarse.

- Las acciones contienen la lógica de la aplicación, comprueban la integridad de las peticiones y preparan los datos necesarios para la capa de presentación.

- Los objetos que representan la petición, la respuesta y la sesión dan acceso a los parámetros de la petición, las cabeceras de respuesta y los datos persistentes sobre el usuario. Se usan con mucha frecuencia en la capa del controlador.

- Los filtros son porciones de código que se ejecutan para cada petición, antes o después de cada acción. Un buen ejemplo podría ser el filtro de seguridad que se encarga de comprobar los permisos del usuario para ejecutar una acción.

El controlador frontal

El controlador frontal en Symfony coincide en gran medida con el patrón de controlador frontal definido en el capítulo de diseño. Todas las peticiones web son manejadas por un sólo controlador frontal, el cual es el único punto de entrada para toda la aplicación. Aunque cabe recordar que en Symfony, un proyecto se compone de una o varias aplicaciones, por lo que podemos tener varios controladores frontales.

En particular, este proyecto se compone de 2 aplicaciones, la correspondiente a la zona pública del portal, y la zona de administración. Para cada aplicación existe un controlador frontal llamados "index.php" y "admin.php" respectivamente, que se localizan bajo el directorio "web" de Symfony (todos los controlodadores frontales se sitúan bajo este directorio en Symfony, y pueden autogenerarse mediante tareas específicas del framework).

Cuando el controlador frontal recibe una petición, éste usa el sistema de routing (su administrador de peticiones) para localizar el nombre de la acción y el nombre del módulo (su diccionario de comandos) que se corresponde con la URL determinada por el usuario. Pero en esencia, el controlador frontal en Symfony realiza también otro tipo de tareas que podemos ver en el siguiente diagrama:

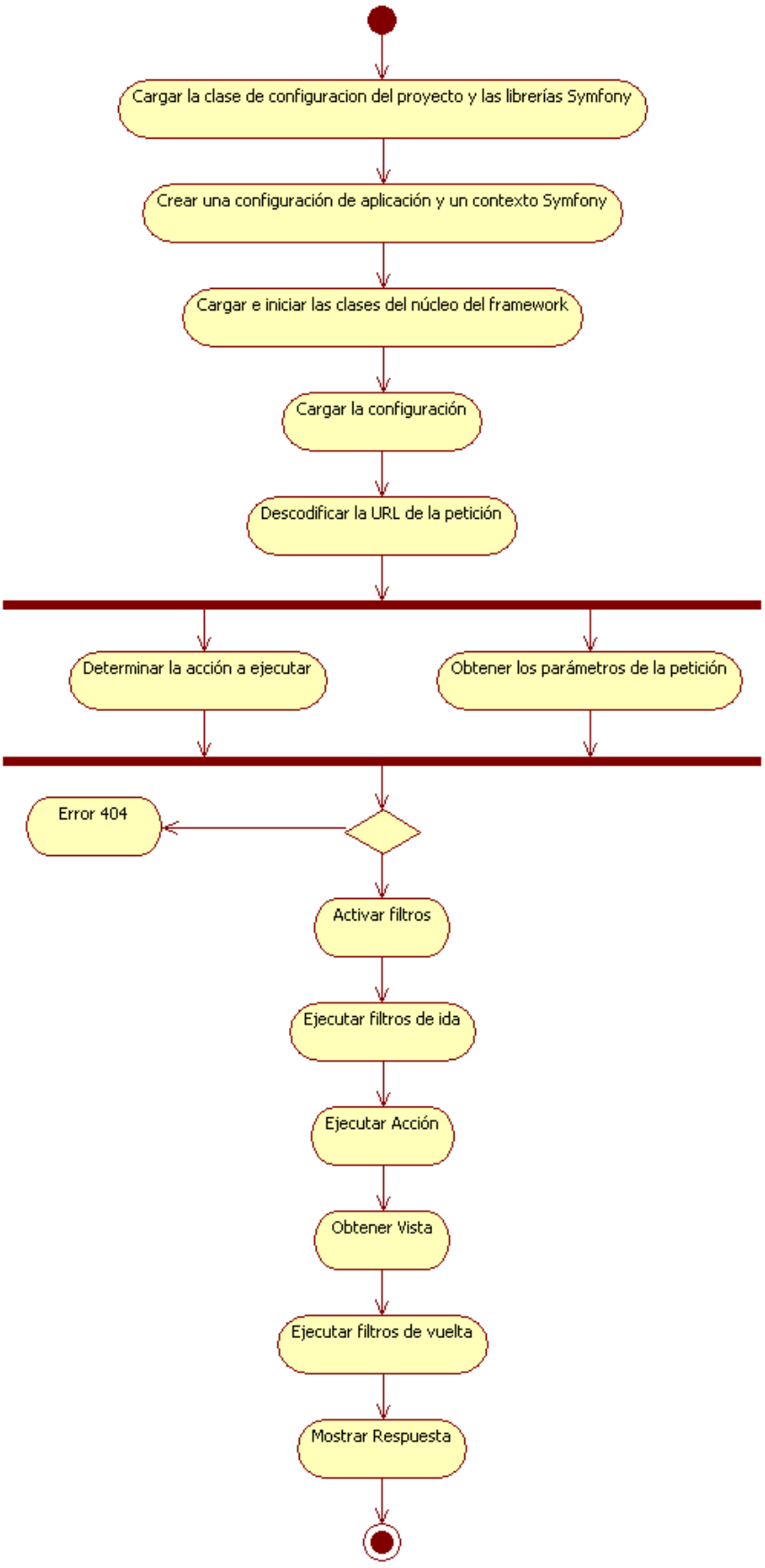


Ilustración 36: Flujo de trabajo del Controlador Frontal

El sistema de enrutamiento

El enrutamiento es un mecanismo que reescribe las URLs para hacerlas más amigables para el usuario. Las URLs pueden verse como instrucciones para el servidor, éstas transportan información requerida del navegador al servidor para provocar una acción conforme desea el usuario. Por tanto, el enrutamiento se sustenta en considerar la URL como parte de la interfaz de usuario. La aplicación puede formatear una URL para brindar información al usuario, y el usuario puede usar la URL para acceder a los recursos de la aplicación.

Symfony desconecta la URL externa de su URI interna. La correspondencia entre la dos se hace por el *sistema de routing*. El sistema de routing usa un archivo de configuración especial, llamado "routing.yml" y almacenado en la carpeta "config" de cada directorio de aplicación, en el que se pueden definir reglas de enrutamiento. Estas reglas siguen unas normas de *pattern matching* y se contrastan con la URL de cada petición de forma secuencial (del principio al fin del fichero "routing.yml"). En el momento que la URL cumple con una de las reglas definidas, el sistema de routing deja de comprobar las reglas posteriores, y redirige el control del framework hacia la acción del módulo indicada por dicha regla.

El sistema de routing de Symfony es un componente muy potente, aunque puede llegar a ser algo complejo de entender. Para conocer en profundidad el funcionamiento de esta parte del framework puede consultarse (http://www.symfony-project.org: Página Oficial de Symfony Framework).

Las acciones

Las acciones son el corazón de la aplicación, porque contienen toda su lógica. Las acciones usan el modelo y definen variables para la vista. Cuando se realiza una petición web en una aplicación Symfony, como se ha comentado anteriormente, la URL define una acción y ciertos parámetros adjuntos.

Las acciones son métodos llamados "executeNombreAccion" de la clase "nombreModuloActions" que hereda de la clase del núcleo de Symfony "sfActions", y agrupadas por módulos. La clase de acciones de un módulo se almacena en un fichero .php llamado "actions.class.php", en el directorio "actions" de cada módulo de la aplicación.

Las acciones pueden escapar o redirigirse a otras acciones de la aplicación, o por el contrario pueden retornar el valor de la vista que debe mostrarse al usuario. Por defecto, en caso de omitir el retorno de una vista concreta, el framework siempre tratará de mostrar la plantilla de la vista del tipo "success", es decir, tratará de buscar un template del módulo con el nombre "nombreAccionSuccess.php".

Capítulo VI. Evaluación y pruebas

1. Introducción

En este capítulo de la memoria se llega al final del ciclo de desarrollo de la aplicación. Aunque la fase de pruebas se realiza a la terminación del desarrollo (habitualmente en sucesivas iteraciones del ciclo), no por ello carece de la misma importancia, incluso mayor si cabe, que las fases anteriores. Previamente a que el proyecto pueda pasar a producción, el sistema debe someterse a las pruebas necesarias que acrediten el correcto funcionamiento del mismo.

Las pruebas realizadas al proyecto se clasifican en dos bloques, según la parte de la aplicación que se desea evaluar. Por un lado, las pruebas funcionales o de comportamiento, tratan de validar la funcionalidad de la zona privada de administración, en su mayoría. Por otro lado, las comprobaciones de visualización y estándares tecnológicos van dirigidas a evaluar la zona pública, ya que éstas carecen de importancia en la parte privada.

Las pruebas funcionales, también conocidas como *pruebas de caja negra*, tienen por objetivo probar que los sistemas desarrollados cumplan con las funciones específicas para los cuales han sido creados. Básicamente, el enfoque de este tipo de prueba es centrarse en el análisis de los datos de entrada y en los de salida para una funcionalidad específica. Estos datos generalmente se definen en los *casos de prueba* preparados antes del inicio de las pruebas, los cuales derivan fácilmente de la especificación de requisitos del Capítulo II.

Al realizar pruebas funcionales lo que se pretende es ponerse en la piel del usuario, usar el sistema como él lo usaría, pero llegando aún más lejos, probando el sistema en los límites de sus posibles vulnerabilidades.

La zona pública de la aplicación es accesible por usuarios de todo el mundo, cada uno utilizando dispositivos de diferente naturaleza. Dadas las diferencias existentes en las tecnologías de los navegadores web y la amplia gama de hardware del mercado, las pruebas de visualización nos permiten observar de qué forma se muestra en pantalla la aplicación para un subconjunto de configuraciones hardware-software.

2. Pruebas funcionales

Las pruebas de software se aplican como una etapa más del proceso de desarrollo de software, su objetivo es asegurar que el software cumpla con las especificaciones requeridas y eliminar los posibles defectos que este pudiera tener. En la actualidad las pruebas de software se han convertido en una de las etapas más críticas del ciclo de vida del desarrollo de software y esto ha causado el origen de diversas metodologías.

En la Ingeniería del software, los *casos de prueba* son un conjunto de condiciones o variables bajo las cuáles el analista determinará si el requisito de una aplicación es parcial o completamente satisfactorio. Pero para reforzar la idea, puede decirse que los casos de prueba ayudan a validar que la aplicación desarrollada realice las funciones para las que ha sido creada en base a los requerimientos del cliente.

A continuación se detallan algunos de los casos de prueba más característicos desarrollados para completar esta fase del ciclo de desarrollo. Mayoritariamente los siguientes casos de prueba tratan de validar el comportamiento de la parte privada o de administración, y hacen referencia a los requisitos funcionales definidos en el Capítulo II. Por tanto, cabe mencionar que muchas de las pruebas han sido realizadas directamente a partir de los requisitos, sin ser documentados en casos de prueba por su complejidad en el proceso de "testeo".

Para evitar ser demasiado redundantes, los casos de prueba se llaman igual y siguen el mismo orden de aparición en el presente documento que sus requisitos funcionales homólogos. Además, únicamente se detallan los valores específicos usados para la entrada, el resultado esperado, y si este resultado es satisfactorio para cada caso:

1. **Insertar zonas.**

   a) Insertar zona de nivel "ciudad" (primer nivel).

   - Objetivos: validar la correcta inserción en la base de datos de una zona de primer nivel en base a datos de entrada requeridos correctos.
   - Pre-requisitos: -
   - Entradas:
     - Tipo zona: ciudad.
     - Zona padre: -
     - Nombre: Nueva Ciudad.
     - Descripción: Descripción de nueva ciudad.
     - Palabras clave: Palabras clave de nueva ciudad.
     - Texto de búsqueda: nueva ciudad, nova cidad, nuva cuidad.

- Resultado esperado: la zona se guarda en la base de datos y se comunica el éxito de la operación con un mensaje informativo.
- Resultado: ok.

b) Insertar zona de nivel “semi-zona” (último nivel).

- Objetivos: validar la correcta inserción de una zona de último nivel en base a datos de entrada requeridos correctos.
- Pre-requisitos: existe una zona de nivel "zona" (segundo nivel) con el nombre “Nueva Zona”.
- Entradas:
  - Tipo zona: semi-zona.
  - Zona padre: Nueva Zona.
  - Nombre: Nueva Semi-Zona.
  - Descripción: Descripción de nueva semi-zona.
  - Palabras clave: Palabras clave de nueva semi-zona.
  - Texto de búsqueda: nueva semi-zona, nova sem-zona, nuva semi-zona.
- Resultado esperado: la zona se guarda en la base de datos y se comunica el éxito de la operación con un mensaje informativo.
- Resultado: ok.

c) Insertar zona sin nombre.

- Objetivos: validar la no inserción de una zona sin nombre.
- Pre-requisitos: -
- Entradas:
  - Tipo zona: zona.
  - Nombre: -
- Resultado esperado: se muestra un mensaje indicando la falta del dato del nombre de la zona y la zona no se almacena en la base de datos.
- Resultado: ok.

2. Modificar zonas.

a) Modificación total de zona.

- Objetivos: validar que es posible modificar cualquier dato de una zona, incluso el tipo de zona o su zona padre.

- Pre-requisitos: la zona a modificar no puede tener zonas “hijas”, y debe existir otra zona en el sistema que no sea de último nivel (semi-zona).
- Entradas:
  - Tipo zona: ciudad.
  - Zona padre: Calasetta.
  - Nombre: Otro Nombre.
  - Descripción: Descripción cambiada.
  - Palabras clave: Nuevas palabras clave.
  - Texto de búsqueda: oto nombre, otro nomre.
- Resultado esperado: se muestra un mensaje indicando que los datos de la zona han sido cambiados.
- Resultado: ok.

b) Vaciar el nombre de la zona.

- Objetivos: validar que no es posible dejar en blanco el nombre de una zona guardada.
- Pre-requisitos: la zona debe existir en la base de datos.
- Entradas:
  - Tipo zona: ciudad.
  - Nombre: -
- Resultado esperado: se muestra un mensaje indicando la falta del dato del nombre de la zona y la zona no se cambia en la base de datos.
- Resultado: ok.

3. Borrar zonas.

a) Borrar zona sin propiedades relacionadas.

- Objetivos: validar que es posible eliminar una zona guardada.
- Pre-requisitos: la zona debe existir en la base de datos y no está relacionada con ninguna propiedad del catálogo.
- Entradas: -
- Resultado esperado: se muestra un mensaje indicando que la zona ha sido eliminada y la zona y sus hijas se borran de la base de datos.
- Resultado: ok.

b) Borrar zona con propiedades asignadas.

- Objetivos: validar que no es posible eliminar una zona guardada asignada a una propiedad del catálogo.
- Pre-requisitos: la zona debe existir en la base de datos y debe estar relacionada con alguna propiedad del catálogo.
- Entradas: -
- Resultado esperado: se muestra un mensaje indicando que la zona no ha sido eliminada y ni la zona ni sus hijas se borran de la base de datos.
- Resultado: ok.

c) Borrar zonas en “batch”.

- Objetivo: validar que es posible realizar el borrado de zonas de forma masiva.
- Pre-requisitos: deben existir una o varias zonas para borrar.
- Entradas: conjunto de identificadores de zona para borrar.
- Resultado esperado: las zonas seleccionadas se borran de la base de datos. En caso que alguna zona no pudiera borrarse, se comunica mediante un mensaje de error, pero en todo caso las zonas válidas se borran igualmente.
- Resultado: ok.

4. Insertar Temporadas.

a) Insertar temporada vacía.

- Objetivos: validar que no es posible la inserción de una temporada sin indicar los datos requeridos.
- Pre-requisitos: -
- Entradas: -
- Resultado esperado: la temporada no se guarda en la base de datos y se comunica de los campos que son requeridos para la inserción.
- Resultado: ok.

b) Insertar temporada completa.

- Objetivos: validar la correcta inserción de una temporada en base a datos de entrada requeridos correctos.
- Pre-requisitos: -
- Entradas:

- Fecha inicial: 16/07/2012.
- Fecha final: 30/08/2012.
- Título: Temporada alta.

- Resultado esperado: la temporada se guarda en la base de datos y se comunica el éxito de la operación con un mensaje informativo.
- Resultado: ok.

c) Insertar temporada con fechas incorrectas.

- Objetivos: validar la no inserción de una temporada con la fecha de inicio posterior a la fecha de fin.
- Pre-requisitos: -
- Entradas:
  - Fecha inicial: 16/07/2012.
  - Fecha final: 15/07/2012.
  - Título: Nueva temporada.
- Resultado esperado: se muestra un mensaje indicando que las fechas no son válidas y la temporada no se almacena en la base de datos.
- Resultado: ok.

5. Modificar Temporadas.

a) Modificación total de temporada.

- Objetivos: validar que es posible modificar cualquier dato de una temporada.
- Pre-requisitos: la temporada a modificar existe en el sistema.
- Entradas:
  - Fecha inicial: 15/07/2012.
  - Fecha final: 16/07/2012.
  - Título: Temporada cambiada.
- Resultado esperado: se muestra un mensaje indicando que los datos de la temporada han sido cambiados y se modifican los datos en la base de datos.
- Resultado: ok.

b) Vaciar la temporada.

- Objetivos: validar que no es posible dejar en blanco cualquiera de los datos requeridos de una temporada.
- Pre-requisitos: la temporada debe existir en la base de datos.
- Entradas:
  - Fecha inicial: -
  - Fecha final: -
  - Título: -
- Resultado esperado: se muestra un mensaje indicando la falta de los datos requeridos y la temporada no se cambia en la base de datos.
- Resultado: ok.

6. Borrar Temporadas.

a) Borrar temporada sin precios relacionados.

- Objetivos: validar que es posible eliminar una temporada guardada.
- Pre-requisitos: la temporada existe en la base de datos y no está relacionada con ningún precio asignado a una propiedad del catálogo.
- Entradas: -
- Resultado esperado: se muestra un mensaje indicando que la temporada ha sido eliminada y la temporada se borra de la base de datos.
- Resultado: ok.

b) Borrar temporada con precios relacionados.

- Objetivos: validar que no es posible eliminar una temporada guardada utilizada para asignar un precio a una propiedad del catálogo.
- Pre-requisitos: la temporada debe existir en la base de datos y debe estar relacionada con algún precio de alguna propiedad del catálogo.
- Entradas: -
- Resultado esperado: se muestra un mensaje indicando que la temporada no ha sido eliminada y la temporada no se borra de la base de datos.
- Resultado: ok.

c) Borrar temporadas en “batch”.

- Objetivo: validar que es posible realizar el borrado de temporadas de forma masiva.

- Pre-requisitos: deben existir una o varias temporadas para borrar.
- Entradas: conjunto de identificadores de temporada para borrar.
- Resultado esperado: las temporadas seleccionadas se borran de la base de datos. En caso que alguna temporada no pudiera borrarse, se comunica mediante un mensaje de error, pero en todo caso las temporadas válidas se borran igualmente.

d) Resultado: ok.

7. Insertar Tipos de Precio.

a) Insertar tipo de precio vacío.

- Objetivos: validar que no es posible la inserción de un tipo de precio sin indicar los datos requeridos.
- Pre-requisitos: -
- Entradas: -
- Resultado esperado: el tipo de precio no se guarda en la base de datos y se comunica de los campos que son requeridos para la inserción.
- Resultado: ok.

b) Insertar tipo de precio completo.

- Objetivos: validar la correcta inserción de un tipo de precio en base a datos de entrada requeridos correctos.
- Pre-requisitos: -
- Entradas:
  - Título: Fines de semana.
- Resultado esperado: el tipo de precio se guarda en la base de datos y se comunica el éxito de la operación con un mensaje informativo.
- Resultado: ok.

8. Modificar Tipos de Precio.

a) Modificación total del tipo de precio.

- Objetivos: validar que es posible modificar cualquier dato de un tipo de precio.
- Pre-requisitos: el tipo de precio a modificar existe en el sistema.
- Entradas:
  - Título: Título cambiado.

- Resultado esperado: se muestra un mensaje indicando que los datos del tipo de precio han sido cambiados y se modifican los datos en la base de datos.

- Resultado: ok.

b) Vaciar el tipo de precio.

- Objetivos: validar que no es posible dejar en blanco cualquiera de los datos requeridos de un tipo de precio.

- Pre-requisitos: el tipo de precio debe existir en la base de datos.

- Entradas:

  - Título: -

- Resultado esperado: se muestra un mensaje indicando la falta de los datos requeridos y el tipo de precio no se cambia en la base de datos.

- Resultado: ok.

9. Borrar Tipos de Precio.

a) Borrar tipo de precio sin precios relacionados.

- Objetivos: validar que es posible eliminar un tipo de precio guardado.

- Pre-requisitos: el tipo de precio existe en la base de datos y no está relacionado con ningún precio asignado a una propiedad del catálogo.

- Entradas: -

- Resultado esperado: se muestra un mensaje indicando que el tipo de precio ha sido eliminado y se borra de la base de datos.

- Resultado: ok.

b) Borrar tipos de precio con precios relacionados.

- Objetivos: validar que no es posible eliminar un tipo de precio guardado utilizado para asignar un precio a una propiedad del catálogo.

- Pre-requisitos: el tipo de precio debe existir en la base de datos y debe estar relacionado con algún precio de alguna propiedad del catálogo.

- Entradas: -

- Resultado esperado: se muestra un mensaje indicando que el tipo de precio no ha sido eliminado y no se borra de la base de datos.

- Resultado: ok.

c) Borrar tipos de precio en “batch”.

- Objetivo: validar que es posible realizar el borrado de tipos de precio de forma masiva.
- Pre-requisitos: deben existir uno o varios tipos de precio para borrar.
- Entradas: conjunto de identificadores de tipos de precio para borrar.
- Resultado esperado: los tipos de precio seleccionados se borran de la base de datos. En caso que algún tipo de precio no pudiera borrarse, se comunica mediante un mensaje de error, pero en todo caso los tipos de precio válidos se borran igualmente.

d) Resultado: ok.

10. Listar Tipos de Precio.

a) Filtrar listado de tipos de precio.

- Objetivos: validar que es posible listar los tipos de precio en base a los filtros definidos.
- Pre-requisitos: existen en el sistema varios tipos de precio creados y modificados en momentos aleatoriamente diferentes.
- Entradas:
  - Fecha de creación inicial: 01/01/2012.
  - Fecha de creación final: 31/01/2012.
  - Fecha de modificación inicial: 01/01/2012.
  - Fecha de modificación final: 31/01/2012.
- Resultado esperado: se muestra un listado de tipos de precio cuya fecha de creación y fecha de modificación se corresponde con las fechas indicadas en los datos de entrada.
- Resultado: ok.

11. Insertar Tipos de Inmueble.

a) Insertar tipo de inmueble vacío.

- Objetivos: validar que no es posible la inserción de un tipo de inmueble sin indicar los datos requeridos.
- Pre-requisitos: -
- Entradas: -
- Resultado esperado: el tipo de inmueble no se guarda en la base de datos y se comunica de los campos que son requeridos para la inserción.

- Resultado: ok.

b) Insertar tipo de inmueble completo.

- Objetivos: validar la correcta inserción de un tipo de inmueble en base a datos de entrada requeridos correctos.
- Pre-requisitos: -
- Entradas:
  - Título: Casa rural.
  - Descripción: Casa rural en la sierra.
- Resultado esperado: el tipo de inmueble se guarda en la base de datos y se comunica el éxito de la operación con un mensaje informativo.
- Resultado: ok.

12. Modificar Tipos de Inmueble.

a) Modificación total de tipo de inmueble.

- Objetivos: validar que es posible modificar cualquier dato de un tipo de inmueble.
- Pre-requisitos: el tipo de inmueble a modificar existe en el sistema.
- Entradas:
  - Título: Título cambiado.
  - Descripción: Descripción cambiada.
- Resultado esperado: se muestra un mensaje indicando que los datos del tipo de inmueble han sido cambiados y se modifican los datos en la base de datos.
- Resultado: ok.

b) Vaciar el tipo de inmueble.

- Objetivos: validar que no es posible dejar en blanco cualquiera de los datos requeridos de un tipo de inmueble.
- Pre-requisitos: el tipo de inmueble debe existir en la base de datos.
- Entradas:
  - Título: -
  - Descripción: -

- Resultado esperado: se muestra un mensaje indicando la falta de los datos requeridos y el tipo de inmueble no se cambia en la base de datos.

- Resultado: ok.

13. Borrar Tipos de Inmueble.

a) Borrar tipo de inmueble sin inmuebles relacionados.

- Objetivos: validar que es posible eliminar un tipo de inmueble guardado.

- Pre-requisitos: el tipo de inmueble existe en la base de datos y ninguna propiedad del catálogo se corresponde con este tipo.

- Entradas: -

- Resultado esperado: se muestra un mensaje indicando que el tipo de inmueble ha sido eliminado y se borra de la base de datos.

- Resultado: ok.

b) Borrar tipos de inmueble con inmuebles relacionados.

- Objetivos: validar que no es posible eliminar un tipo de inmueble guardado utilizado como tipo de una propiedad del catálogo de inmuebles.

- Pre-requisitos: el tipo de inmueble debe existir en la base de datos y debe estar relacionado con algún inmueble del catálogo.

- Entradas: -

- Resultado esperado: se muestra un mensaje indicando que el tipo de inmueble no ha sido eliminado y no se borra de la base de datos.

- Resultado: ok.

c) Borrar tipos de inmueble en “batch”.

- Objetivo: validar que es posible realizar el borrado de tipos de inmueble de forma masiva.

- Pre-requisitos: deben existir uno o varios tipos de inmueble para borrar.

- Entradas: conjunto de identificadores de tipo de inmueble para borrar.

- Resultado esperado: los tipos de inmueble seleccionados se borran de la base de datos. En caso que algún tipos de inmueble no pudiera borrarse, se comunica mediante un mensaje de error, pero en todo caso los tipos de inmueble válidos se borran igualmente.

d) Resultado: ok.

14. Insertar Opciones.

a) Insertar opción vacía.

- Objetivos: validar que no es posible la inserción de una opción sin indicar los datos requeridos.
- Pre-requisitos: -
- Entradas: -
- Resultado esperado: la opción no se guarda en la base de datos y se comunica de los campos que son requeridos para la inserción.
- Resultado: ok.

b) Insertar opción completa.

- Objetivos: validar la correcta inserción de una opción en base a datos de entrada requeridos correctos.
- Pre-requisitos: -
- Entradas:
  - Grupo: interior.
  - Tipo: texto.
  - Siempre visible: sí.
  - Valor por defecto: 2.
  - Nombre: Número de terrazas.
  - Descripción: Cantidad de terrazas disponibles.
- Resultado esperado: la opción se guarda en la base de datos y se comunica el éxito de la operación con un mensaje informativo.
- Resultado: ok.

15. Modificar Opciones.

a) Modificación total de la opción.

- Objetivos: validar que es posible modificar cualquier dato de una opción para los inmuebles.
- Pre-requisitos: la opción a modificar existe en el sistema.
- Entradas:
  - Grupo: exterior.

- Tipo: texto.
- Siempre visible: no.
- Valor por defecto: 1.
- Nombre: Terrazas.
- Descripción: Terrazas disponibles.

- Resultado esperado: se muestra un mensaje indicando que los datos de la opción han sido cambiados y se modifican los datos en la base de datos.
- Resultado: ok.

b) Vaciar la opción.

- Objetivos: validar que no es posible dejar en blanco cualquiera de los datos requeridos de una opción.
- Pre-requisitos: la opción debe existir en la base de datos.
- Entradas: -
- Resultado esperado: se muestra un mensaje indicando la falta de los datos requeridos y la opción no se cambia en la base de datos.
- Resultado: ok.

16. Borrar Opciones.

a) Borrar opción distinta de “Número de plazas”.

- Objetivos: validar que es posible eliminar una opción guardada.
- Pre-requisitos: la opción a borrar existe en la base de datos y no se corresponde con la opción “Número de plazas”.
- Entradas: -
- Resultado esperado: se muestra un mensaje indicando que la opción ha sido eliminada y se borra de la base de datos.
- Resultado: ok.

b) Borrar opción “Número de plazas”.

- Objetivos: validar que no es posible eliminar la opción “Número de plazas”.
- Pre-requisitos: la opción debe existir en la base de datos y se corresponde con la opción “Número de plazas”.
- Entradas: -

- Resultado esperado: se muestra un mensaje indicando que la opción no ha sido eliminada y no se borra de la base de datos.

- Resultado: ok.

c) Borrar opciones en “batch”.

- Objetivo: validar que es posible realizar el borrado de opciones de forma masiva.

- Pre-requisitos: deben existir una o varias opciones para borrar.

- Entradas: conjunto de identificadores de opción para borrar.

- Resultado esperado: las opciones seleccionadas se borran de la base de datos. En caso que alguna opción no pudiera borrarse, se comunica mediante un mensaje de error, pero en todo caso las opciones válidas se borran igualmente.

d) Resultado: ok.

17. Listar Opciones.

a) Filtrar listado de opciones.

- Objetivos: validar que es posible listar las opciones en base a los filtros definidos.

- Pre-requisitos: existen en el sistema varias opciones creadas.

- Entradas:

  - Grupo: interior.

  - Tipo: texto.

  - Siempre visible: 1.

  - Valor por defecto: 1.

- Resultado esperado: se muestra un listado de opciones cuyos datos se corresponden con los valores de los filtros indicados.

- Resultado: ok.

18. Insertar Inmuebles.

a) Insertar inmueble vacío.

- Objetivos: validar que no es posible la inserción de un inmueble sin indicar los datos requeridos.

- Pre-requisitos: -

- Entradas: -

- Resultado esperado: el inmueble no se guarda en la base de datos y se comunica de los campos que son requeridos para la inserción.
- Resultado: ok.

b) Insertar inmueble completo.

- Objetivos: validar la correcta inserción de un inmueble en base a datos de entrada requeridos correctos.
- Pre-requisitos: -
- Entradas:
  - Código: C001.
  - Operación: Casa de Vacaciones.
  - Tipo: villa.
  - Zona: Maladroxia.
  - Publicado: sí.
  - Sugerido: sí.
  - Last Minute: no.
  - Título: Villa García.
  - Descripción: Villa espaciosa de lujo.
  - Descripción larga: Villa espaciosa de lujo en Maladroxia.
  - Texto de búsqueda: villa lujo, vila de luxo, villa lujos
- Resultado esperado: el inmueble se guarda en la base de datos y se comunica el éxito de la operación con un mensaje informativo.
- Resultado: ok.

19. Modificar Inmuebles.

a) Modificación total del inmueble.

- Objetivos: validar que es posible modificar cualquier dato de un inmueble del catálogo.
- Pre-requisitos: el inmueble a modificar existe en el sistema.
- Entradas:
  - Código: C002.
  - Operación: Casa de Vacaciones.

- Tipo: villa.
- Zona: Co.
- Publicado: no.
- Sugerido: no.
- Last Minute: no.
- Título: Villa Casula.
- Descripción: Villa en ruinas.
- Descripción larga: Villa en ruinas en Co.
- Texto de búsqueda: villa ruina, vila ruinosa, villa ruinas

- Resultado esperado: se muestra un mensaje indicando que los datos del inmueble han sido cambiados y se modifican los datos en la base de datos.
- Resultado: ok.

b) Vaciar el inmueble.

- Objetivos: validar que no es posible dejar en blanco cualquiera de los datos requeridos de un inmueble.
- Pre-requisitos: el inmueble debe existir en la base de datos.
- Entradas: -
- Resultado esperado: se muestra un mensaje indicando la falta de los datos requeridos y el inmueble no se cambia en la base de datos.
- Resultado: ok.

c) Indicar valores a las opciones del inmueble.

- Objetivos: validar que es posible cambiar los valores de las opciones o servicios adicionales de un inmueble.
- Pre-requisitos: el inmueble está dado de alta en el catálogo.
- Entradas: aleatoriamente por cada opción damos un valor o marcamos su casilla (según el tipo de la opción), e indicamos si debemos mostrarla o esconderla.
- Resultado esperado: se muestra un mensaje indicando que los datos del inmueble han sido cambiados y se modifican los datos en la base de datos.
- Resultado: ok.

20. Borrar Inmuebles.

a) Borrar inmueble

- Objetivos: validar que es posible eliminar un inmueble guardado en el catálogo de la aplicación.
- Pre-requisitos: el inmueble a borrar existe en la base de datos.
- Entradas: -
- Resultado esperado: se muestra un mensaje indicando que el inmueble ha sido eliminado y se borra de la base de datos junto con sus datos relacionados.
- Resultado: ok.

b) Borrar inmuebles en “batch”.

- Objetivo: validar que es posible realizar el borrado de inmuebles de forma masiva.
- Pre-requisitos: deben existir uno o varios inmuebles para borrar.
- Entradas: conjunto de identificadores de inmueble para borrar.
- Resultado esperado: los inmuebles seleccionados se borran de la base de datos. En caso que algún inmueble no pudiera borrarse, se comunica mediante un mensaje de error, pero en todo caso los inmuebles válidos se borran igualmente.

c) Resultado: ok.

21. Listar Inmuebles.

a) Filtrar listado de inmuebles.

- Objetivos: validar que es posible listar los inmuebles en base a los filtros definidos.
- Pre-requisitos: existen en el sistema varios inmuebles dados de alta.
- Entradas:
  - Código: -
  - Número de plazas: 2.
  - Tipo: villa.
  - Zona: Maladroxia.
  - Publicado: sí.
  - Sugerido: no.

- Last Minute: no.

- Resultado esperado: se muestra un listado de inmuebles cuyos datos se corresponden con los valores de los filtros indicados.

- Resultado: ok.

22. Ordenar Inmuebles.

a) Subir inmueble.

- Objetivos: validar que es posible ordenar un inmueble en el listado del catálogo subiendo la prioridad en una posición.

- Pre-requisitos: el inmueble existe en la base de datos y existen otros inmuebles con mayor prioridad.

- Entradas: -

- Resultado esperado: el inmueble se sitúa una posición por delante de la anterior en el listado del catálogo.

- Resultado: ok.

b) Bajar inmueble.

- Objetivos: validar que es posible ordenar un inmueble en el listado del catálogo bajando la prioridad en una posición.

- Pre-requisitos: el inmueble existe en la base de datos y existen otros inmuebles con menor prioridad.

- Entradas: -

- Resultado esperado: el inmueble se sitúa una posición por detrás de la anterior en el listado del catálogo.

- Resultado: ok.

c) Subir inmueble al tope.

- Objetivos: validar que es posible ordenar un inmueble en el listado del catálogo subiendo la prioridad al máximo y ponerlo el primero de la lista.

- Pre-requisitos: el inmueble existe en la base de datos y existen otros inmuebles con mayor prioridad.

- Entradas: -

- Resultado esperado: el inmueble se sitúa en primera posición en el listado del catálogo.

- Resultado: ok.

23. Agregar imágenes a la galería de imágenes del Inmueble

a) Subir imágenes válidas.

- Objetivos: validar que es posible agregar imágenes que cumplen los requisitos al catálogo de imágenes de un inmueble.
- Pre-requisitos: el inmueble existe en la base de datos.
- Entradas: se seleccionan del sistema un conjunto de archivos con formato de imagen que no superen los 2Mb de tamaño.
- Resultado esperado: las imágenes del inmueble se almacenan en el sistema y se visualizan sus miniaturas en el listado del catálogo de imágenes.
- Resultado: ok.

b) Subir imágenes incorrectas.

- Objetivos: validar que no es posible agregar imágenes que no cumplen los requisitos al catálogo de imágenes de un inmueble.
- Pre-requisitos: el inmueble existe en la base de datos.
- Entradas: se seleccionan del sistema un conjunto de archivos sin formato de imagen o que superen los 2Mb de tamaño.
- Resultado esperado: las imágenes del inmueble no se almacenan en el sistema, se comunica del error al cargar las imágenes y no se visualizan sus miniaturas en el listado del catálogo de imágenes.
- Resultado: ok.

24. Eliminar imágenes a la galería de imágenes del Inmueble

a) Borrar imágenes.

- Objetivos: validar que es posible borrar imágenes de la galería de imágenes de un inmueble.
- Pre-requisitos: el inmueble existe en la base de datos y tiene imágenes en su galería.
- Entradas: se seleccionan de la galería un conjunto de imágenes.
- Resultado esperado: las imágenes del inmueble se borran del sistema y se dejan de visualizar sus miniaturas en el listado del catálogo de imágenes.
- Resultado: ok.

25. Ocupar Inmueble

a) Reservar un rango de fechas.

- Objetivos: validar que es posible definir un rango de fechas para ocupar un inmueble.
- Pre-requisitos: el inmueble existe en la base de datos.
- Entradas:
  - Fecha inicio: 01/07/2012.
  - Fecha fin: 15/07/2012.
- Resultado esperado: el rango de fechas queda definido en el calendario para las fechas definidas.
- Resultado: ok.

b) Modificar datos informativos del rango.

- Objetivos: validar que es posible modificar los datos de información de un rango de fechas: nombre, comentario y si la reserva está confirmada.
- Pre-requisitos: el inmueble y el rango de fechas existe en la base de datos.
- Entradas:
  - Nombre: Santiago García.
  - Comentarios: Llegará a primera hora de la mañana.
  - Confirmada: sí.
- Resultado esperado: la información se almacena en la base de datos.
- Resultado: ok.

26. Desocupar Inmueble

a) Borrar un rango de fechas.

- Objetivos: validar que es posible eliminar un rango de fechas de ocupación de un inmueble.
- Pre-requisitos: el rango existe en la base de datos.
- Entradas: el identificador del rango a borrar.
- Resultado esperado: el rango de fechas queda eliminado del calendario del inmueble.
- Resultado: ok.

27. Agregar precio de Inmueble.

a) Insertar precio sin valor.

- Objetivos: validar que no es posible la inserción de precio sin indicar los datos requeridos.
- Pre-requisitos: -
- Entradas: -
- Resultado esperado: el precio no se guarda en la base de datos y se comunica de los campos que son requeridos para la inserción.
- Resultado: ok.

b) Insertar precio completo.

- Objetivos: validar la correcta inserción de un precio en base a datos de entrada requeridos correctos.
- Pre-requisitos: -
- Entradas:
  - Temporada: mayo.
  - Tipo de precio: semanal.
  - Precio: 200.
- Resultado esperado: el precio se guarda en la base de datos y se comunica el éxito de la operación con un mensaje informativo.
- Resultado: ok.

c) Insertar precio repetido.

- Objetivos: validar que no es posible guardar un precio que ya se ha definido para el conjunto inmueble-temporada-tipo.
- Pre-requisitos: que ya exista en la base de datos un valor para el precio.
- Entradas:
  - Temporada: mayo.
  - Tipo de precio: semanal.
  - Precio: 100.
- Resultado esperado: el precio no se guarda en la base de datos y se comunica mediante un mensaje informativo que ya se definido un valor para ese precio.

- Resultado: ok.

28. Modificar precio de Inmueble.

a) Modificación total de tipo de inmueble.

- Objetivos: validar que es posible modificar cualquier dato de un precio de inmueble.
- Pre-requisitos: el precio de inmueble a modificar existe en el sistema.
- Entradas:
  - Temporada: junio.
  - Tipo de precio: semanal.
  - Precio: 150.
- Resultado esperado: se muestra un mensaje indicando que los datos del precio del inmueble han sido cambiados y se modifican los datos en la base de datos.
- Resultado: ok.

b) Vaciar el precio de inmueble.

- Objetivos: validar que no es posible dejar en blanco el valor de un precio de inmueble.
- Pre-requisitos: el tipo de inmueble debe existir en la base de datos.
- Entradas:
  - Temporada: junio.
  - Tipo de precio: semanal.
  - Precio: -
- Resultado esperado: se muestra un mensaje indicando la falta de los datos requeridos y el precio de inmueble no se cambia en la base de datos.
- Resultado: ok.

29. Borrar precios de Inmueble.

a) Borrar precio.

- Objetivos: validar que es posible eliminar un precio de inmueble guardado.
- Pre-requisitos: el precio existe en la base de datos.
- Entradas: -

- Resultado esperado: se muestra un mensaje indicando que el precio de inmueble ha sido eliminado y se borra de la base de datos.
- Resultado: ok.

b) Borrar precio en “batch”.

- Objetivo: validar que es posible realizar el borrado de precios de inmueble de forma masiva.
- Pre-requisitos: deben existir uno o varios precios de inmueble para borrar.
- Entradas: conjunto de identificadores de tipo de inmueble para borrar.
- Resultado esperado: los precios del inmueble seleccionados se borran de la base de datos.
- Resultado: ok.

30. Insertar Página.

a) Insertar página vacía.

- Objetivos: validar que no es posible la inserción de páginas sin indicar los datos requeridos.
- Pre-requisitos: -
- Entradas: -
- Resultado esperado: la página no se guarda en la base de datos y se comunica de los campos que son requeridos para la inserción.
- Resultado: ok.

b) Insertar página completa.

- Objetivos: validar la correcta inserción de una página en base a datos de entrada requeridos correctos.
- Pre-requisitos: -
- Entradas:
  - Página padre: Agencia.
  - Plantilla: 2 fotos y 2 textos.
  - Publicada: sí.
  - Ver en menú: sí.
  - Título: Página de prueba.
  - Descripción: una página de prueba cualquiera.

- Palabras clave: página de prueba, página cualquiera.

- Resultado esperado: la página se guarda en la base de datos y se comunica el éxito de la operación con un mensaje informativo.

- Resultado: ok.

31. Modificar Página.

a) Vaciar página.

- Objetivos: validar que no es posible modificar una página y dejar en blanco los datos requeridos.

- Pre-requisitos: la página a modificar debe existir en la base de datos.

- Entradas: -

- Resultado esperado: la página no se cambia en la base de datos y se comunica de los campos que son requeridos para la modificación.

- Resultado: ok.

b) Cambiar la página por completo.

- Objetivos: validar que es posible cambiar por completo los datos de una página en base a datos de entrada requeridos correctos.

- Pre-requisitos: -

- Entradas:

  - Página padre: Eventos.

  - Plantilla: página simple.

  - Publicada: no.

  - Ver en menú: no.

  - Título: Página de un evento.

  - Descripción: una página para publicar un evento.

  - Palabras clave: página de evento, evento público.

- Resultado esperado: la página se cambia en la base de datos y se comunica el éxito de la operación con un mensaje informativo.

- Resultado: ok.

32. Borrar Página.

a) Borrar página.

- Objetivos: validar que es posible eliminar una página guardada.
- Pre-requisitos: la página existe en la base de datos.
- Entradas: -
- Resultado esperado: se muestra un mensaje indicando que la página ha sido eliminada y se borra de la base de datos.
- Resultado: ok.

b) Borrar página en “batch”.

- Objetivo: validar que es posible realizar el borrado de páginas de forma masiva.
- Pre-requisitos: deben existir una o varias páginas para borrar.
- Entradas: conjunto de identificadores de páginas para borrar.
- Resultado esperado: las páginas seleccionadas se borran de la base de datos.
- Resultado: ok.

33. Autentificarse como Administrador

a) Log-in correcto.

- Objetivos: validar que es posible abrir sesión como administrador y acceder a la zona de administración.
- Pre-requisitos: debe existir en el sistema un usuario con privilegios de administrador.
- Entradas:
  - Usuario: admin.
  - Contraseña: admin.
- Resultado esperado: se abre sesión de administrador y se accede a la página principal de administración.
- Resultado: ok.

b) Log-in fallido.

- Objetivos: validar que no es posible abrir sesión como administrador sin introducir los datos de acceso válidos.

- Pre-requisitos: debe existir en el sistema un usuario con privilegios de administrador.

- Entradas:

  - Usuario: admin.

  - Contraseña: otra.

- Resultado esperado: se comunica mediante un mensaje de error que los datos de acceso no son correctos.

- Resultado: ok.

34. Cerrar sesión de Administrador.

a) Log-out.

- Objetivos: validar que es posible cerrar sesión como administrador para no tener acceso a la zona de administración.

- Pre-requisitos: debe estar abierta una sesión de administrador.

- Entradas: -

- Resultado esperado: se cierra la sesión de administración y se muestra la pantalla de entrada de datos de acceso para la zona de administración.

- Resultado: ok.

35. Cambiar Idioma del Portal.

a) Cambiar de italiano a inglés.

- Objetivos: validar que es posible cambiar el idioma del portal y visualizar el contenido en el idioma indicado.

- Pre-requisitos: el contenido de las páginas debe tener la traducción correspondiente.

- Entradas: -

- Resultado esperado: se recarga la “home” del portal y a partir de entonces se visualizan los contenidos en el idioma que se ha seleccionado. En caso de que un contenido no tenga traducción, se visualiza el contenido específico con su valor por defecto si lo tuviera, en otro caso no se muestra.

- Resultado: ok.

36. Listar Inmuebles del Portal.

a) Filtrar inmuebles del portal.

- Objetivos: validar que es posible listar el catálogo en el portal público y filtrar el resultado del listado según los parámetros de búsqueda.
- Pre-requisitos: deben existir inmuebles dados de alta en el catálogo.
- Entradas: -
  - Fecha de llegada: 08/07/2012.
  - Fecha de partida: 08/08/2012.
  - Zona: Maladroxia.
  - Tipo: Apartamento.
- Resultado esperado: se visualiza un listado de inmuebles del catálogo situado en la zona y el tipo indicado, y que están disponibles en alquiler en el periodo de fechas señalado.
- Resultado: ok.

37. Búsqueda rápida de Inmuebles.

a) Buscar por palabras clave.

- Objetivos: validar que es posible listar el catálogo en el portal público filtrando el resultado del listado según palabras clave.
- Pre-requisitos: deben existir inmuebles dados de alta en el catálogo.
- Entradas: -
  - Palabras clave: le saline.
- Resultado esperado: se visualiza un listado de inmuebles del catálogo que contienen en alguna parte de sus datos almacenados la cadena indicada como palabras clave.
- Resultado: ok.

38. Ver detalle de Inmueble.

a) Mostrar detalle.

- Objetivos: validar que es posible visualizar toda la información relacionada con un inmueble determinado de forma extendida.
- Pre-requisitos: debe existir un inmueble dado de alta en el catálogo.
- Entradas: -
- Resultado esperado: se visualiza de forma detallada y categorizada toda la información almacenada en la base de datos relativa a un inmueble.
- Resultado: ok.

39. Contactar.

a) Formulario de contacto válido.

- Objetivos: validar que es posible contactar con el administrador del portal mediante el uso de un formulario específico que ha sido completado con datos requeridos válidos.
- Pre-requisitos: -.
- Entradas: -
  - Nombre: Santiago García.
  - Ciudad: Torrent.
  - Llegada: 01/07/2012.
  - Partida: 15/07/2012.
  - E-mail: info@sangarbe.com.
  - Régimen: Bed & Breakfast.
  - Ciudad del inmueble: Maladroxia.
  - Tipo del inmueble: Apartamento.
  - Localización del Inmueble: Mar.
  - Número de adultos: 2.
  - Número de niños: -
  - Número de habitaciones: 2.
  - Comentario: -
- Resultado esperado: se construye un correo y se envía por e-mail a la cuenta de e-mail del administrador, y se muestra una pantalla de confirmación del envío realizado.
- Resultado: ok.

b) Formulario de contacto incorrecto.

- Objetivos: validar que no es posible contactar con el administrador del portal mediante el uso de un formulario específico si no se indican datos requeridos correctos.
- Pre-requisitos: -.
- Entradas: -

- Resultado esperado: se muestra el formulario con los datos introducidos y mensajes de error sobre los campos incorrectos. El e-mail al administrador no es enviado.

- Resultado: ok.

3. Pruebas de visualización

Resoluciones

Debido al carácter publicitario del portal, es importante también constatar que la apariencia del mismo no resulte desagradable para el usuario. La zona pública ha sido desarrollada para una resolución óptima de 1024 píxeles de ancho, por lo que en resoluciones menores los navegadores web necesitan mostrar barras de desplazamiento para poder visualizar todo el contenido.

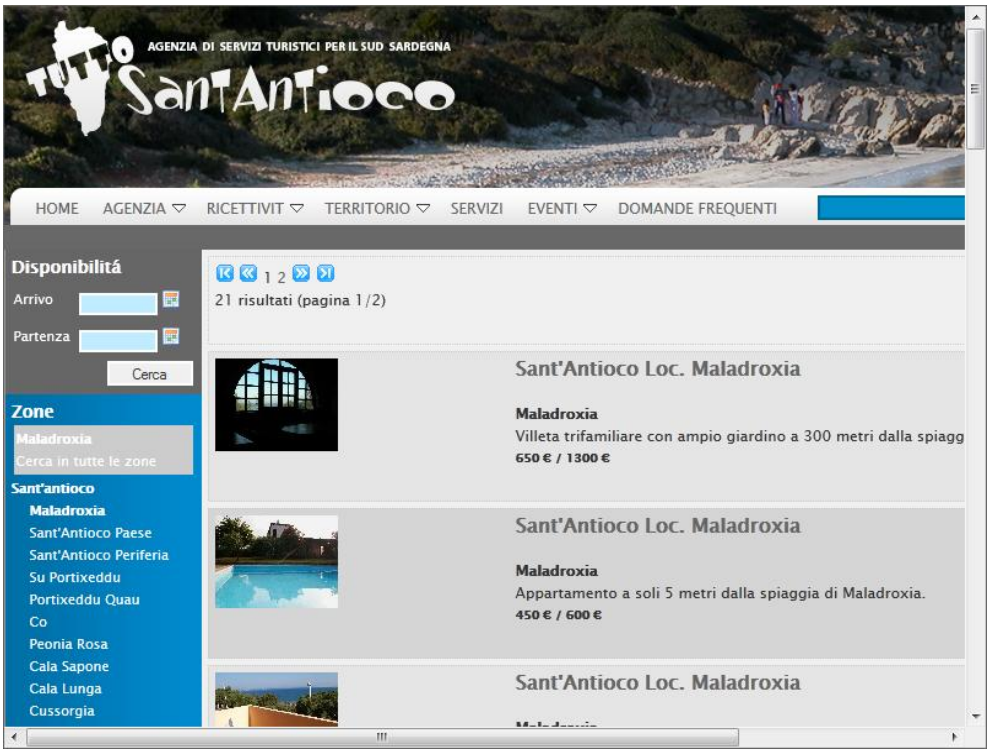


Ilustración 37: Visualización a 800x600

Para resoluciones mayores, el contenido se muestra correctamente como en la resolución por defecto. Aunque para resoluciones con un ancho mayor a 1500 píxeles la imagen de la cabecera se repite a lo ancho de la pantalla, ya que éste ancho "límite" se corresponde con el ancho de la imagen utilizada para la cabecera de la página.

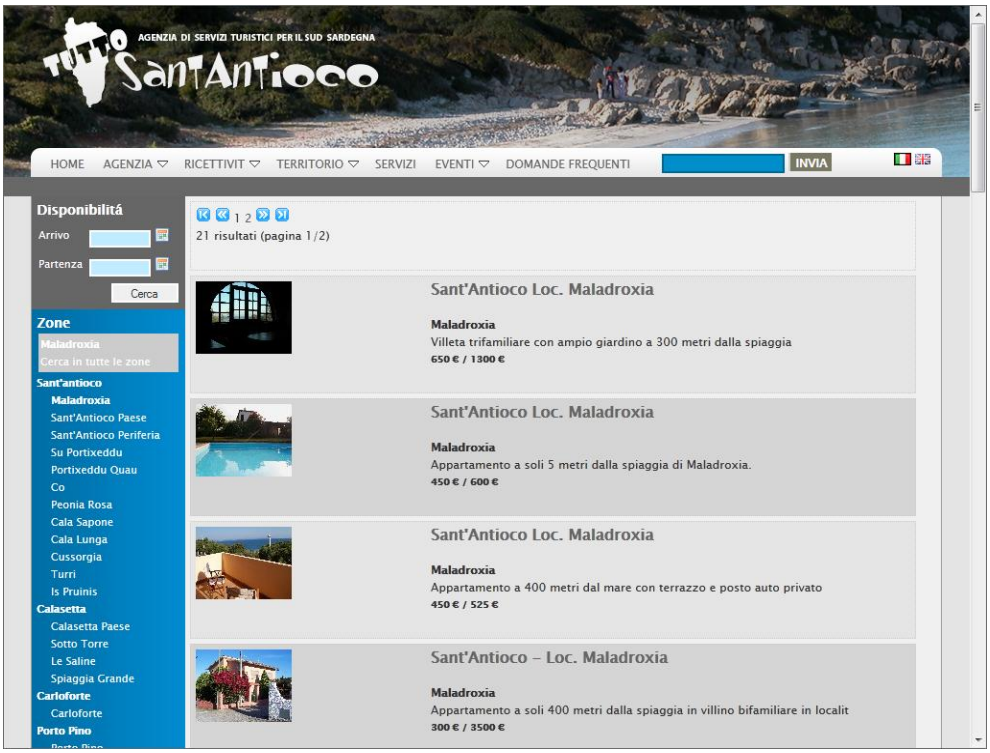


Ilustración 38: Visualización a 1024x768

Se pueden utilizar servicios online para comprobar el resultado de visualizar el portal en diferentes resoluciones (http://www.viewlike.us: Página que permite visualizar una dirección de Internet con diferentes resoluciones de pantalla), ya que la aplicación se encuentra publicada en Internet a día de hoy (http://pfc.sangarbe.com: Dirección de Internet donde se ha publicado el proyecto).

Navegadores

Desafortunadamente, cada compañía desarrolladora de navegadores web implementa según su criterio la forma de representar algunos elementos de los documentos web. Estas diferencias obligan en muchas ocasiones al desarrollador web a adaptar de forma específica parte del código, para que el documento se visualice de igual manera en diferentes navegadores. Esto es especialmente común para el código *Javascript*, que aporta comportamiento dinámico a las páginas web.

Por tanto, el portal ha sido probado en los navegadores web mayoritarios a día de la creación de esta memoria (IExplorer, Chrome y Firefox). Para hacernos una idea, a continuación se muestra una estadística del porcentaje de uso a nivel global de los 5 navegadores más importantes del mercado:

1. Internet Explorer: 37,45%
2. Google Chrome: 28,4%
3. Mozzilla Firefox: 24,78%
4. Safari: 6,62%

5. Opera: 1,95%

A continuación podemos ver una captura de la misma página del portal mostrada en los tres navegadores principales:

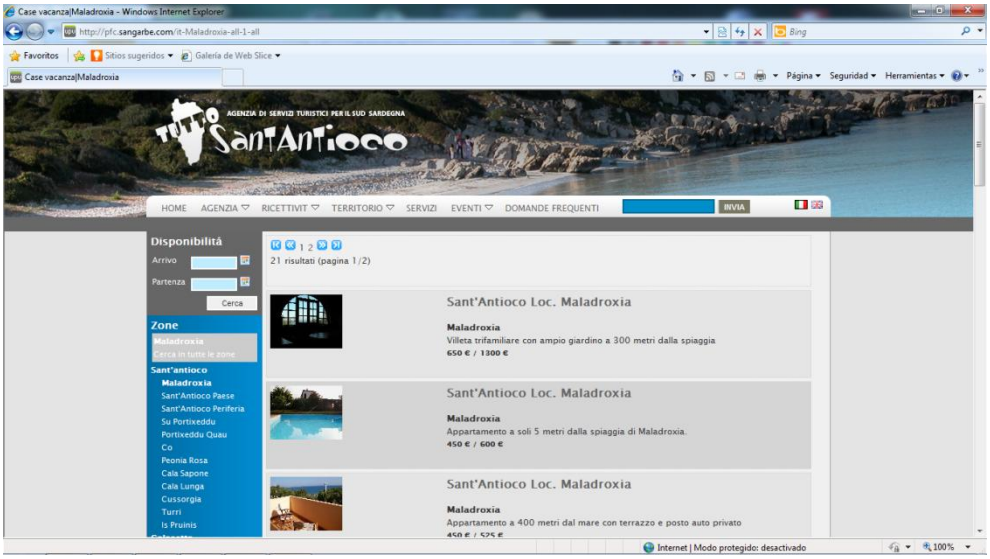


Ilustración 39: Visualización en Internet Explorer

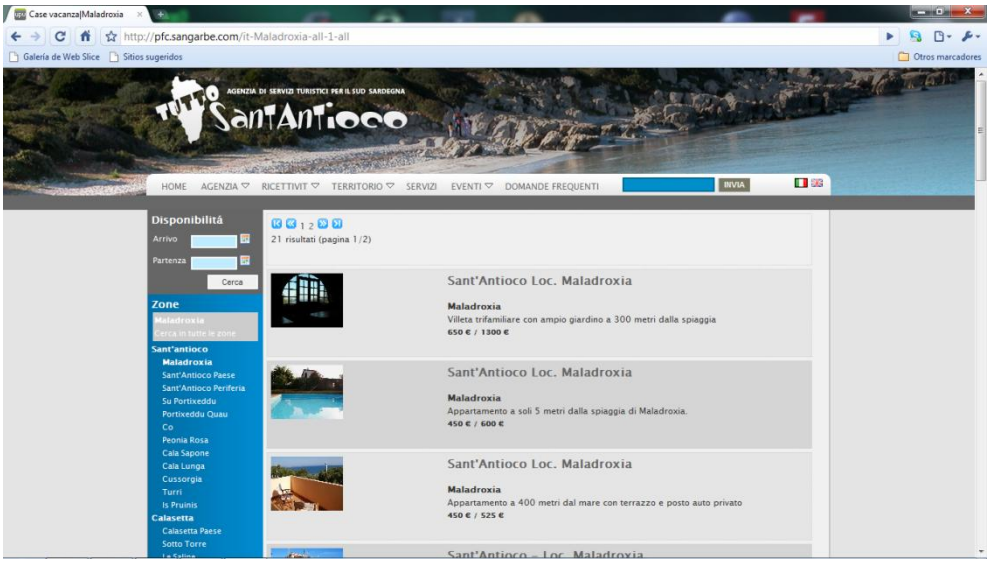


Ilustración 40: Visualización en Google Chrome

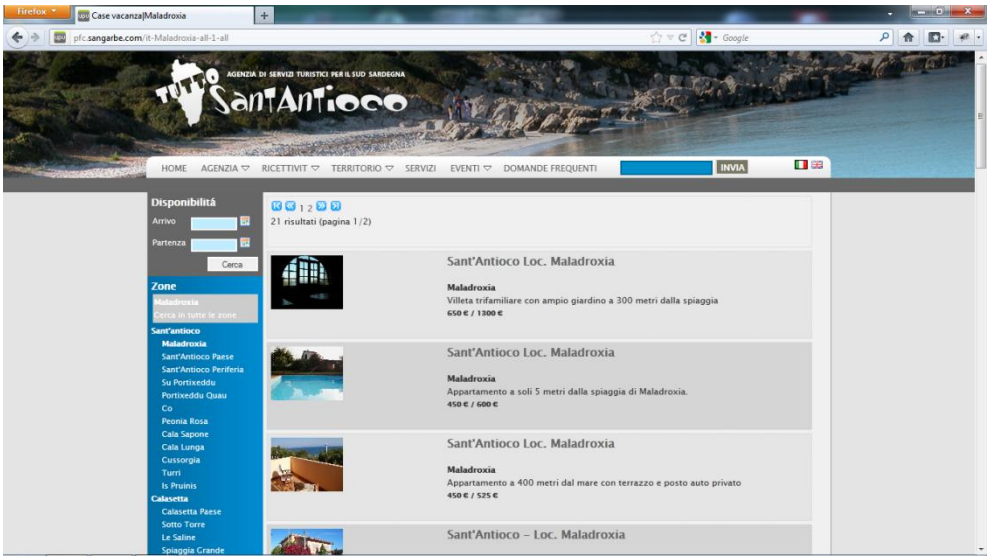


Ilustración 41: Visualización en Mozzilla Firefox

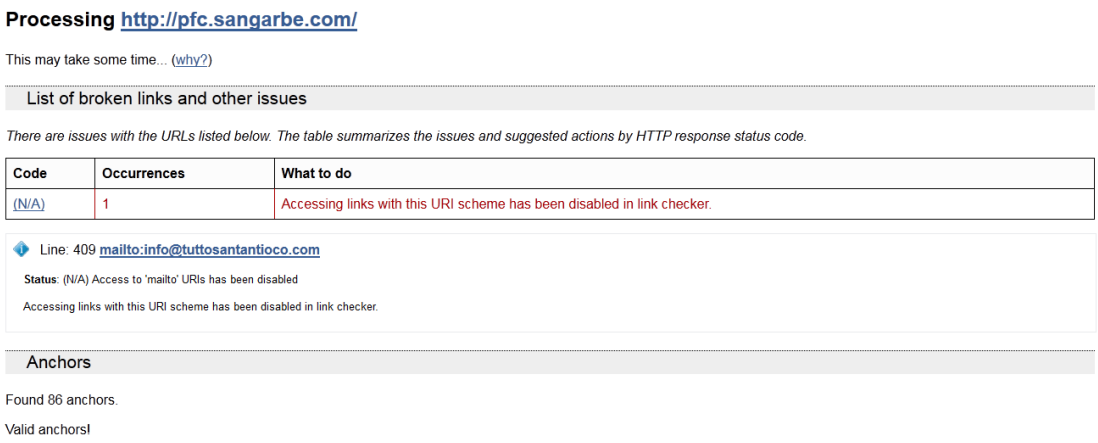
El resultado obtenido es prácticamente idéntico en los navegadores probados, lo que se considera aceptable y libre de errores visuales.

Comprobación de enlaces

Puesto que las páginas del portal se crean dinámicamente mediante las herramientas de administración de la parte privada, el número de enlaces puede ser variable. Pero validar los enlaces generados por el CMS sirve de ayuda para verificar la inexistencia de enlaces mal formados (lo que afecta tanto al comportamiento del CMS como al posicionamiento del portal en los *buscadores web*) que resultan en errores 404 del servidor (error de página no encontrada).

Para esta validación se ha utilizado el validador de links del W3C (World Wide Web Consortium, comunidad internacional que desarrolla estándares para la web), que podemos encontrar en http://validator.w3.org/checklink.

Puesto que el CMS permite agregar ilimitadas páginas de contenido al portal, y ordenarlas en una estructura de menú con múltiples niveles, se ha restringido la comprobación hasta 3 niveles de profundidad. Esto permitirá realizar una exploración del portal lo bastante amplia como para asegurar la validez de los links creados por el CMS.



Processing http://pfc.sangarbe.com/

This may take some time... (why?)

List of broken links and other issues

There are issues with the URLs listed below. The table summarizes the issues and suggested actions by HTTP response status code.

Code	Occurrences	What to do
(N/A)	1	Accessing links with this URI scheme has been disabled in link checker.

Line: 409 mailto:info@tuttosantantioco.com

Status: (N/A) Access to 'mailto' URIs has been disabled

Accessing links with this URI scheme has been disabled in link checker.

Anchors

Found 86 anchors.

Valid anchors!

Ilustración 42: Captura del validador de enlaces

En la imagen anterior queda reflejado el resultado del validador para la página principal, aunque el validador ha obtenido idéntico resultado para cada página en 3 niveles de profundidad del portal. Como puede observarse, cada enlace conduce a una página válida.

Capítulo VII. Conclusiones

A continuación, se aportan las conclusiones obtenidas del trabajo realizado desde la óptica subjetiva del autor. Se ponen de manifiesto las dificultades encontradas y las aportaciones personales conseguidas en el transcurso de creación del proyecto. Por último se enumeran posibles mejoras o ampliaciones para versiones futuras del producto.

1. Trabajo realizado

La aplicación desarrollada es el resultado del trabajo realizado para la implementación de los requisitos establecidos en el Capítulo II de este documento. El cumplimiento de dichos requisitos es la premisa adoptada para su desarrollo, por lo que satisface el acuerdo alcanzado con el cliente en su totalidad.

Se han cuidado los detalles para cumplir de la mejor forma posible con las finalidades principales de la aplicación. Por un lado se aporta una interfaz sencilla e intuitiva, especialmente en la zona privada de gestión, que permite realizar las tareas administrativas de forma óptima. Por otro lado se han vigilado el aspecto y la usabilidad de la parte pública, siguiendo en lo posible los estándares existentes, para servir eficazmente de reclamo publicitario.

El producto obtenido es un producto personalizado y adaptado al negocio de un cliente en particular. No obstante, con la dedicación de no demasiado trabajo extra y algún cambio en la lógica implementada, podría obtenerse un producto genérico adaptado al sector turístico y de alquiler de inmuebles.

Cabría preguntarse si valdría la pena dedicar dichos recursos y obtener un producto comercial que aportara algún beneficio adicional, principalmente económico.

2. Valoración personal

La finalización de este proyecto supone el cierre de una etapa muy importante. Ha sido un trayecto duro, lento y lleno de dificultades, pero cabe decir también, que no ha estado carente de ciertas satisfacciones. Pese a considerarse un pequeño éxito, sin duda podría haberse hecho mejor. Pero hay que aceptar que las circunstancias y la falta de tiempo en ocasiones no lo permiten.

Como consecuencia del trabajo realizado, se ha podido obtener una visión global de todo el proceso de desarrollo de una aplicación web desde cero. Además ha

permitido enfrentarse a problemas reales, poniendo a prueba todos los conocimientos obtenidos tanto en la época académica como en la profesional.

El trato con el cliente ha sido especialmente complejo, sobre todo para la definición de los requisitos de la aplicación. Al tratarse de un cliente real (no siempre disponible por motivos laborales) de nacionalidad Italiana (que ha requerido de un intermediario), ha existido un problema notable de comunicación, lo que ha provocado una pérdida de agilidad importante en el desarrollo.

Aunque la selección de las tecnologías empleadas para este trabajo ha sido realizada en base al conocimiento previo de algunas de ellas, se ha intentado mantener un óptimo ratio calidad/agilidad. Esto ha provocado la necesidad de profundizar en el conocimiento de algunas de las tecnologías usadas, con la consecuente pérdida de agilidad en el desarrollo, debido al tiempo empleado en la curva de aprendizaje.

En conjunto, la realización de este proyecto ha sido una experiencia muy positiva, que ha servido para aprender, mejorar y evolucionar, tanto en lo profesional como en lo personal. Ha conseguido aportar el perfeccionamiento en el uso de las tecnologías utilizadas, adquirir experiencia en el trato con clientes, poner a prueba la capacidad de gestión del tiempo y desarrollar aptitudes para la generación de documentación técnica.

Para concluir, comentar que el campo de las aplicaciones web es muy amplio, con mucho potencial y en continua expansión. No obstante, presenta una competencia extrema, en gran parte debido a un intrusismo desmedido. Esta circunstancia favorece la aparición de malos profesionales, que genera mucha desconfianza en los clientes y provoca el menosprecio del trabajo implicado en este tipo de sector.

3. Futuras mejoras

La creación de este proyecto ha tenido un doble objetivo principal: satisfacer las necesidades básicas de un cliente real y la construcción de un buen proyecto final de carrera. Por tanto, se ha intentado cubrir dichas necesidades teniendo en cuenta los recursos limitados (sobre todo de tiempo) que se disponen en la realización de un proyecto final de carrera. En consecuencia, existen funcionalidades que no han podido implementarse pero que podrían ser muy interesantes para mejorar y ampliar la aplicación en el futuro:

- Sistema de registro y panel privado para inquilinos: este sistema ayudaría a obtener un mejor *feedback* de los clientes, permitiendo a éstos el acceso a posibles funcionalidades privadas.

- Registro mediante redes sociales: podría facilitarse el registro mediante el uso de conectividad con redes sociales mayoritarias como *facebook* o *twitter*.

- Marketing en Redes Sociales: publicación automática en redes sociales (*facebook* o *twitter*) de ofertas, promociones, anuncios con respecto a inmuebles y otras acciones, como modificación de precios, rebajas *last minute*, etc.

- Sistema de reservas online: sistema por el cual un cliente interactuaría directamente con el sistema para realizar reservas sin necesidad de intervención o confirmación del administrador.

- Sistema de pagos on-line: inclusión de la posibilidad de realizar los pagos de forma electrónica mediante TPV virtual u otro tipo de pasarela de pago como *PayPal*.

- Sistema de puntuación de inmuebles: permitiría a un usuario registrado puntuar y valorar los inmuebles en los que ha realizado reservas.

- Sistema de comentarios para inmuebles: permitiría a usuarios registrados escribir comentarios públicos sobre inmuebles.

- Ampliar la edición de contenido del CMS: poder gestionar otro tipo de contenido de las páginas creadas, por ejemplo: poder incrustar videos, o galerías de imágenes.

- Sistema de creación de plantillas de páginas: para poder crear las plantillas con las que se crean las páginas de contenido desde la propia aplicación.

Capítulo VIII. Bibliografía y referencias

1. Estructura del documento

En el presente documento o memoria se describen detalladamente las fases de construcción de un Sistema de Gestión de Contenidos web para una agencia turística. Este documento está dividido en capítulos, cada uno de los cuales representa una fase del ciclo de vida del sistema y su evolución, así como la documentación necesaria para entender la construcción del mismo. A continuación se describen cada uno de esos capítulos.

- **Introducción**.

  En este capítulo se hace una pequeña presentación del proyecto al lector. Se describen levemente los objetivos a cumplir por el proyecto, la motivación para su realización y el contexto en el que ha sido construido.

- **Especificación de requisitos**.

  Capítulo dedicado a describir el conjunto de necesidades que el proyecto debe cumplir. Para la descripción de tales requisitos se sigue la norma IEEE std. 830-1998.

- **Análisis**.

  Este capítulo especifica el modelo visual descriptivo de la aplicación, construido a partir de los requisitos obtenidos. Para la construcción del modelo se hace uso de un conjunto de notaciones y diagramas del estándar UML.

- **Diseño**.

  En este capítulo se describen las opciones y soluciones adoptadas para el diseño de la aplicación. También se realiza un estudio en mayor profundidad del patrón de diseño MVC.

- **Implementación**.

  En este apartado se describen las tecnologías utilizadas para la construcción de la aplicación, en especial del framework *Symfony*, y algunos detalles de implementación en base a dicho framework.

- **Evaluación y pruebas**.

  Capítulo encargado de exponer las diferentes pruebas realizadas a la aplicación y el comportamiento y resultados obtenidos de las mismas.

- **Conclusiones**.

  Valoración personal del autor sobre el proyecto y el trabajo realizado, así como las anotaciones para posibles futuras mejoras de la aplicación.

- **Bibliografía y referencias**.

  Listado de la documentación consultada para la realización del proyecto. También se comenta la estructura de capítulos del documento.

2. Bibliografía

- *Bear Bibeault y Yehuda Katz "JQuery in Action".*
- *Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides "Design Patterns: Elements of Reusable Object-Oriented Software".*
- *http://pfc.sangarbe.com: Dirección de Internet donde se ha publicado el proyecto.*
- *http://www.apache.org: Página Oficial de Apache Server.*
- *http://www.doctrine-project.org: Página Oficial de Doctrine ORM.*
- *http://www.mysql.com: Página Oficial de MySQL.*
- *http://www.php.net: Página Oficial de PHP.*
- *http://www.propelorm.org: Página Oficial de Propel ORM.*
- *http://www.symfony-project.org: Página Oficial de Symfony Framework.*
- *http://www.uml.org: Página Oficial del Lenguaje de Modelado Unificado.*
- *http://www.viewlike.us: Página que permite visualizar una dirección de Internet con diferentes resoluciones de pantalla.*
- *http://www.w3schools.com: Página de referencia para consulta sobre tecnologías web.*
- *http://www.wikipedia.org: La Wikipedia.*
- *IEEE STD 830-1998 "Guide to Software Requeriments Specifications".*
- *Jaimie Sirovich, Cristian Darie "Professional Search Engine Optimization with PHP".*
- *Peter Lavin "Object-Oriented PHP Concepts, Technques and Code".*
- *Tim Converse, Joyce Park y Clark Morgan "PHP5 and MySQL Bible".*

Anexos

A. Detalles de instalación

El presente anexo no pretende ser una guía paso a paso de instalación. Este anexo sirve para clarificar el proceso de instalación y configuración de la aplicación desarrollada en este proyecto para la forma en que va a ser distribuida. También se ponen de manifiesto algunos de los problemas que pueden encontrarse o errores comunes que se pueden cometer durante este proceso.

Se asume que ya se dispone de un servidor con los requisitos software de una aplicación Symfony 1.2 instalados. Puede consultar estos requisitos en la sección 2.5 *Supuestos y dependencias*, del capítulo 2 *Especificación de requisitos* de este documento.

Para obtener información detallada sobre el proceso de instalación y todas las posibilidades que ofrece el framework Symfony, se puede consultar (http://www.symfony-project.org: Página Oficial de Symfony Framework).

1. Sandbox

Este proyecto se ha desarrollado a partir de una distribución *Sandbox* de Symfony, versión 1.2. La *Sandbox* es un proyecto Symfony pre-empaquetado y configurado por defecto en ciertos aspectos sensibles del framework, para realizar una instalación fácil y rápida. Pese a que no se recomienda el uso de una distribución *Sandbox* para proyectos reales (se utiliza más para un primer contacto y el aprendizaje temprano del framework), no existe problema alguno para crear versiones *release* a partir de este tipo de distribución. De hecho, gracias a que la *Sandbox* incorpora todas las librerías del framework bajo su estructura de directorios, basarse en este tipo de distribución permite generar un paquete de instalación auto-contenido de forma muy rápida. Por tanto, el proceso de instalación de la aplicación es muy parecido al proceso de instalación de una *Sandbox* del framework.

A grandes rasgos, la instalación de la aplicación consta de tres pasos que se describen a continuación:

- **Descomprimir el paquete "src.zip" en el directorio destino del servidor web.**

  Con esto se despliega el código fuente y la estructura de directorios de la aplicación (framework incluido) en su localización final.

- **Configurar el servidor web.**

Para evitar que archivos sensibles o privados de la aplicación sean visibles de forma pública, es necesario cambiar el *document root* por defecto de la aplicación a la carpeta *web*. La carpeta *web* contiene los únicos recursos públicos así como los controladores de la aplicación, el resto de carpetas pueden y deben ser inaccesibles desde la web. Si se tiene dudas de cómo realizar esta configuración puede consultarse la web (http://www.apache.org: Página Oficial de Apache Server).

- **Crear la base de datos y otorgar permisos.**

  En la carpeta *data* de la aplicación se proporciona el archivo llamado "dump.sql", que consiste en la definición de la base de datos que utilizaremos y los datos de inicio necesarios para el correcto funcionamiento de la solución. En la carpeta "config" existe un archivo llamado "databases.yml" que contiene la información relativa a la conexión con la base de datos, y en el que podrá consultar (o modificar) datos necesarios para crear la base de datos y el usuario con el que se accederá a la misma. Cree la base de datos y cargue los datos del archivo *dump*, luego cree el usuario y otorgue todos los permisos sobre la base de datos creada (no debe olvidarse utilizar la información del archivo "databases.yml" para este fin). En caso de dudas de cómo realizar este proceso, puede consultarse el manual de referencia de MySQL en (http://www.mysql.com: Página Oficial de MySQL).

Una vez realizados los pasos anteriores, será posible acceder correctamente a la aplicación mediante un navegador web. Puede comprobarse el funcionamiento de la aplicación accediendo a la página principal del portal utilizando la dirección del tipo *http://nombre-de-dominio/index.php* o simplemente *http://nombre-de.dominio*.

2. Estructura de directorios diferente

En ocasiones será difícil realizar una instalación estándar de la aplicación, entre otros motivos, porque no se tendrá permiso para administrar el espacio de alojamiento a voluntad o no se tendrá control sobre el servidor. Esto es habitual cuando se requiere de un servicio comercial de alojamiento en cualquier ISP del mercado. Los planes de alojamiento comerciales suelen tener restricciones, y muchos se basan en paneles de administración que siguen sus propias reglas. Una de las restricciones más comunes es no poder modificar la estructura de directorios pre-establecida. Éste es el caso, por ejemplo, de los alojamientos basados en el panel Plesk de Parallels, sin duda uno de los paneles de administración para servidores de alojamientos más conocidos.

Plesk define una estructura de directorios propia para controlar el espacio, entendido como zona virtual, de un alojamiento. En concreto, habitualmente los alojamientos en Plesk poseen dos carpetas llamadas *httpdocs* y *httpsdocs*, que se utilizan para albergar el contenido de un sitio web en función de si éste debe ser público (y por tanto accesible por todos) o protegido con SSL (sólo accesible con la aceptación de un certificado instalado en el servidor). Estos directorios son los directorios raíz para cualquier sitio web, uno para cada tipo de conexión HTTP o HTTPS.

En principio, podría pensarse que debido a esta restricción una aplicación Symfony no puede instalarse de forma adecuada en este tipo de servidores, ya que se

ha visto con anterioridad que el directorio público debía ser *web* y no *httpdocs* (o *httpsdocs* si queremos usar SSL), y el resto debían ser privados. Pero nada más lejos de la realidad.

Afortunadamente, Symfony permite configurar su estructura de directorios interna para poder adecuarlo a escenarios como el descrito anteriormente. Para ello se deben seguir los siguientes cambios:

1) Copiar el contenido de la carpeta *web* en la carpeta *raíz* del alojamiento, después crear una carpeta nueva bajo la carpeta "*raíz*" y copiar el resto de carpetas del proyecto en ella.

2) Modificar el archivo de configuración del proyecto llamado *ProjectConfiguration.class.php* que se encuentra en la carpeta *config*. Mediante los métodos *SetRootDir* y *SetwebDir* se indica al framework cuál es el directorio raíz del proyecto y cuál es el directorio web respectivamente.

3) Modificar los archivos correspondientes a los controladores frontales, indicando la ruta correcta donde se encuentra el archivo de configuración del proyecto.

4) Restringir el acceso a la carpeta que contiene al framework y el código de la aplicación mediante el uso de archivos *htacces*.

Para el caso comentado de Plesk, un ejemplo práctico siguiendo las directrices anteriores podría ser el siguiente:

1) Copiar el contenido de la carpeta *web* en el directorio *httpdocs*. Dentro de *httpdocs* creamos una carpeta llamada *private* y copiamos el resto de carpetas del proyecto dentro de *private*. La estructura de directorios resultante se observa en la siguiente figura.

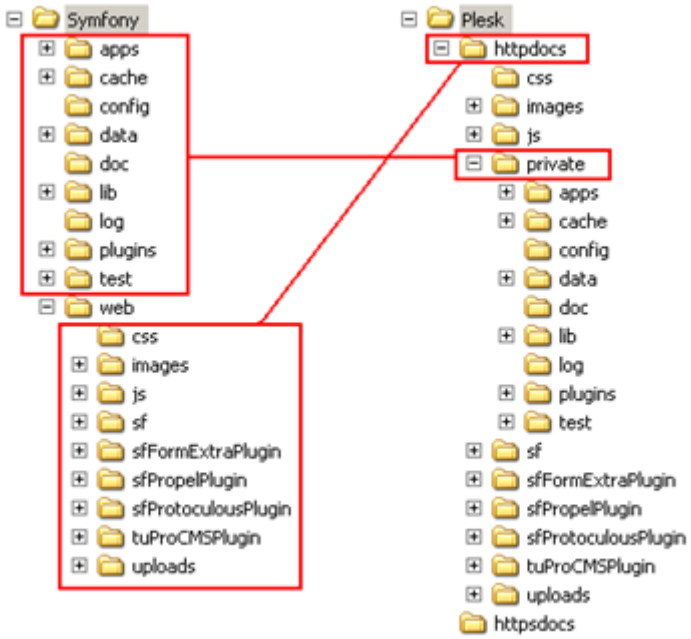


Ilustración 43: Cambio de carpetas Symfony en Plesk

2) Localizar y editar el archivo *ProjectConfiguration.class.php* en la carpeta *httpdocs/private/config*. Dentro del método de la clase *ProjectConfiguration* añadimos las siguientes dos líneas de código:

```
$this->SetRootDir(dirname(__FILE__).'/../');

$this->SetwebDir(dirname(__FILE__).'/../../');
```

Con esto se indica al framework que el directorio *web* debe ser *httpdocs* y que el directorio raíz del proyecto Symfony es *httpdocs/private*.

3) Localizar y editar el archivo "*index.php*" (o cualquier otro controlador de la aplicación) en el directorio "httpdocs". Modificar en la directiva "require_once" la ruta hacia el archivo "ProjectConfiguratio.class.php" editado anteriormente. Es decir, sustituir la primera línea por:

```
require_once(dirname(__FILE__).'/private/config/ProjectConfiguration.clas
s.php');
```

Con este cambio el framework está preparado para localizar todos los archivos que necesita.

4) Crear un archivo ".htacces" con una única línea (`deny all`), y colocarlo en el directorio "httpdocs/private". Con esto conseguimos que el contenido de esa carpeta no sea servido por el servidor HTTP (por tanto no accesible de forma pública).

Por lo general, no debería haber ningún problema para instalar la aplicación en cualquiera de los servidores del mercado, siempre y cuando se cumplan los requisitos tecnológicos necesarios. Como acaba de verse, el framework Symfony nos ofrece grandes facilidades de configuración y versatilidad.

Para obtener más información sobre la estructura de directorios de Symfony y detalles de funcionamiento puede consultarse la documentación disponible en *http://www.symfony-project.org/doc/1_2/*. Para consultas sobre la utilización de archivos ".htacces" visítese la página oficial de Apache en *http://www.apache.org*.

3. Errores y problemas comunes

A continuación se describen algunos de los errores cometidos comúnmente y problemas típicos que, de no tenerse en cuenta, pueden frustrar cualquier instalación de la aplicación. Se presenta en forma de tabla, diferenciando el problema que se obtiene, la posible causa y la solución que debe adoptarse.

Problema	Posible Causa	Solución
En general cualquier tipo de error: suelen darse errores PHP, errores de página no encontrada, errores internos del	No se cumple alguno de los requisitos mínimos.	Aunque parece una obviedad, hay que asegurarse de que el servidor llega a los requisitos mínimos

servidor o de acceso a base de datos.		exigidos. Actualizar el servidor con el software adecuado o migrar la aplicación a un servidor actualizado.
Al acceder a la aplicación se obtiene como respuesta una página en blanco.	Se ha alcanzado el máximo de memoria que un script PHP puede consumir.	Aumentar este límite en el archivo php.ini de la instalación de PHP hasta obtener un resultado. Este valor corresponde a la directiva `memory_limit`.
Errores de página no encontrada cuando se accede a las rutas de la aplicación.	El archivo .htacces no está siendo interpretado por el servidor web.	Asegurarse que en el archivo de configuración de Apache la directiva `httpdconf allow override` está a `on`.
Los cambios de configuración sobre PHP en el php.ini no tienen efecto.	No se está editando el archivo php.ini correcto.	Es habitual que las instalaciones de PHP posean dos tipos diferentes de configuración, una para la línea de comandos en línea y otra para los scripts procesados por el servidor web. Hay que asegurarse de estar editando el archivo para la configuración del servidor web. Podemos encontrar fácilmente su ubicación exacta utilizando la función `php_info()` en un script y buscarlo en el resultado obtenido.
Errores PHP al intentar acceder o consultar la base de datos.	Las extensiones PDO de PHP no se encuentran instaladas o activadas.	Editar el archivo php.ini y activar las extensiones php_pdo y php_pdo_mysql (deben estar instaladas,
	Los datos de conexión a la base de datos son incorrectos.	Validar y en caso necesario modificar los datos de conexión a la base de datos en el archivo databases.yml de la carpeta conf de Symfony.
Se obtiene una página en blanco o el servidor web	Posible problema con los permisos de la carpeta	Borrar el contenido de la carpeta "cache" de la

devuelve un error 500.	"cache".	estructura de directorios de Symfony (se regenera automáticamente con el siguiente acceso a la aplicación).

B. Herramientas utilizadas

1. Eclipse PDT

Eclipse PDT es un entorno de desarrollo preparado específicamente para el desarrollo de aplicaciones con PHP. Está formado por el entorno Eclipse más el plugin de herramientas para PHP o PDT (del inglés PHP Developmet Tools).

Ilustración 44: Logotipo Eclipse

Eclipse es un entorno de desarrollo integrado de código abierto multiplataforma. Típicamente ha sido usado para desarrollar entornos de desarrollo integrados (del inglés IDE), como el IDE de Java llamado Java Development Toolkit (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse).

Eclipse fue desarrollado originalmente por IBM aunque ahora es desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios.

PDT extiende las capacidades de Eclipse para proporcionar facilidades en el desarrollo de aplicaciones web mediante PHP: inspector de código, visualización de funciones, coloreado y formateo de código, ayuda a la terminación de código, debugging, etc.

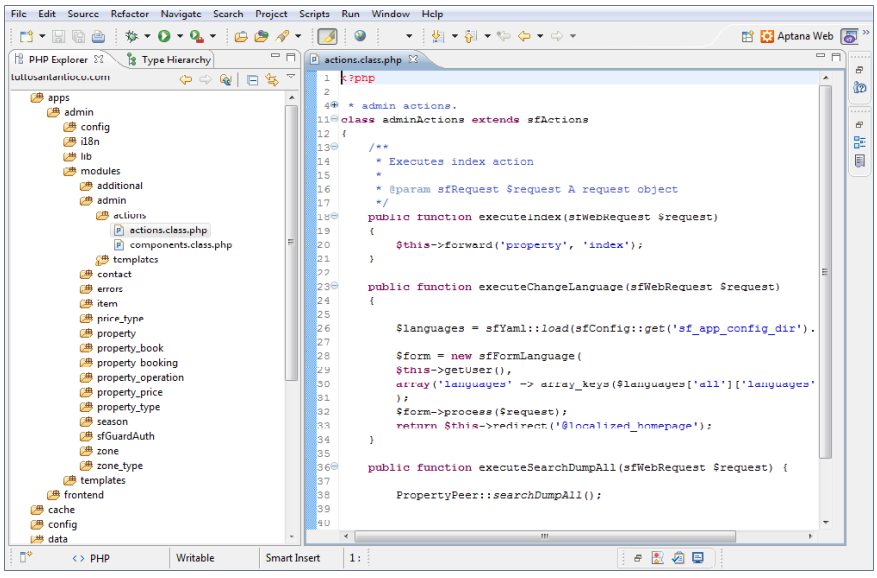


Ilustración 45: Apariencia de Eclipse

2. WampServer

WampServer es un entorno de desarrollo web para Windows. Permite crear aplicaciones web con Apache, Php y MySQL, ya que empaqueta todas estas herramientas en un único instalador, permitiendo su uso y configuración de una forma rápida y sencilla. WAMP viene de las siglas de la herramientas que conforman dicho entorno: Windows + Apache + MySQL + PHP.

Ilustración 46: Logotipo de WampServer

WampServer incluye también otro tipo de herramientas secundarias como phpMyAdmin para la gestión de las bases de datos, o XDebug (una extensión de PHP para potenciar la depuración del código fuente).

Cabe destacar la posibilidad que ofrece WampServer para añadir tantas versiones finales como se desee, simplemente descargándolas de su sitio web. Esto permite la creación de múltiples entornos, con diferentes versiones de las herramientas incluidas, cada una con sus propios archivos de configuración de manera diferenciada.

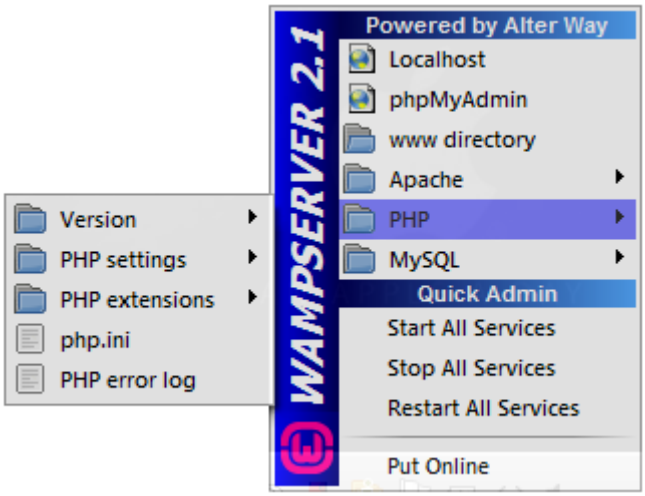


Ilustración 47: Menú principal de WampServer

Desde su menú principal, disponible mediante un icono en el área de notificaciones de la barra de Windows, se tiene acceso directo a los archivos de configuración de las herramientas así como diferentes opciones que facilitan el uso del entorno. Concretamente, desde el menú será posible:

- Gestionar los servicios de Apache y MySQL.
- Poner en marcha o apagar el entorno, así como dar acceso a todos o sólo a localhost.

- Instalar y cambiar diferentes versiones de Apache, MySQL y PHP.
- Gestionar la configuración de las herramientas del entorno.
- Acceder a los archivos de los logs.
- Acceder a los archivo de configuración.
- Crear alias.

3. phpMyAdmin

phpMyAdmin es una herramienta de software libre escrita en PHP para el manejo de la administración de MySQL a través de la web. phpMyAdmin soporta un amplio rango de operaciones con MySQL. Las operaciones usadas con más frecuencia están soportadas por la interfaz de usuario (gestión de bases de datos, tablas, campos, relaciones, índices, usuarios, permisos, etc.) mientras que para otro tipo de operaciones está disponible la posibilidad de ejecutar de forma directa cualquier sentencia SQL.

Ilustración 48: Logotipo de la herramienta phpMyAdmin

phpMyAdmin dispone de gran cantidad de documentación, y para permitir su uso por el mayor número de usuarios se ha traducido a 62 lenguajes distintos.

phpMyAdmin ha ganado numerosos premios. Entre otros, fue elegida como la mejor aplicación PHP en varios premios y ha ganado cada año los SourceForge.net Community Choice Awards como "La mejor herramienta o utilidad para Administradores de Sistemas".

Algunas de sus características son:

- Interfaz web intuitiva
- Soporte para la mayoría de operaciones de MySQL
- Importación de datos desde CSV o SQL.
- Exportación de datos a varios formatos: CSV, SQL, XML, PDF y otros.
- Administración de multiples servidores.
- Y mucho más...

4. StarUML

StarUML es un proyecto de código abierto para el desarrollo de una plataforma UML/MDA de forma rápida, flexible, extensible, y de libre acceso que se ejecuta en la plataforma Windows. El objetivo del proyecto StarUML es construir una herramienta de modelado de software que sirva de plataforma convincente para el remplazo de herramientas UML comerciales como Rational Rose o Together.

Ilustración 49: Logotipo de StarUML

Entre muchas de las características que ofrece esta herramienta, la que más interesa para la realización de este proyecto es su soporte para UML 2.0, poniendo a disposición del desarrollador la capacidad de creación de los principales diagramas de este lenguaje de modelado.

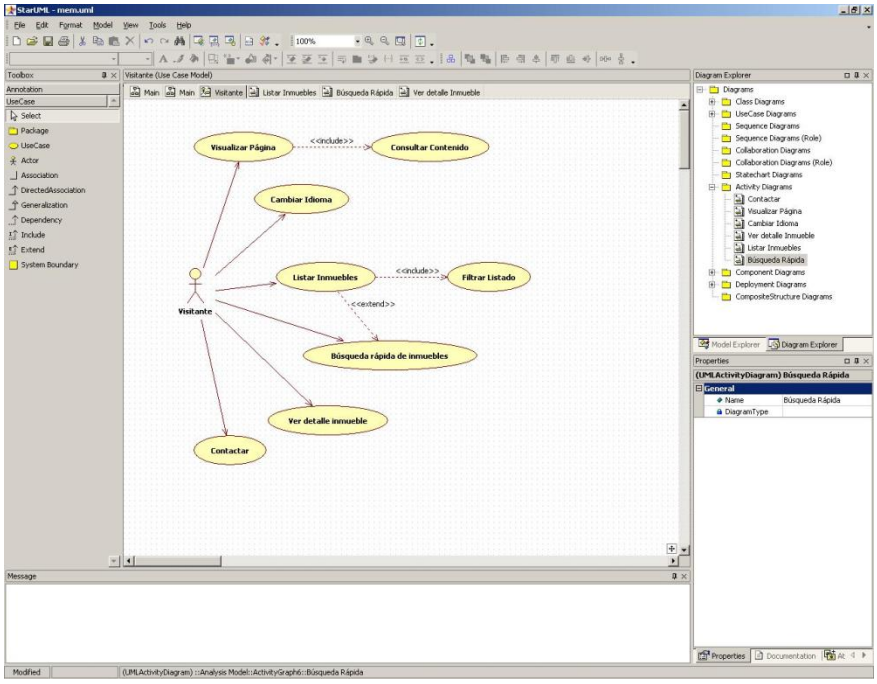


Ilustración 50: Apariencia de StarUML

5. Pencil

Pencil es una herramienta de fácil manejo y código abierto para crear diagramas y prototipos de interfaces gráficas de usuario. Esta herramienta se encuentra disponible como add-on del Navegador web Firefox, aunque existen versiones independientes.

Ilustración 51: Logotipo de la herramienta Pencil

Pencil permite interrelacionar los diagramas o las pantallas de interfaz creadas, por lo que es posible crear interfaces gráficas navegables (en cierta manera "ejecutables"). Esta característica puede ser de gran ayuda para generar un prototipo rápido desechable con el que detectar problemas o captar mejor los requisitos de una aplicación.

Algunas de las características más importantes son:

- Plantillas incorporadas para la generación de diagramas y prototipos.
- Documentos multi-página.
- "Linking" entre páginas.
- Edición de texto en pantalla con soporte para texto enriquecido.
- Exportación a HTML, PNG, Openoffice, Word y PDF.
- Plantillas definidas por el usuario.
- Operaciones de dibujo estándar: alineación, z-ordenación, escalado, rotación...
- Adición de objetos externos.
- Colecciones personales.
- Ajuste de objetos.

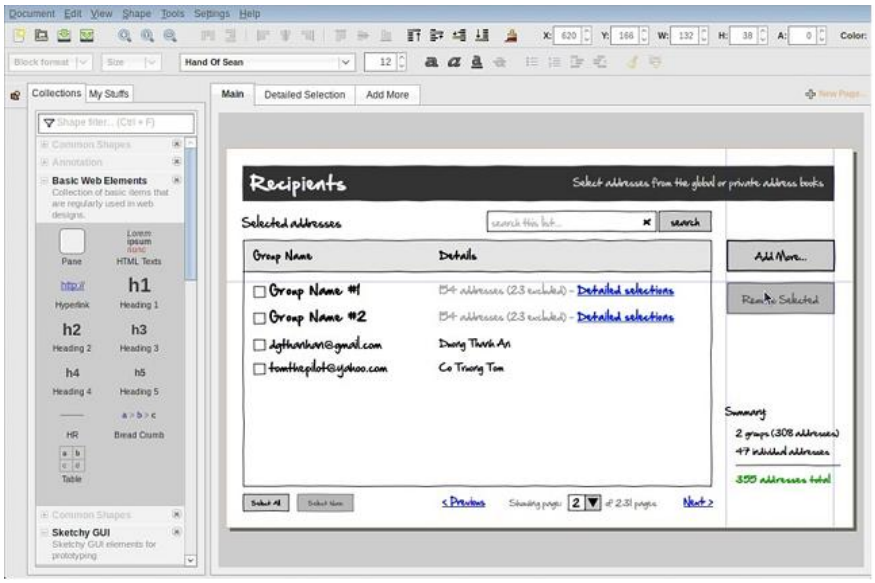


Ilustración 52: Apariencia de la herramienta Pencil