ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación :

INGENIERO TÉCNICO DE TELECOMUNICACIÓN, ESPECIALIDAD EN SONIDO E IMAGEN

Título del proyecto:

MIGRACIÓN DE INMOBILIARIA JOOMLA A SYMFONY2

César Zubillaga Beraza

Javier Navallas Irujo

Pamplona, 09/09/2016

Indice

Introducción

El trabajo realizado en este Proyecto Fin de Carrera consiste en analizar una página multilingüe para una inmobiliaria, ver los fallos existentes, analizar las posibles mejoras y presentar una propuesta de actualización de la página.

Esta página fue desarrollada con el gestor de contenidos Joomla en la versión 1.6. A fecha de comienzo de las reuniones la versión era la 3.0, por lo que se partía de una versión bastante antigua. Con anterioridad a este proyecto, nunca había trabajado con este CMS y nunca había aceptado un trabajo en el que tuviera que desarrollar o mantener un sitio web de estas características.

En la primera reunión que mantuve con el cliente le deje totalmente claro estos puntos. De realizar yo el trabajo crearía un gestor de contenido propio basándome en la estructura y base de datos existente, aplicando mejoras que yo creía convenientes siempre y cuando al cliente le parecieran correctas. Esto supondría quizás un coste mayor que contratar a un especialista en Joomla que corrigiera los errores, pero el insistió en que quería trabajar conmigo.

Una vez llegados a un acuerdo monetario en base a las funcionalidades que tendría la web, siempre con una estimación de las horas invertidas, comenzó la fase de desarrollo del proyecto.

El primer paso fue contactar con la empresa que desarrolló la página existente, pero ésta estaba desaparecida. Al no existir manera de acceder al hosting, no hay acceso a los archivos ni a la base de datos. Esto fue el primer impedimento no previsto para realizar el trabajo y el primer error cometido.

Todo imprevisto supone un cambio en el proyecto y esto hay que notificarlo al cliente. La comunicación de que el contexto encontrado podía suponer un incremento en el precio o una disminución de la calidad del mismo (eliminar funcionalidades, diseño…) no fue muy bien recibido, pero tras la reunión se me otorgó libertad para hacer lo que creyera conveniente.

La primera idea que tuve para solucionar el problema de la base de datos fue rastrear la web para conseguir un duplicado de la información necesaria. La alternativa a ello era ir uno por uno por todos los inmuebles e ir copiando y pegando datos, descargando imágenes, todo para euskera y castellano. Esto último no era una opción, me resultaba mucho más productivo hacerlo mediante un rastreo, además que aprendería nuevas maneras de solucionar este tipo de problemas.

Antes de ponerme con este punto tuve que diseñar la base de datos, y para ello empleé como referencia la página web. De aquí extraje el diseño de lo que podría ser su estructura y planteé algunos cambios que mejorarían su posterior ordenación y filtrado.

Una vez diseñada y completa la base de datos era hora de pasar a escribir el código para mostrar ese contenido. Pocos meses atrás había estado en un congreso de Symfony2 en Madrid. De todas las charlas que estuve retuve bastante poco ya que mi nivel en este framework era casi nulo, pero hubo algunos ejemplos de funcionamiento de bundles o módulos que me llamaron mucho la atención, lo que atrajo mi curiosidad y decidí comenzar a trabajar en este entorno.

Antes de hacer este proyecto había hecho algunas pruebas con páginas en modo local y una de ellas la puse en producción. Proyectos pequeños en los cuales mi inversión temporal fue enorme en comparación con el precio de venta, pero esta experiencia me sirvió para ver las posibilidades de este framework, lo que hizo que tomara la decisión (que considero acertada) de hacerlo en este entorno.

El segundo imprevisto fue detectar que en la página web se habían colado enlaces de páginas pornográficas que no se había detectado hasta que cayó en mis manos. Eran muy difíciles de ver ya que había dos puntos en el trabajo: el primero era la inserción en la base de datos de los enlaces porno, el segundo un javascript que ocultaba su contenido en la parte front al cargarse la web, pero eso no evitaba que google indexara esos enlaces (lo que supongo aumentaría su ranking SEO).

El cliente para aquel entonces ya se estaba poniendo nervioso, dado que no podía soportar que estuvieran esos enlaces porno en su página web. Tras varias llamadas para presionar para la resolución del problema le expliqué qué era lo que íbamos a hacer y cuánto tiempo nos costaría.

Lo primero era poner la página web en obras ya que teníamos la base de datos poblada y todas las fotografías en mi ordenador. Para hacer esto se hizo un cambio de los DNS que apuntarán a mi servidor virtual, donde tengo otros proyectos alojados.

El siguiente paso era desarrollar el front-end, así los clientes podrían ver sus viviendas en la web.

El último paso del trabajo era el back-end para que pudieran insertar, editar y eliminar ofertas, lo que completaría la funcionalidad de la página y, por tanto, pondría punto final a la fase de ejecución del proyecto.

En este documento he intentado contar los pasos más importantes que seguí para hacer el trabajo en orden cronológico tal y como los he descrito esta introducción.

Entorno y herramientas de trabajo

En el siguiente apartado se explica brevemente cuales son las características del entorno de desarrollo y producción, las herramientas utilizadas, qué función tienen y en qué parte del proyecto han sido utilizadas.

Desarrollo

- Windows 7 64 bits , 4.00GB Ram. Inter Core i7-3610QM CPU a 2.30GHz.
- WAMP (Windows, Apache 2.4.2, MySQL 5.5.2.4 (InnoDB) y PHP 5.4.3)

Producción

- CentOS 6.4
- LAMP (Linux, Apache 2.2.15, MySQL 5.1.63 (InnoDB) y PHP 5.5.0)

IDE Netbeans

Una herramienta perfecta para escribir código PHP, CSS, JS, HTML, entre otros lenguajes. Además puede importar proyectos de tipo Symfony2. De esta manera, cuando llamas a un componente o estas dentro de una clase, el sistema ya conoce los métodos que tiene y cuáles son sus entradas.

Symfony2

Se trata un framework PHP basado en el patrón modelo vista controlador (MVC) con una documentación y una comunidad muy extensa. Además existe un gran número de módulos o bundles desarrollados de forma interna por SensioLabs o por terceros.

Bundles

Los módulos facilitan el trabajo en el desarrollo de una aplicación ya que realizan funciones cotidianas en cualquier website y están mantenidos por la comunidad. Estos son los bundles que se utilizan en la aplicación.

- Grewar image bundle https://github.com/Gregwar/ImageBundle es una librería que se encarga de reescalar, recortar, poner marcas de agua y comprimir imágenes de manera automática. Su integración es muy sencilla con el entorno Symfony. Existen otras funcionalidades que no se han utilizado dentro del proyecto.
- Ckeditor Bundle https://github.com/egeloen/IvoryCKEditorBundle es un editor de texto html muy utilizado por los desarrolladores de CMS (como por ejemplo wordpress), para dar una interfaz para usuarios con pocos conocimientos en HTML.
- Knp Paginator Bundle https://github.com/KnpLabs/KnpPaginatorBundle se encarga de paginar resultados obtenido de base de datos, array s. Está preparado para recibir una consulta a base de datos y mediante parámetros limitar los resultados devueltos, lo que hace que sea mucho más eficiente.
- I18n routing Bundle https://packagist.org/packages/jms/i18n-routing-bundle un extra al componente que viene en el framework de Symfony muy útil cuando se trata de websties multilingües. La configuración permite denominar para una misma ruta interna de Symfony varias URL y que todas las peticiones acaben en un mismo controlador.

Bootstrap (html5, css3 y Javascript + JQuery)

Bootstrap3 es un framework front-end que facilita el desarrollo visual, además de dar algunas funcionalidad básicas de cualquier website, como por ejemplo el carrousel o el adaptativo. Se ha utilizado tanto en el panel de control como en la parte visualizada por el usuario.

http://getbootstrap.com

MySQL Workbrench

En este proyecto se ha utilizado este programa para crear la base de datos, con sus tablas y relaciones.

https://www.mysql.com/products/workbench/

IOJS & NODE.JS

Utilizando el motor javascript v8 de Chrome y las especificaciones de ECMA Script crea un entorno que se utiliza entre otras cosas para poder crear servidores locales que funcionan con un lenguaje javascript. Se ha utilizado para capturar los datos y fotografías que existían en la página web.

http://iojs.org/

http://nodejs.org/

NPM

Es un gestor de módulos o paquetes utilizado en este caso dentro de node e iojs. En este trabajo se han utilizado:

- **http** propio del core node.js gestiona la creación de un servidor que pone, escucha y responde a peticiones http.
- **Mysql** te conecta con una base de datos https://www.npmjs.com/package/mysql .
- **jsdom** es la manera de poder utilizar selectores javascript y jquery dentro node.js e iojs al capturar una página https://www.npmjs.com/package/jsdom
- **assert** módulo de aseveración https://www.npmjs.com/package/assert .

http://npmjs.org/

COMPOSER

Gestor de dependencias PHP para instalar Symonfy2 y bundles. Una herramienta imprescindible en cualquier proyecto que requiera de alguna librería. En un archivo JSON se configuran las necesidades del proyecto y dentro de la línea de comandos se instalan las librerías. Utilizado tanto en producción como en desarrollo.

https://getcomposer.org/

Análisis

Los primeros pasos de un proyecto web de este tipo son los de analizar la situación de la página web, tanto en el front-end como en el back-end. Ver cómo es su estructura, contenido y funcionamiento, un trabajo que ya se ha realizado antes y se puede mantener o mejorar.

Una vez analizada se presentan los problemas y las mejoras que el cliente puede necesitar, se debaten y se toma la decisión de en cuales actuar, todo bajo presupuesto, aunque esta parte no se detallará.

La web

Se trata de una página desarrollada en Joomla 1.6 con un módulo específico para este tipo de proyectos, en este caso una inmobiliaria.

Tiene su parte pública o front-end, que es donde los usuarios pueden navegar buscando inmuebles en diferentes provincias y localidades, con un tipo de vivienda u otro (chalet, adosado, etc.).

El back-end funciona como un gestor de contenido donde suben fotografías y viviendas, crean nuevos pueblos y provincias, y destacan unas ofertas de otras activándolas para que aparezcan en la sección de destacados.

Front-end

La **página principal** (home) es donde se muestra un banner y unas ofertas destacadas, tanto inmuebles como promociones.

Desde la página principal se accede a las secciones de **promociones** e **inmuebles**, donde se listan las ofertas acompañados de unas fotografías y un **formulario** de búsqueda. El formulario acepta la entrada de los siguientes datos: provincia, localidad, tipo de vivienda. A estas secciones les acompaña una columna donde se muestran las ofertas destacadas.

Para cada oferta existe una página descriptiva del **inmueble** y de la **promoción**, donde hay un contador de visitas, la posibilidad de descargar la información en PDF y de ver la dirección en la que está. Esta sección tiene una galería de fotografías del inmueble que se activa al cliquear en los thumbnail.

También desde la **página principal se puede acceder a: Donde estamos** es un mapa de google donde aparece la localización del negocio;**Contacto** es el típico formulario de contacto; y**Aviso legal** es otra sección de contenido estático.

Esta web tiene la característica de ser bilingüe, con su contenido en **castellano** y **euskera**.

Problemática

Lo más grave que se muestra con la primera reunión es que no existe acceso a la base de datos, ya que la empresa que desarrollo el website desapareció y absolutamente toda la web está en un hosting al cual no se tiene acceso. Lo único que está a nombre de la empresa es el dominio, el cual se puede transferir a otra empresa de registro de dominios y así poder apuntar los DNS al hosting que se desee.

El hecho de que la base de datos de la web anterior sea inaccesible plantea bastantes complicaciones en el planteamiento inicial del proyecto. Al haber casi 70 inmuebles y estimando como número promedio de fotografías por inmueble 4, existen 280 fotografías que habría que descargar, renombrar y actualizar en la base de datos. Eso solo con las fotografías,

pero de forma adicional habría que extraer la información asociada a la descripción y el titulo (en castellano y euskera), además de otras características relacionadas con las características de la vivienda. Esto planteaba un trabajo demasiado repetitivo y tedioso como para hacerlo manualmente. En la siguiente tabla muestro una aproximación del trabajo a realizar.

	Img (4)	**Título (2 lang)**	**Desc. (2 lang)**	**16 caract.**
Total (70 ofer.)	280	140	140	1120

Aún quedaba un problema adicional que para la empresa contratante es realmente grave. Debido a algún tipo de ataque se pudo colar publicidad (enlaces) de páginas pornográficas en las descripciones de los inmuebles y títulos. Esto les tenía totalmente preocupados ya que daba una imagen no deseada de su empresa y su resolución se consideraba un tema prioritario en el proyecto.

Propuesta

En este apartado se recogen las características que tendrá la página web una vez esté funcional, tanto la parte pública como el panel de control o gestor de contenido. Además se proponen unas fases de desarrollo de trabajo, con unos hitos y así poder ir avanzando y no esperar hasta la finalización del trabajo para ponerla en producción.

Front end

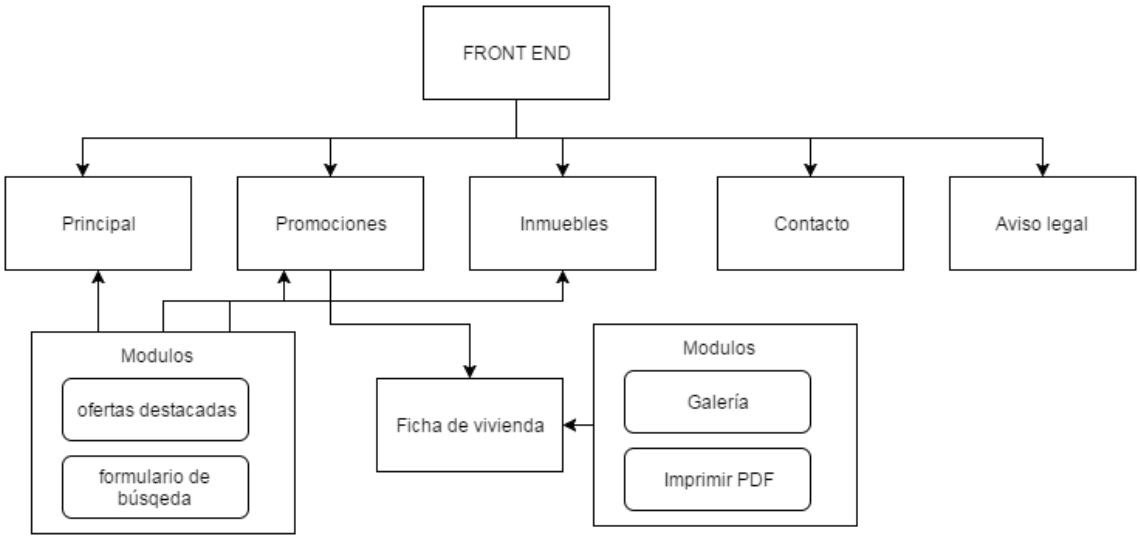

Fig. 1Estructura de la página web antigua

Por un lado eliminar las secciones de promociones e inmuebles y agruparlas en un solo listado, así las ofertas podrán ser de vivienda de segunda mano o nuevas. Dentro de esta seccións propone:

1. **Buscador** para encontrar las ofertas por tipo de transacción, tipo de propiedad y provincia, donde:
    a. **Transacción:** venta o alquiler.
    b. **Tipo de propiedad:** Inmueble o promoción.
2. **Filtrar** los resultados por precio, número de habitaciones, garaje, ascensor y calefacción.
3. **Ordenar** las viviendas por precio ascendente o descendente
4. **Buscador** por referencia.

Sección en la que se muestra el inmueble con las características y todas las fotografías que tiene esa vivienda, eliminando la galería y mostrando las fotografías haciendo scroll a tamaño completo. Las funcionalidades que tenía la web anterior de imprimir PDF y contador de visitas se eliminan. La dirección también desaparece, ya que desde la empresa nunca se rellenaba ese campo.

La sección de **destacados** se mantiene en la página principal pero se elimina del resto de la web.

Contacto cambia el formulario dejando solo nombre, email y comentario. **Donde estamos** se mantiene. **Aviso legal** pasa a ser un contenido estático en todas las páginas, pero se muestra utilizando un efecto modal de Bootstrap.

El **cambio de idioma** tiene que recordar donde está el usuario en ese momento y no redirigirle a la página de inicio.

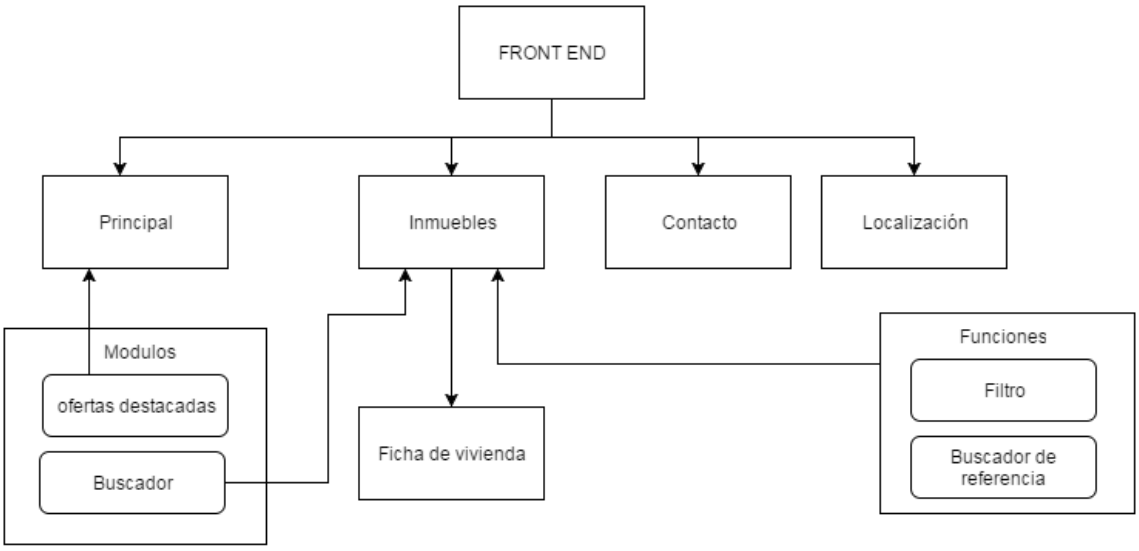

Fig 2 Estructura de la web nueva

Back end

Sin estudiar el gestor actual de Joomla se propone una administración con 4 entradas:

Ofertas

Donde se listen todas y exista la posibilidad de buscarlas por referencia. El proceso de creación en dos pasos. El primero sería configurar los campos no traducibles, referencia, superficie, habitaciones, baños, trastero, garaje, ascensor, terraza, calefacción, precio, precio final, la localidad, el tipo de propiedad, transacción y vivienda. Al ser validados todos estos campos se pone una imagen, descripciones y títulos por defecto, que serán editados en el segundo paso.

Editar las ofertas tendrán 4 sub secciones: oferta o campos no traducibles, campos traducibles (título y descripción), uno para el euskera y otro para el castellano, fotografías asociadas a la ofertas, donde poder clicar y así editarla, y eliminar la oferta.

Fotos

Debe permitir subir fotografías y asociarlas a una referencia mediante select. Debe presentar un listado de todas las fotografías, buscador por referencia de oferta, y su correspondiente edición. Dentro de la edición debe existir la posibilidad de eliminar la fotografía, y en caso de

ser la principal poner cualquier otra asociada a esa oferta como principal. De no haber más fotografías se cargará una por defecto.

Localidades

Presentará el listado de todas las localidades que se vayan creando, posibilitando su edición tanto en euskera como en castellano dentro del mismo formulario, y su asociación a una provincia dentro de la CAV y C.F de Navarra, que son las provincias que se ha decidido cargar.

Destacados

Presentará el listado de todas las ofertas y su referencia, y un checkbox a su lado donde poder marcar cuales se quiere que estén activas. Por otro lado la posibilidad de elegir en qué posición se desea situar.

Diseño

Utilizando el framework Bootstrap se pretende rediseñar la página que ya tiene 5 años para actualizar su distribución y estética. Este framework facilita unas herramientas para diseñar la página teniendo en cuenta los diferentes tamaños de pantalla, así generamos una vista adaptada a las diferentes resoluciones de dispositivos.. Esta parte del trabajo es menos prioritaria para la empresa contratante, por lo que se sitúa a la cola de la pila de trabajos.

Fases

La primera fase planteada es diseñar la nueva base de datos e importar la información de la antigua, recogiendo todo lo que hay en la página, tanto fotografías como descripciones. De esta manera se tiene una copia de seguridad del sitio.

Una vez guardada toda la información, el dominio se transfiere a otra empresa y desde ahí se cambian los DNS apuntando a un servidor donde se pondrá un mensaje informativo de que la página está en obras.

La siguiente fase se ocupará de limpiar el contenido indeseado y mostrarlo utilizando Symfony2 con la consiguiente reestructuración de la página web y las nuevas funcionalidades del front-end. Una vez completada esta fase en el entorno de desarrollo se lleva al de producción.

Finalmente se deberá crear un back-end hecho a medida para este caso, mostrárselo al cliente y subirlo al entorno de producción.

Base de datos

El primer punto es analizar la página web, su estructura, enrutamiento y las características de cada vivienda. Es de aquí de donde se va a desprender la información que construye la base de datos, todo siempre visto desde un punto de vista multilingüe.

El motor de la base de datos escogido será **InnoD**. Un motor de MySQL que permite las transacciones, realizando un número de operaciones encadenadas sin persistencia hasta que todas ellas se hagan sin ningún tipo de error, haciendo que la base de datos sea consistente. Las claves foráneas son otra característica de este motor, pudiendo relacionar tablas entre sí y estableciendo una lógica que la hace más consistente. Posteriormente y utilizando MySQL Workbench se trabaja para crear las tablas, las columnas con el tipo de datos y sus relaciones.

A la final es cuando se utiliza ingeniería inversa para crear la base de datos en el servidor local.

Ofertas

Se trata de la colección de viviendas que están ofertadas en la página web. Antes de enumerar las características que tiene una vivienda se ha de segregar las que son traducibles y las que no. En la Fig. 1 están representadas las características básicas de una oferta.

No traducibles: calefacción, terraza, ascensor, garaje, trastero, baños, habitaciones, superficie, fotografías, precio y precio final.

Traducibles: tipo de transacción, tipo de vivienda, localidad y provincia en la que está situada, tipo de propiedad, ruta, descripción y título.

De las traducibles hay unas que son únicas de la vivienda como es el **título**, la **ruta** y la **descripción**, pero hay otras características que se repiten, como el **tipo de vivienda**, el tipo de **transacción,** el **tipo de propiedad**, la **localidad** y la **provincia** por otro lado. Estas características ennumeradas estarán recogidas en sendas tablas a parte.

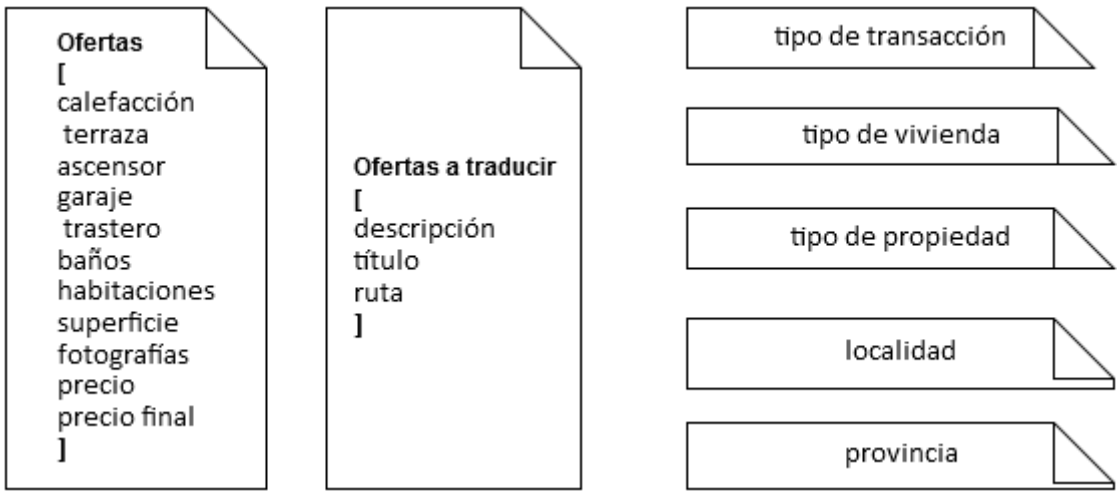

Fig. 3 Características de una Oferta

Tipo de viviendas

Se trata de una característica de la vivienda, ya que puede ser un chalet, un adosado, un piso…

Tipo de propiedad

Una vivienda puede ser de obra nueva (promociones) o de segunda mano. Se decide variar el nombre de segunda mano por vivienda.

Tipo de transacción

La vivienda puede cambiar de propietario/habitante de tres maneras posibles: por venta, o alquiler.

Traducciones y rutas

Todas las tablas que deben de ser traducidas tienen una tabla espejo donde se almacena la traducción. El listado de tablas a traducir son:

- Provincias
- Tipo de propiedad
- Tipo de transacción
- Tipo de vivienda
- Ofertas

Para saber en qué idioma se almacena se utiliza un identificador del lenguaje acompañado a la traducción, el cual está relacionado con una tabla de idiomas activos. Actualmente existen el euskera y el castellano, con sus iso_code y su formato de fechas.

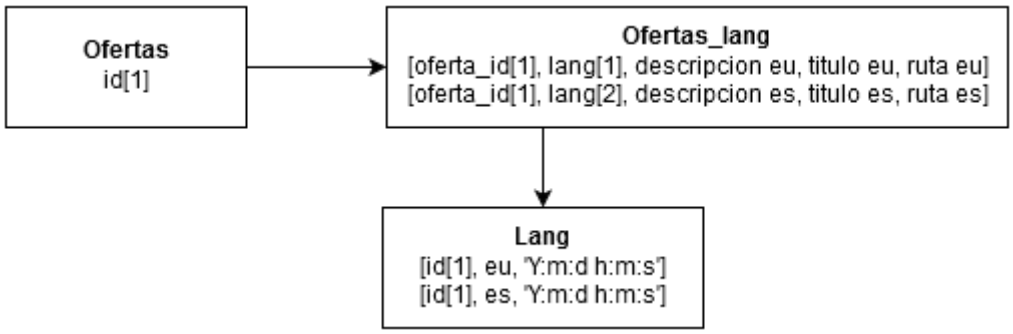

Fig 4 Ejemplo de traducción de características

Existe un caso particular en el que se crea una tabla sólo con un campo, el identificador. Se trata de casos en los que todas las propiedades de esta tabla son traducibles, existiendo en una tabla relacionada con sus traducciones. Por lo que no hay un dato compartido para todas los idiomas. Este tipo de casos de traducciones ocurren con el tipo de transacción, tipo de propiedad y tipo de viviendas.

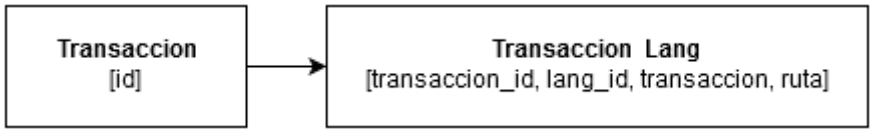

Fig 5 Ejemplo de caso particular en tipo de transacciones

El campo route, como bien explica su nombre, sirve para crear las rutas que generan la estructura de la web.

- En el caso de la tabla ofertas_lang sirve para ver la oferta individualmente.

- La columna route de las tablas transacciones, tipo de propiedad y provincia son utilizadas para crear el formulario de filtrado que genera una ruta donde se mostrarán los resultados.

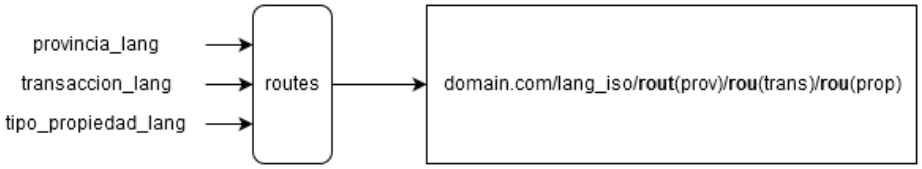

Fig 6 Parámetros de construcción de la ruta para búsqueda de inmuebles

Estas funcionalidades se explicarán con más detalle en la sección de Controladores.

Localización de la oferta

Una oferta tiene que pertenecer a un pueblo, y este pueblo ha de pertenecer a una provincia.

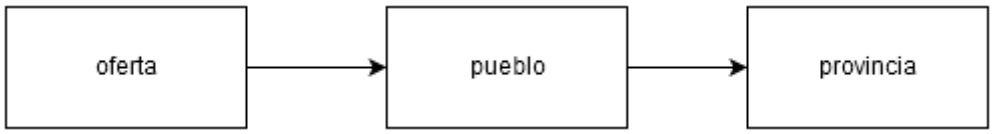

Fig 7 Relación entre oferta, pueblo y provincia

En un caso de mayor envergadura se debatiría poder cargar en la base de datos todas las provincias de España, municipios, códigos postales. Al ser una inmobiliaria con un público muy focalizado en una zona poco extensa se decide afrontar el trabajo de la siguiente manera:

Provincias

Actualmente trabajan en la provincia de Gipuzkoa, pero en el pasado había ventas en otras provincias. Por proximidad se cargan las provincias de la CAV y la CFN.

Localidad

La ciudad o pueblo donde está situada la vivienda. Esta localidad está relacionada con la provincia a la que pertenece.

Dentro de la oferta se ingresa el identificador de la localidad, el cual nos lleva hasta la provincia a la que pertenece.

Fotografías de las ofertas

En el listado de viviendas de la página actual aparece la oferta acompañada de una fotografía. Dentro del detalle de la oferta aparece un número de imágenes más. Por lo tanto se desprende que hay que marcar una de ellas como principal y que las demás quedarán como secundarias.

.

Ofertas destacadas

En la página principal se quieren destacar unas ofertas, por lo que se crea una tabla llamada destacados que contendrá los identificadores de las ofertas que se quieran destacar y la posición en la que se quieran destacar.

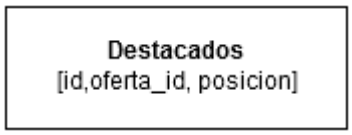

Fig 8 Tabla de destacados

Claves foráneas o relaciones entre tablas

Utilizando el motor InnoDB se relacionarán las tablas entre sí para que luego al generar las entidades en Symfony2 exista una relación entre las mismas. El método que se ha elegido para nombrar esas claves foráneas y evitar duplicidades es el siguiente:

Nombre de la columna FK + Nombre de la tabla que quiero relacionar. Ejemplo con *transacciones_lang* quedaría de la siguiente manera: *lang_id_transacciones_lang*.

El resultado de aplicar este método para nombrar las claves foranes se puede apreciar haciendo una select dentro de la base de datos en las tablas de la base de datos INFROMATION SCHEMA.

```
SELECT `CONSTRAINT_NAME` , `COLUMN_NAME` , `TABLE_NAME` ,
`REFERENCED_TABLE_NAME` , `REFERENCED_COLUMN_NAME`
FROM `KEY_COLUMN_USAGE`
WHERE TABLE_SCHEMA = 'App_bd'
AND CONSTRAINT_NAME != 'PRIMARY'
ORDER BY `KEY_COLUMN_USAGE`.`REFERENCED_TABLE_NAME` ASC
```

CONSTRAINT_NAME	COLUMN_NAME	TABLE_NAME	REFERENCED_TABLE_NAME	REFERENCED_COLUMN_NAME
lang_id_provincias_lang	lang_id	provincias_lang	lang	id
lang_id_tipos_viviendas_lang	lang_id	tipos_viviendas_lang	lang	id
lang_id_ofertas_lang	lang_id	ofertas_lang	lang	id
lang_id_tipo_propiedad_lang	lang_id	tipo_propiedad_lang	lang	id
lang_id_transacciones_lang	lang_id	transacciones_lang	lang	id
ofertas_id_destacados	ofertas_id	destacados	ofertas	id
ofertas_id	ofertas_id	fotos	ofertas	id
ofertas_id_ofertas_lang	ofertas_id	ofertas_lang	ofertas	id
provincias_id_provincias_lang	provincias_id	provincias_lang	provincias	id
provincias_id_pueblos	provincias_id	pueblos	provincias	id
pueblos_id_ofertas	pueblos_id	ofertas	pueblos	id
tipos_viviendas_id_ofertas	tipos_viviendas_id	ofertas	tipos_viviendas	id
tipos_viviendas_id_tipos_viviendas_lang	tipos_viviendas_id	tipos_viviendas_lang	tipos_viviendas	id
tipo_propiedad_id_ofertas	tipo_propiedad_id	ofertas	tipo_propiedad	id
tipo_propiedad_id_tipo_propiedad_lang	tipo_propiedad_id	tipo_propiedad_lang	tipo_propiedad	id
transacciones_id_ofertas	transacciones_id	ofertas	transacciones	id
transacciones_id_transacciones_lang	transacciones_id	transacciones_lang	transacciones	id

Fig 9 Foreign Keys de la base de datos

Ver estas relaciones de una manera más visual es posible gracias a MySQL workbench, donde podemos ver los diagramas de relaciones, y el tipo de relaciones que se establecen entre las diferentes entidades (1:n,n:m).

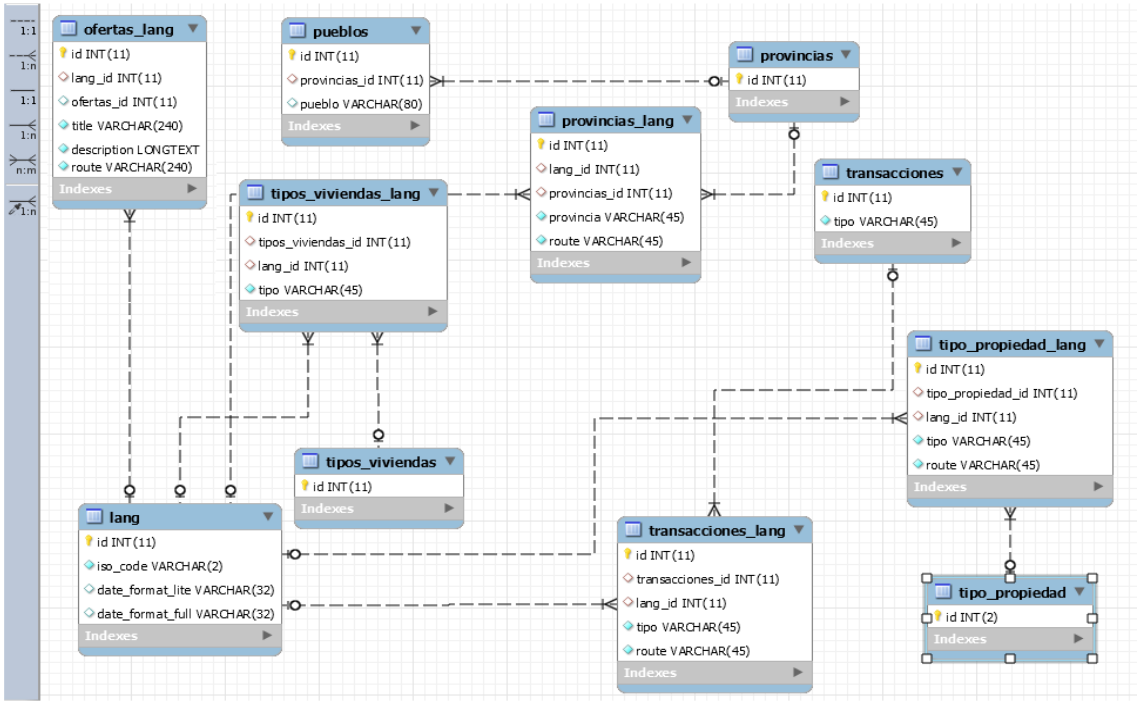

Fig 10 Relación BD y traducciones

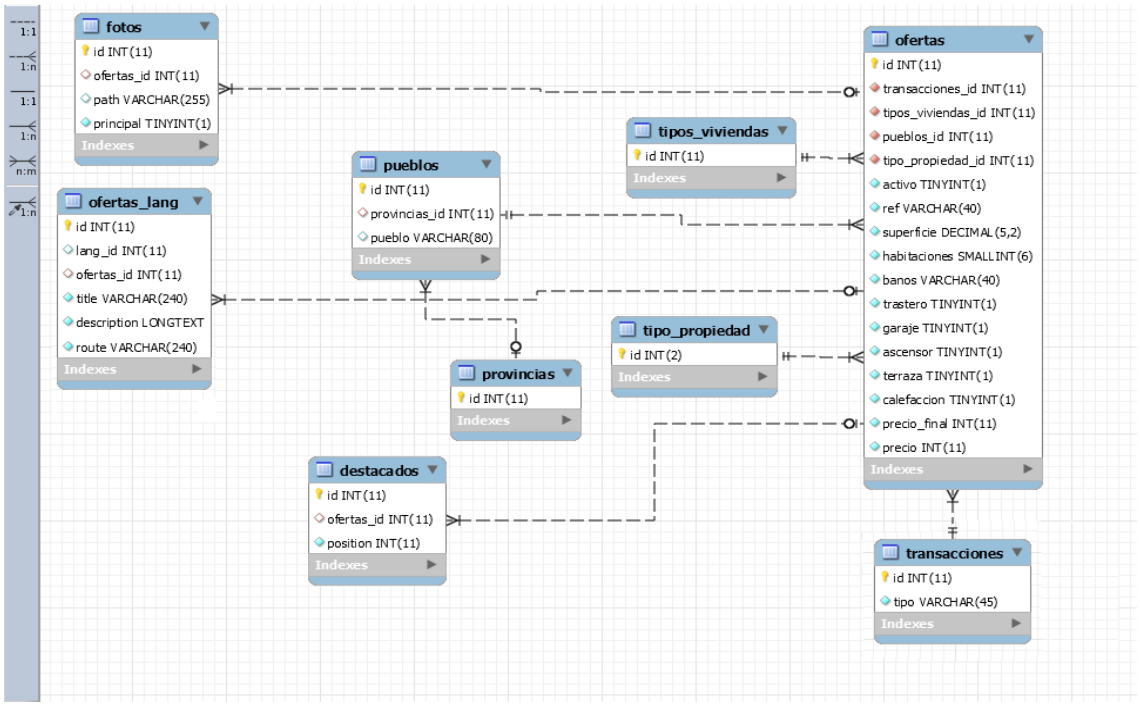

Ilustración 11 Ofertas y resto de entidades relacionadas

Importación de la base de datos

En esta sección se describe como se utiliza la web, su contenido y su DOM para obtener toda la información relativa a las ofertas.

Dentro del listado de todas las viviendas se obtiene la imagen principal y las rutas que nos llevarán a la descripción del inmueble. Esta obtención de datos se realiza desde el navegador dirección al servidor local.

Posteriormente, al tener todas las rutas de las viviendas, se crea otro servidor local que hace peticiones al servidor remoto para recoger cada una de sus características e introducirlas en la base de datos local, así como las fotografías.

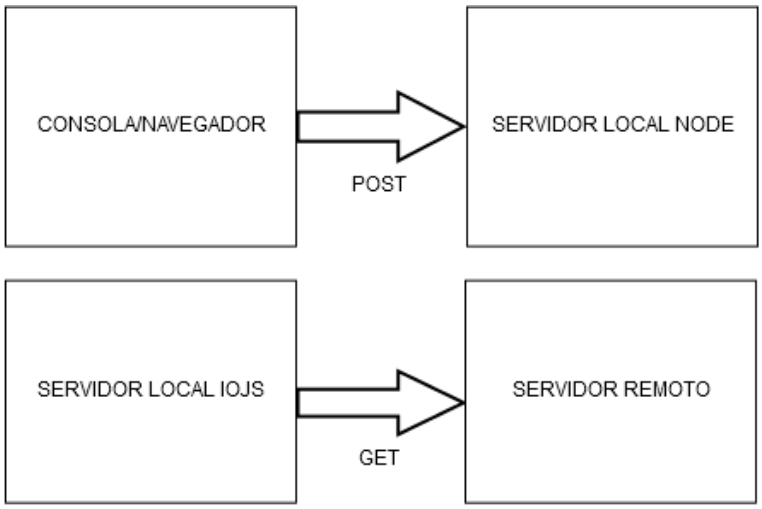

Fig 12 Importación de base de datos

Listado de viviendas

En el DOM de la lista se encuentran la información contenida dentro de unas clases y etiquetas html. Una unidad del listado tiene la clase *ea_shadow_border_objshort* que se utiliza para montar un *each()* JQuery con ese selector. En cada iteración se crea un objeto *data* JSON, cargando en el mediante diferentes selectores y métodos todos los datos, que se envían con una petición AJAX al servidor NODE local, el cual está escuchando esas peticiones para recibir los datos y actualizar o insertar. Ese Script corre dentro de la consola en el navegador en cada una de las páginas del listado.

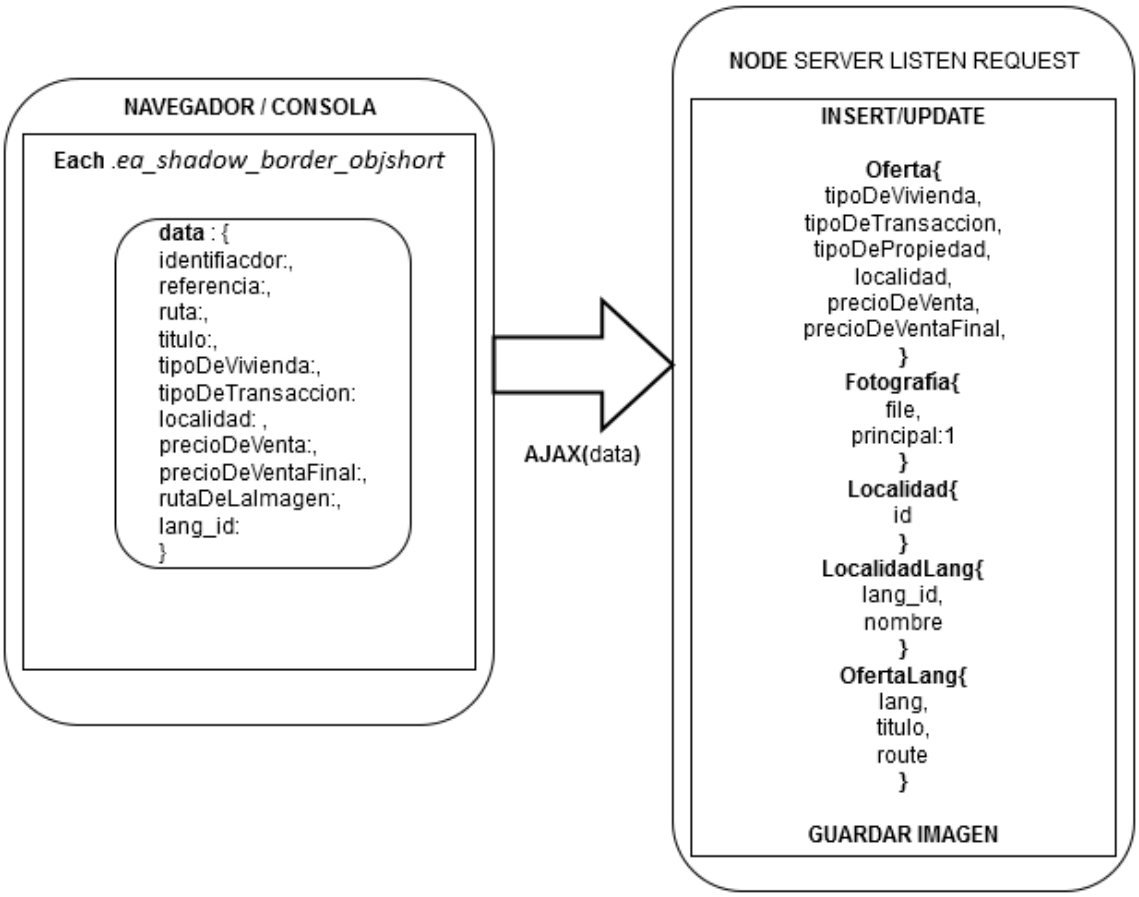

Fig.13 Captura de datos dentro del listado de viviendas

Captura datos en navegador

Como se explica en la figura 12 dentro del navegador, en la consola que proporciona en este cas Chrome, se ejecuta un Javascript utilizando la librería JQuery para iterar sobre los elementos de una página. En este caso usando el selector $('.ea_shadow_border_objshort') se obtienen todos los elementos del DOM que tengan esa clase. En la función pasada por parámetro al iterador each se accede a los datos que hay dentro de cada loop utilizando otros selectores para crear un objeto con esta descripción:

[*identificador de la oferta, referencia, ruta, titulo, tipo de vivienda, tipo de transacción, localidad, precio de venta, precio final, ruta remota de imagen_principal, lang*]

El **identificador** está contenido dentro de la ruta que nos lleva a ver el inmueble en detalle, solamente hay que romper la cadena de texto.

El lenguaje **lang** se calcula en función de la ruta en la que nos encontramos y se establece antes de entrar en el boocle. Básicamente diferencia si encuentra en el href el string '/es/' o '/eu/'.

La **ruta** se ha de enviar ya que luego se utilizará para hacer un bucle y obtener todas las viviendas. El resto de detalles son obtenidos fácilmente con un selector JQuery.

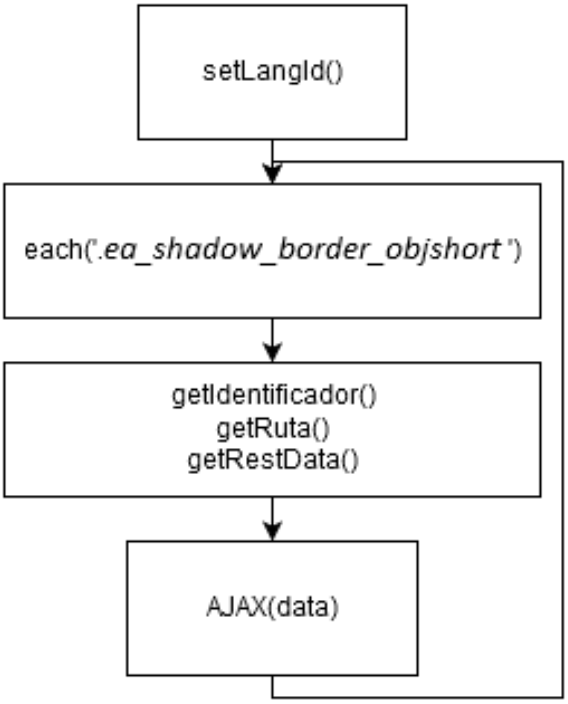

Fig 14 Carga de datos para petición AJAX

Captura de datos en servidor local

En el servidor local NODE se utilizan los módulos **mysql** y **http** para atender esas peticiones que han sido preparadas en navegador y actualizar o insertar en la base de datos utilizando el objeto data descrito anteriormente.

Antes de crear la oferta hay que obtener las claves foráneas de sus relaciones. En el caso de la **localidad** de no existir en la tabla se hace un *insert*, y si existe se busca y se utiliza su identificador para insertarlo en la fila de ofertas. En cuanto al **tipo de vivienda**, **tipo de transacción** y la **propiedad** hay una estructura de control mediante sentencias condicionales que detecta de que tipo es y le asigna un identificador en función de las mismas, evitando queries, ya que el número de condiciones de control es muy pequeño. En el caso de tipo de transacción se trata entre diferencia obra nuevo o viviendas.

```
If(typoDeVivienda == 'Obra Nueva'){
        Vivienda_id = 1;
}else{
        Vivienda_id = 2;
}
```

El resto de datos (*identificador de la oferta, referencia, ruta, titulo, precio de venta, precio final,lang)* son obtenido directamente del objeto data.

Con las **imágenes** hay dos puntos a trabajar. El primero es descargar la imagen. Utilizando la ruta que tiene su atributo *src* y guardarla en una carpeta local mediante el módulo FS. Actualizar la base de datos para saber qué imagen está asociada a qué oferta es obligatorio, pero el atributo *src* ha de ser limpiado y para obtener el nombre de la fotografía, no su ruta remota. Esta fotografía se ha de saber que es la que está asociada a la oferta en el listado de ofertas, por lo que al insertar un nuevo registro en la tabla de fotos se marca como principal.

Vivienda individual

En el paso anterior se consigue obtener las rutas de todas las viviendas tanto para euskera como para castellano, que nos sirven para correr un programa en IOJS que recoja la respuesta del servidor remoto de cada una de las viviendas, y así obtener el resto de datos y fotografías que describen a las ofertas.

La forma de obtener los datos es utilizando un módulo (https://github.com/tmpvar/jsdom) que permite utilizar selectores JQuery dentro de IOJS. Se trata de alguna manera de hacer que se comporte como si fuera un navegador. Esto facilita mucho el trabajo utilizando un lenguaje ya conocido.

Para hacer este trabajo se crea un módulo IOJS que subdivide el trabajo en tres secciones: obtención de las **imágenes** asociadas a la oferta, **detalles** y **descripción**. El programa al ejecutarse recibe un parámetro [img, detalle, descripcion] y dentro de su ejecución se decide qué trabajo se realiza. En el caso de pasar el parámetro para las descripciones es necesario saber si se trata de euskera o castellano [eu,es]. Las llamadas a este programa se hace de manera manual.

Descargando imágenes:

De manera muy similar que en la captura de datos desde el navegador, se busca un elemento del DOM con las clases *.ea_wrapper .ea_left_col .shadowbox y* se hace un bucle con la función each() de JQuery. Dentro de este bucle en cada iteración se tiene acceso a todos los elementos cuyo padre sea el elemento actual. Uno de ellos es *img* y al obtener el atributo *src* se puede descargar la imagen. Con el módulo fs de NODE se lleva de la ruta remota a la local, y el nombre es almacenado en la base de datos. De darse el caso de que la imagen ya existe en el sistema de archivos o en la base de datos, se comprueba si es la que está activa, de ser así, no se actúa ya que se ha persistido con anterioridad.

Descripción:

Como estos datos son en castellano y euskera, el programa recibe un parámetro extra para determinar el identificador del idioma y, en función de éste, toma unos registros u otros de la base de datos. Así, selecciona unas rutas bilingües de las que se obtiene la descripción para luego actualizar o insertar sus registros correspondientes.

Es en este campo donde hubo que hacer un trabajo de limpieza de los links a páginas pornográficas. Esto resulta muy sencillo utilizando JQuery. Dado que en la descripción no se incorpora ningún hipervínculo de forma intencionada, se pueden eliminar simplemente buscando todas las etiquetas *a (anchor)* de la descripción y eliminando ese enlace html con el selector *.parent()*.

```
$('a').parent().remove()
```

Detalles:

Se trata de saber qué características no traducibles tiene la vivienda, y para ello se utilizan unas sentencias condicionales que cambiarán los valores que se la dan por defecto al inmueble. Existe un selector *.ea_default_border .ea_equipment* que permite, mediante la funcion *each(),* obtener los siguientes datos:trastero, ascensor, baños, superficie, calefacción, terraza y garaje.

Antes de actualizar la BD se limpia bien la entrada:

- la superficie que se trata de una columna de tipo decima(5,2) ha de evitar contener caracteres no válidos como *m2*, * * .
- El número de baños es de tipo integer, por lo que WC/Baños ha de eliminarse.
- La calefacción que antes se describía con un campo de texto (Gas natural, calefacción eléctrica…) ahora es un campo Booleano, por lo que todas las que tengan sean del tipo que sean pasarán a true.

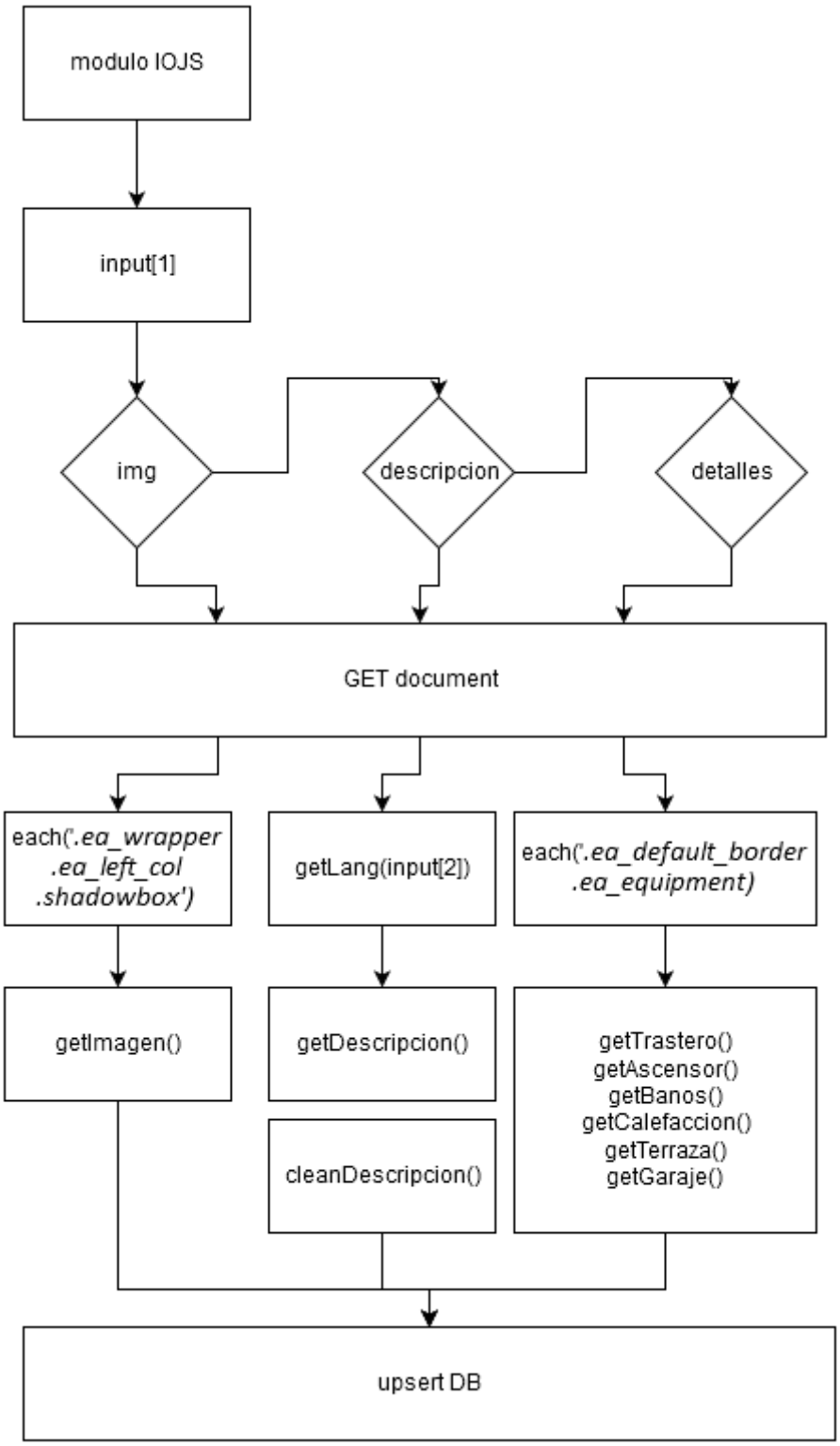

Fig 15 Módulo IOJS para captura de datos

Destacados

Esta sección que aparece en la página principal de la web consiste en las ofertas que quieren destacar. Esta parte no tiene mayor complicación que obtener los identificadores de las viviendas a mano e insertar los registros. Las imágenes que se presentan son las que las viviendas de las ofertas destacadas tienen como principales.

De la base de datos a entidades

Una vez que la base de datos está completa y bien relacionada entre sí es el momento de utilizar una de las herramientas más potentes e interesantes de Symfony2, reverse engineering (http://symfony.com/doc/current/cookbook/doctrine/reverse_engineering.html). De esta manera se construyen los objetos que se encargan de controlar la base de datos, con sus relaciones entre sí gracias a las Foreign Keys.

Ejemplo:

Por cada una de las tablas se crea una clase (ofertas => Ofertas.php), y por cada una de las columnas de la BD se crean propiedades en esa clase (precio, $precio). Para cada una de estas propiedades hay dos métodos que se crean automáticamente, el setter y el getter, que se encarga de actualizar o devolver el valor de esa propiedad. Cuando existe una relación entre dos entidades (Ofertas →OfertasLang) se crea una propiedad en Ofertas de nombre $ofertasLang. Esta propiedad contiene las traducciones de cada instancia de Ofertas, y siempre que se actualice el método setOfertasLang ha de recibir como parámetro una entidad OfertasLang, manteniendo la integridad referencial antes de llegar a hacer ninguna transacción.

El proceso es el siguiente: accediendo a la carpeta del proyecto por CMD, en este caso es *c:/wamp/www/App/*

1. `php app/console doctrine:mapping:convert annotation .scr/AppBundle/resources/config/doctrine/metadata/orm --from-database –force`
2. `php app/console doctrine:mapping:import AppBundle annotation`
3. `php app/console doctrine:generate:entities AppBundle`

Paso 1, doctrine se encarga de analizar la base de datos y obtener las tablas y sus relaciones. Con esto crea las clases mediante anotaciones con sus propiedades.

Paso 2, una vez revisado esto se importan las clases al módulo.

Paso3, teniendo las clases se generan los métodos que han de utilizarse para lectura y escritura de todas sus propiedades, ya que se tratan de propiedades privadas.

Así es como las entidades están listas para usarse, pero antes se han de hacer algunos ajustes dentro de cada una de ellas para que estén relacionadas entre sí (one-to-many o many-to-one) si se quiere aprovechar todo el potencial.

Ejemplo de relación one-to-many:

Una oferta está relacionada con muchas descripciones de esa oferta, tantas como lenguajes activos haya.

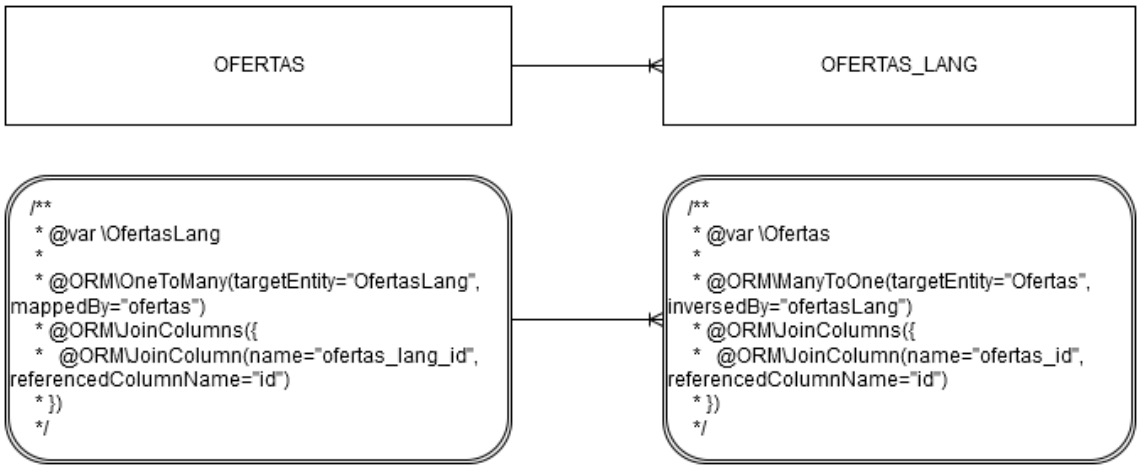

Fig 16 Ejemplo relación one-to-many, many-to-one

Al tener todas las relaciones mapeadas dentro de cada entidad se generan los *getters* y *setters* otra vez por consola.

```
Php app/console doctrine:generate:entities AppBundle
```

Repositorios de entidades

Solo con las entidades hay veces que no es suficiente para obtener toda la información necesaria, ya que hay casos en los que se requiere hacer una consulta un poco más compleja de lo normal. En el siguiente apartado están todas las entidades y los métodos que requerían ese tipo de sentencias DQL.

Destacados

- **findAllDestacados(lang_id)** Una consulta para poder seleccionar todas las ofertas activas que estén destacadas para aparecer en la página principal ordenadas por la posición de manera ascendente.
- **findAllNoDestacados(lang_id)** Seleccionar todas las ofertas activas que no estén destacadas.

Ofertas

- **findMaxPrecio()** & **findMinPrecio()** Son los métodos que devuelven el máximo y el mínimo de los precios para que el formulario de filtrado pueda construir el rango de intervalo de precios.
- **findMaxHabitaciones()** Otro método utilizado por el filtro de ofertas para construir el *checkbox* de número de habitaciones.
- **findAllActiveWithFile(id_lang)** Busca todas las ofertas activas de ese idioma que tengan una fotografía y devuelve la ruta a la imagen y el objeto ofertas.
- **findAllWithFileFilter(id_lang, filter)** Similar al anterior pero con filtrado.
- **findAllActiveFormWithFile(id_tipoPropiedad, id_transaccion, provincia_id, id_lang)** Busca todas las ofertas activas de ese idioma que correspondan con los parámetros propiedad, transacción y provincia.
- **findAllActiveFormWithFileFilter(id_tipoPropiedad, id_transaccion, provincia_id, id_lang, filter)** Similar al anterior pero con un filtrado.

ProvinciasLang

- **findAllActive(id_lang)** Cuando buscamos en los inmuebles sacar solo los pueblos que estén asociadas a ofertas activas.

TipoPropiedadLang

- **findAllActive(id_lang)** Cuando buscamos en los inmuebles sacar solo los tipos de propiedad que estén asociadas a ofertas activas.

TransaccionesLang

- **findAllActive(id_lang)** Cuando buscamos en los inmuebles sacar las transacciones que tienen asociadas ofertas activas.

Estructura del proyecto

Symfony2 es un framework que tiene unas pautas a la hora de crear un proyecto que han sido variadas mínimamente para organizarlo y separar algunos aspectos del front-end y del back-end.

Los controladores en el primer nivel de la carpeta son los dedicados al FE. En la carpeta de vistas, dentro de la llamada Backend existen cuatro carpetas donde viven las plantillas que necesitan los controladores.

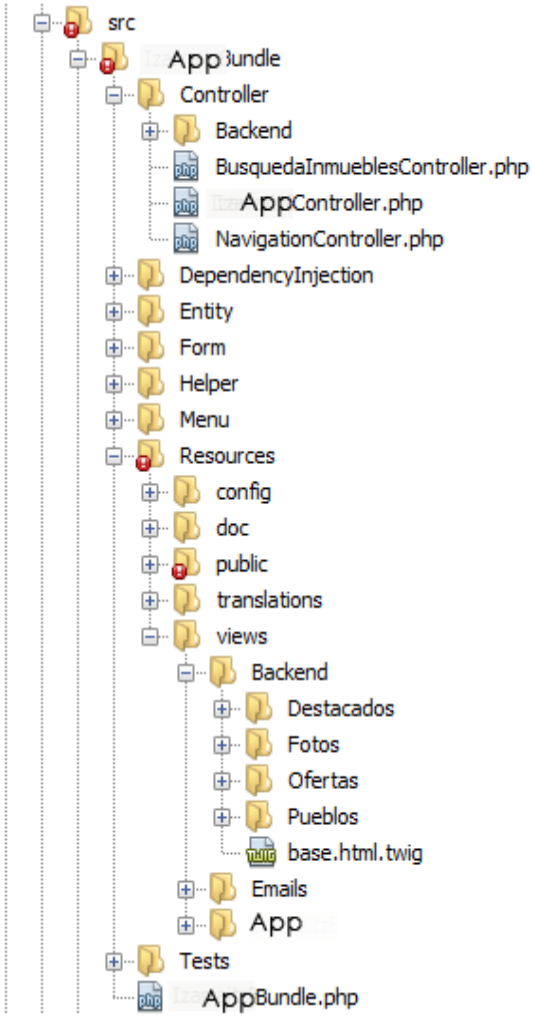

Fig 17 Estructura del Bundle

En la parte pública, solo comentar la carpeta uploads, que es donde se almacenarán todas las imágenes que el administrador sube. El resto sigue el patrón normal de SF.

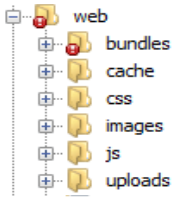

Fig 18 Estructura de la parte pública

Desarrollo del front-end

Esta sección es la responsable de mostrar las ofertas en los diferentes idiomas y posibilitar su filtrado. Esta es la parte más compleja de la web front. Además existen varias secciones estáticas como es el contacto, aviso legal y la localización, que son de rápido desarrollo.

Los pasos a seguir son estructurar bien el enrutamiento y las variables que se necesitan para generar el contenido. Ese enrutamiento determinará los controladores y acciones o métodos que se necesitan. Así, con esta estructura se crean las plantillas y los javascripts que sean necesarios por cada una de las acciones.

Enrutamiento

En esta sección se explica la estructura de rutas de la página que se configuran dentro del archivo *routing.yml* del bundle, pero no se entra en detalles del funcionamiento, ya que eso se tratará en la sección acerca de los *controladores.*

Multilingüe

A grandes rasgos existen 6 rutas: home, listado de inmuebles, vista de inmueble, localización, contacto y aviso legal, pero al existir otro idioma esto hay que multiplicarlo por dos. De haber n idiomas serían n*6 rutas. Esto es poco escalable y difícil de mantener, pero existen herramientas diseñadas para tratar este problema de muchas webs.

El i18n-routing-bundle hace que este problema no sea tan complejo. La idea es configurar en el archivo *routing.yml* un parámetro extra *locales* por cada ruta y pasarle la ruta que ha de escuchar a cada identificador de lengua (*locale*).

```
App_contacto:

    locales:  { es: "/es/contacto", eu: "/eu/kontaktua" }

    defaults: { _controller: AppBundle:App:contacto }
```

Esto hace más fácil dentro de las plantillas poder imprimir las urls utilizando la función *path()*.

Estáticas

Contacto

```
App_contacto:

    locales:  { es: "/es/contacto", eu: "/eu/kontaktua" }

    defaults: { _controller: AppBundle:App:contacto }
```

Localización

```
App_localizacion:

    locales:  { es: "/es/localizacion", eu: "/eu/kokapena" }

    defaults: { _controller: AppBundle:App:localizacion }
```

Aviso legal

```
App_aviso_legal:

    locales:  { es: "/es/aviso-legal", eu: "/eu/lege-oharra" }

    defaults: { _controller: AppBundle:App:avisoLegal }
```

Home

Escucha las peticiones a la carpeta raíz + variable *locale* y las envía al controlador *App:indexAction()* pero ¿qué sucede una petición a la carpeta raíz? Como bien se desprende del nombre del método al que apunta, se hará una redirección.

```
App:

    locales: { es:"/", eu:"/"}

    defaults: {_controller: AppBundle:App:redirect, _locale: 'es'}

App_homepage:

    locales:  { es: "/es", eu: "/eu" }

    defaults: { _controller: AppBundle:App:index }
```

Dinámicas

Listado de inmuebles

Por una parte, si el usuario visita la sección de inmuebles se encuentra con un listado completo de todas las existentes. Dentro de esta ruta se puede ir a otra ruta donde se realiza un filtrado de las viviendas por tres parámetros: tipo de propiedad, provincia, tipo de transacción.

A estas dos rutas se puede aplicar un filtro de los resultados según características, baños, calefacción, número de habitaciones, etc.

```
App_lista_inmuebles:

    locales:  { es: "/es/inmuebles", eu: "/eu/higiezinak" }

    defaults: { _controller: AppBundle:App:listaInmuebles }

App_busqueda_inmuebles:

    locales:  { es: "/es/{transaccion}/{tipoPropiedad}/{provincias}", eu:
"/eu/{transaccion}/{tipoPropiedad}/{provincias}" }

    defaults: { _controller: AppBundle:App:busquedaInmuebles }
```

Inmueble

Vista completa del inmueble con todos los detalles y las fotografías activas.

```
App_vista_inmueble:

    locales:  { es: "/es/inmueble/{id}/{title}", eu: "/eu/jabetza/{id}/{title}" }

    defaults: { _controller: AppBundle:App:vistaInmuebles, _translate: "title" }
```

Helpers

Es importante describir los *helpers*, que sirven para poder trabajar y organizar la lógica de negocio. Es una manera de organizar el código por funciones que se necesitarán dentro en la aplicación. Así el código queda más limpio y reutilizable. En la aplicación hay tres:

RoutingHelper

Utilizado para limpiar rutas generadas de forma dinámica según el título de la vivienda.

OfertasHelper

Es el encargado de construir la sentencia DQL que filtra las viviendas listadas, para ello utiliza las variables GET de la ruta.

InmueblesHelper

Crea el intervalo de precios que sirve para filtrar las ofertas por rango.

FormTypes

Son clases que extienden *AbstractType*, estos son los encargados de construir los formularios dentro de Symfony. De esta manera este proceso se saca de los controladores, haciendo el código más limpio y reutilizable. Dentro del front-end tenemos los siguientes:

PropiedadTransaccionProvinciaType

Encargado de construir el formulario que hará la búsqueda de ofertas por tipo de vivienda, propiedad y provincia. Dependiendo del *locale* se configura con textos traducidos. Los siguientes métodos son utilizados para construir el formulario:

- ***private* translateForm(locale)** Traduce el botón *submit* y los nombres o placeholders de los *selects*.
- ***private* createTipoPropiedadLangChoices (locale)** Listado de propiedades a elegir.
- ***private* createTipoTransaccionChoices (locale)** Listado de tipos de transacciones.
- ***private* createProvinciasChoices (locale)** Listado de provincias activas.
- ***public* buildForm(FormBuilderInterface $builder, array $options)** Construye el formulario a partir del *builder* de entrada y las opciones.
- ***private* getName ()** Le da un nombre que debe ser único para este formulario.

Controladores

A estas alturas, las rutas han sido descritas y configuradas. Ahora es momento de describir el funcionamiento de cada una de ellas.

Tenemos métodos que pueden ser públicos o privados. Los primeros son los que están apuntados por el ruter, pero para hacer el código más limpio y legible, la idea es reducir el número de líneas de cada uno de ellos. Según la guía de buenas prácticas de Symfony, el número límite de un método de un controlador ha de ser 20, por lo que en ciertos casos es necesario utilizar métodos y variables privadas cuya tarea sea la de reducir ese tamaño.

NavigationController

Se encarga de enrutar correctamente el cambio de idioma según donde esté el usuario. En la web anterior el cambio de idioma se realizaba desde cualquier punto a la página index, pero en esta se ha intentado mantener una memoria de estado, exceptuando cuando el filtro para buscar inmuebles está activo. El filtro al tratarse de un formulario y el cambio de idioma un link, aumenta demasiado la complejidad para mantener esos valores, por lo que se opta para una siguiente fase si el cliente lo desea.

Este controlador es llamado desde la plantilla base, por lo tanto, desde todas las páginas, se le pasan los parámetros de la petición HTTP. Dentro del controlador se utiliza el servicio *router* de Symfony2 para construir la ruta que se le pasará a la plantilla renderizada. Este *twig* solo escucha la ruta que tiene que imprimir. Dentro de los parámetros *request* obtiene el local en el que está para poner un idioma activo y otro inactivo.

Se dan tres posibles casos para transformar la ruta a los idiomas activos en función de la ruta en la que esté el usuario:

App_vista_inmueble:

Dentro del método *vistaInmueble(),* se busca la oferta y sus traducciones ordenadas por el identificador de lenguaje de forma ascendente. Se generan las rutas con el identificador y el título de la oferta.

App_busqueda_inmuebles:

Aquí se reciben más parámetros que hay que traducir y para ellos se utilizan tres métodos privados:

- getTransaccionLang(Transaccion)
- getProvinciasLang(provincia)
- getTipoPropiedadLang(propiedad)

El proceso es el mismo que el anterior, se buscan y ordenan por el identificador del lenguaje de forma ascendente. Las rutas se generan con esos parámetros.

Resto

Se obtiene la ruta y se traduce a los dos idiomas.

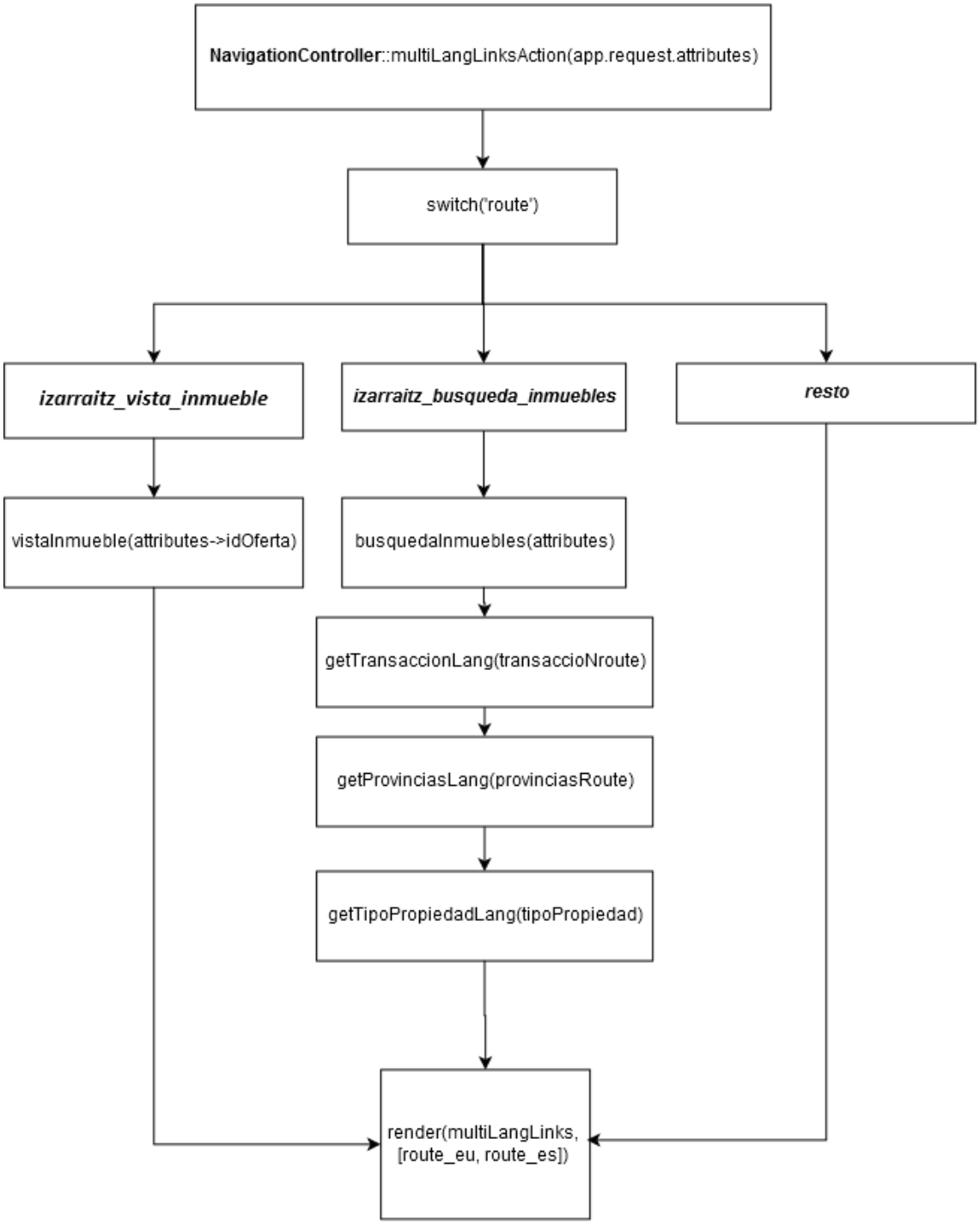

Fig 19 Lógica del controlador de navegación

AppController

Éste es el controlador de mayor tamaño de toda la aplicación, aquí viven los métodos que generan casi todo el front-end de la web. Se listan los siguientes métodos públicos:

IndexAction

Consulta a la base de datos utilizando el repositorio *Destacados*, con el método *findAllDestacados()*. Este método nos devuelve las entidades ofertas, ofertas *lang* y la ruta a la fotografía principal de la oferta.

listaInmueblesAction

Encargado de obtener el listado con todas las ofertas activas. Aquí viven dos métodos privados:

- **checkListaInmueblesFilter(Request)** Comprueba si le llega por el filtro con la solicitud (*request*). De ser así llama al *helper Ofertas* que recibe los parámetros de la URL para generar la sentencia DQL que devolverá el resultado pedido. En caso contrario obtiene todas las ofertas.
- **listaInmueblesForm(Request)** Genera un formulario capaz de hacer una búsqueda más refinada de las viviendas. Ese formulario se crea con el FormType *PropiedadTransaccionProvinciaType*.

Además se utiliza el *knppaginatorbundle* para paginar los resultados de las ofertas.

vistaInmueblesAction

Se busca la *ofertaLang* por su identificador e identificador de lengua. Al estar relacionados *ofertas* y *ofertasLang* el resto de campos se obtienen directamente. Por último se buscan las fotografías asociadas a esa oferta.

Contacto

Este método renderiza la plantilla *contacto.html.twig* pasándole un formulario de contacto, pero antes hay tres métodos privados que serán utilizados para hacer el código más comprensible.

- **setContactVariables(request)** Según la variable *locale* configura el formulario y los mensajes flash informativos de si ha habido éxito o no en el envío del correo. Variables privadas [formParams, flash]
- **createContactForm (request)** Crea el formulario para ser renderizado por la plantilla.
- **checkConcactForm (request)** De ser enviado el formulario al servidor, este método analiza y valida. De ser correcto el contenido del formulario, se envía el correo al administrador de la inmobiliaria e informa de que ha salido todo correctamente. En caso contrario se informa de que ha habido un error en el envío.

BusquedaInmueblesController

Encargado de mostrar el formulario de búsqueda-filtrado. Este controlador tiene varios métodos privados que ayudan a limpiar el código. Primeramente se llama a *createAction(Request)* y aquí se obtiene el identificador del lenguaje. Con este identificador se llama a los siguientes métodos:

- **getTipoDeVivienda(idLang)**
- **getTransaccion (idLang)**
- **getTipoPropiedad (idLang)**
- **getPueblos (provincias)** Saca un listado de pueblos que pertenecen a una provincia, en caso de ser *null* la provincia obtiene todos los pueblos.
- **getMaxHabitaciones ()** Busca el número máximo de habitaciones dentro de todas las ofertas, limitando ese número a 4.

La finalidad de dedicarle un controlador es poder utilizar este formulario en otro punto de la web (*index)*.

Trabajando con plantillas

En este trabajo se está funcionando con plantillas *twig* y algunas funcionalidades de Symfony2 como es *assetic*, que es una manera muy buena de gestionar archivos CSS, JS e imágenes. Una de las funcionalidades más destacables es la de controlar las versiones de los archivos para entregar al cliente siempre la última versión.

Antes de entrar en detalle de cómo funciona cada una ha de saberse que todas heredan las propiedades de una plantilla padre, lo que ayuda a estructurar el código y facilitar los cambios. Este archivo es llamado base.html.twig que se explicará con más detalle a continuación.

Dentro de algunas de ellas se ha incluido algo de código Javascript para darle funcionalidad o trabajar el diseño adaptativo con JQuery.

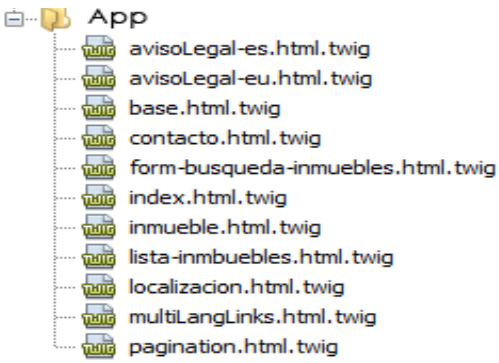

Ilustración 20 Listado de twigs para el front end

Bloques de la plantilla base

Se trata de las partes de la plantilla que heredan las plantillas hijas. Dentro éstas, se puede retocar estos bloques para añadir código JS, CSS o cambiar el algún valor.

Bloque de css: aquí es donde van los archivos de bootstrap y las hojas de estilo propio.

Bloque de javascripts: archivos bootstrap, jquery y algunas funciones básicas de javascript.

Bloque Title: es el título de la página web que cambia en cada sección.

Navegación: un listado de las secciones de la página. Para mantener el estado actual dentro del menú se utilizan variables globales dentro de la plantilla como es app.request.attributes, con el método get('_route'). De esta manera al conocer la ruta actual se le da una clase (.*active*) al <li> padre del link que apunta a esa sección.

Bloque de contenido: donde cada una de las plantillas tiene espacio para poner su contenido específico.

Footer: es una llamada al controlador-acción **AppBundle:avisoLegal** para imprimir el aviso legal en el idioma activo y utilizar esa respuesta para hacer una ventana modal de bootstrap.

Diseño de plantilla base: se trata de dar una estructura con una disposición de los elementos. Container tiene un ancho determinado y container-fluid es el 100% de la ventana.

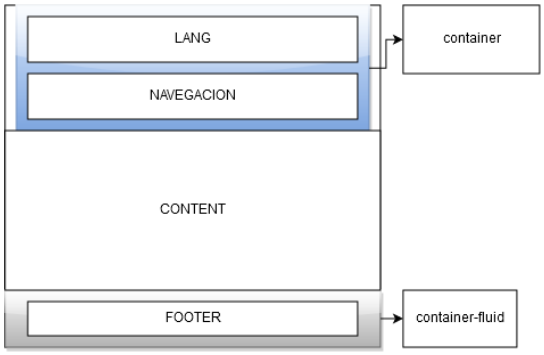

Ilustración 21 Diseño para todas las páginas

Plantillas

Index

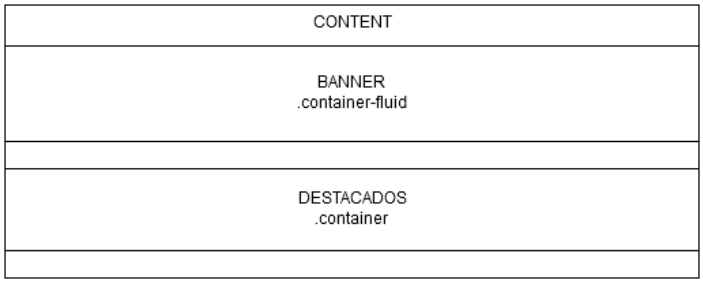

Ilustración 1 Estructura de la plantilla index

En esta plantilla hay dos secciones, el *banner* rotativo (*bootstrap default*) que ocupa todo el ancho de la pantalla y la sección de destacados. Esta última sección necesita de un código JS para reescalarse, y su función es buscar aquel de todos los *banners* que tiene el tamaño más grande y luego aplicárselo al resto. Este código se ejecuta siempre que hay un reescalado de la ventana.

En la sección destacados está preparado para que encajen 4 en una fila utilizando col-lg-3.

Listado de inmuebles

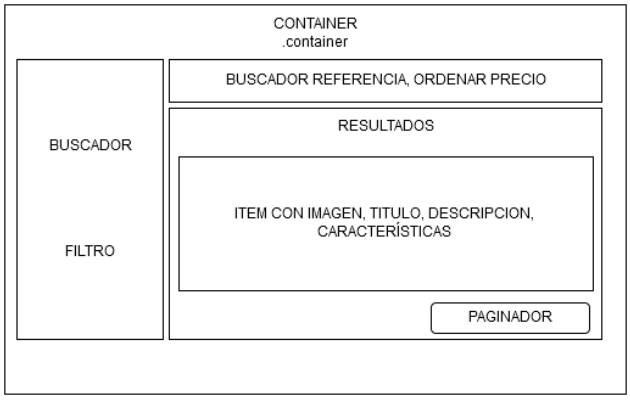

Ilustración 2 Estructura del listado de inmuebles

En la columna izquierda está el formulario de búsqueda de ofertas por provincia, transacción y tipo de propiedad. En su parte inferior está el formulario de filtrado.

La columna derecha tenemos dos bloques: ofertas y buscador-ordenador. Las ofertas son el listado de los resultados devueltos por el servidor, con la imagen principal reescalada y con

marca de agua gracias al *bundle* ***gregwar/image-bundle***. El buscador-ordenador son dos inputs (select y text) que se encargan de buscar por referencia una oferta, y ordenar de forma ascendente o descendente las ofertas por su precio. Aquí es necesario utilizar un código JS para que:

- Ordene los precios al cambiar el estado del input de tipo select
- Busque la referencia

La idea es enviar variables GET a OfertasHelper teniendo en cuenta si ya existe un filtrado previo. Buscando en *window.location.href* la existencia de la palabra *filter* se puede saber si el filtro está activo, de estarlo se añade al *href* otra variable más, en caso contrario se añade a la ruta ?filter=true&var1=val1.

Finalmente está el paginador de *knpbundles*.

Inmueble

Aquí está toda la información de la oferta. En la columna izquierda está la superficie, tipo de vivienda, precio, localidad, habitaciones, baños, trastero, garaje y calefacción. Las que son de tipo Booleano van acompañadas de un *gliph-icon* de bootstrap. En la columna derecha están todas las fotografías con su marca de agua.

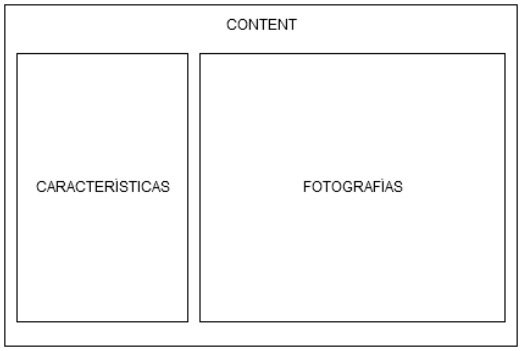

Fig 23 Estructura de página inmueble

Contacto

Al utilizar mensajes flash para informar al usuario del éxito en el envío del correo electrónico hay que reservar una parte dentro de esta plantilla para mostrarlos. En cuanto al formulario, al querer darle un diseño más personalizado se ha de tratar no solo con la función {{form(form)}} de twig.

- form_start(form, {'attr': {'class':'form-horizontal'}})
- form_errors(form.inputName)
- form_widget(form.inputName, {'attr':{'class':'form-control', 'placeholder':'Texto'}})
- form_end(form)

Esta misma configuración se podría hacer dentro de la clase que crea este formulario.

Localización

Mapa donde poder localizar la oficina de la inmobiliaria utilizando un *plugin* Javascript jquery.gmap.min.js (http://github.com/marioestrada/jQuery-gMap) que necesita de JQuery para funcionar además de la api (http://maps.google.com/maps/api/js?sensor=false) de Google.

Cargar el mapa es muy sencillo. Necesitamos un elemento del DOM y cargarle la función gMap con los siguientes parámetros de configuración:

- **maptype(string)** el tipo de mapa, en este caso ROADMAP
- **scrollwheel(boolean)** false
- **zoom(integer)** la cantidad de zoom que queremos [0,25]
- **markers(obj)** [‘address’:’Jose de artetxe kalea 8 – 20730 Azpeitia]

Es muy importante darle un tamaño CSS para poder ver el mapa.

Aviso legal

La única plantilla que no utiliza el traductor de Symfony2. En este caso se han utilizado dos plantillas: aviso-legal.es.html.twig y aviso-legal.eu.html.twig. Simplemente es texto y html.

Traducciones

En la sección de la base de datos se explica cómo se ha planteado la estrategia para traducir los textos de las ofertas, pero para otras traducciones que viven dentro de las plantillas se trabaja con la herramienta de traducciones de Symfony2 (http://symfony.com/doc/current/book/translation.html).

La manera de trabajar es la siguiente:

1. Creamos página web con todas los textos de las plantillas rodeadas de la etiqueta {%trans%}Texto{%endtrans%}
2. Finalizada la web se crea el archivo *messages.eu.xlf* dentro de la carpeta AppBundle\Resources\translations
3. Cada uno de los textos rodeados por las etiquetas se traduce en un archivo xml

```
<?xml version="1.0"?>
<xliff version="1.2" xmlns="urn:oasis:names:tc:xliff:document:1.2">
    <file source-language="es" datatype="plaintext" original="file.ext">
        <body>
            <trans-unit id="1">
                <source>Hola Mundo!</source>
                <target>Kaixo Mundo!</target>
            </trans-unit>
        </body>
    </file>
</xliff>
```

Desarrollo del Back End

El gestor de este nuevo sitio web pretende ser mucho más sencillo que el anterior realizado en Joomla, donde los pasos para crear una nueva oferta sean sencillos así como destacar las viviendas.

Las secciones de este panel de control son:

- **Ofertas**: Listado de las ofertas, con buscador por referencia.
- **Fotos**: Listar las fotos con un buscador por referencia de oferta. Poder poner cualquiera de las fotografías como principal y subir una foto para asignársela a una oferta.
- **Localidades**: Listar las localidades y poder editar tanto el nombre como a la provincia a la que pertenece, así como eliminarla.
- **Destacados**: Listar todas las ofertas y con un checkbox seleccionar las que se quiere destacar. Poder editar el orden en el que se destacan.

Panel de control para website CRUD

Utilizando en generador CRUD basado en una entidad de doctrine (http://symfony.com/doc/current/bundles/SensioGeneratorBundle/commands/generate_doctrine_crud.html) con Symfony2 proporciona una estructura de controladores, formulario y vistas que nos sirve como comienzo para trabajar. Cambiando la plantilla base y algunos métodos de los controladores se consigue un trabajo rápido y eficaz, pero hay que contar el añadido de tener que editar los campos de manera multilingüe.

Mediante el uso de la consola y el siguiente código

```
php app/console generate:doctrine:crud --entity=AppBundle:Pueblos
```

se genera la siguiente estructura dentro del bundle.

```
AppBundle
        Controlers
                PueblosControler.php
        Form
                PueblosType.php
        Resources
                Views
                        Pueblos
                                Edit.html.twig
                                New.html.twig
                                Index.html.twig
                                Show.html.twig
                Config
                        Routing
                                Pueblos.yml
```

Además de importarse las rutas al archivo *routing.yml* que se han creado en la sub-carpeta routing, el controlador tiene los métodos para poder editar, crear, indexar y ver todos los

pueblos existentes. Además se crean las clases que extienden *AbstractType* dentro de la carpeta forms para poder utilizarse a la hora de editar esos objetos. *Esto* sirve como punto de partida para poder empezar.

Antes de entrar en los controladores y las rutas existe un paso previo para poder continuar el trabajo. La organización del back-end se prefiere separar del front-end para que las vistas y los controladores vivan dentro de una sub-carpeta, quedando la organización de la siguiente manera:

```
AppBundle
        Controlers
                Backend
                        PueblosControler.php
Resources
                Views
                        Backend
                                Pueblos
                                        Edit.html.twig
                                        New.html.twig
                                        Index.html.twig
                                        Show.html.twig
```

Este cambio obliga a cambiar los *namespaces* de los controladores y las rutas de las plantillas renderizadas. En los routing de cada uno de los CRUD se ha de actualizar las rutas a los controladores añadiendo *Backend* en el parámetro *defaults*.

Name space del Bundle:

AppBundle/Controller/Backend

Cambio en el archivo *pueblos.yml*

```
admin_pueblos:
    path:     /
    defaults: { _controller: "AppBundle:Backend/Pueblos:index" }
```

Plantillas:

Cómo se puede ver arriba hay 4 plantillas para todos los controladores, pero uno se elimina del comportamiento (READ o SHOW), porque dentro del listado o de la edición se pueden ver todos los campos.

Controladores

Ya se han descrito todas las clases y métodos creados para poder implementar un gestor CRUD de las entidades. Cuando se trata con entidades y relaciones sencillas no existe mayor complejidad, Symfony2 ya ha hecho el trabajo, pero en algunos casos no es suficiente (véase caso ofertas). La idea es reducir el tiempo que se dedica para hacer cambios en la web, por lo

que se trata de implementar un comportamiento especial, como por ejemplo en la entidad Destacados. Esto implica la creación de nuevos métodos que se ocupen de tareas para implementar ese diseño.

DestacadosController

Encargado de insertar en la tabla de destacados los identificadores de las ofertas que aparecerán en la página principal, y el orden en el que deben aparecer.

indexAction

Lista todas las ofertas, primero las que están destacadas acompañadas de un checkbox checked, y después el resto. Se pueden seleccionar tantas ofertas como se quieran destacar, pero es importante que por lo menos haya una oferta seleccionada.

Cuando llegan los identificadores por el método POST se pasa por dos controladores y una variable privada, getDestacados (Input POST de los identificadores a destacar).

eliminarDestacados

Se encarga de recorrer con un bucle todos los destacados, en caso de no estar el identificador dentro de los recibidos por POST entonces lo elimina de la BD, en caso contrario se hace *unset* dentro de getDestacados eliminando ese identificador.

setDestacados

Los identificadores que hay en setDestacados pasan a existir en la tabla y todos tienen la posición 0, por lo que saldrán en primer lugar.

ordenarAction

Mediante unas entradas de tipo texto asociadas a cada oferta se introduce la posición en la que se desea poner el destacado. En caso de no ser un valor numérico pasa a ser cero.

FotosController

Encargado de listar todas las fotografías, poder subir, eliminar y editarlas. Además de tener la funcionalidad de buscarlas por referencia.

indexAction(request)

Aquí se crea un formulario utilizando el método *createFormBuilder* con la entidad ofertas como entrada, y añadiendo el campo referencia. Si llega la referencia por este formulario se ajusta la búsqueda, en caso contrario se listan todas.

createAction(request)

Un formulario utilizando el método *createFormBuilder* con la entidad Fotos como entrada. Al recibir los datos en caso de que exista otra foto asociada a esa oferta que sea la principal y la que se recibe está configurada para serlo, entonces se cancela a la otra como principal.

editAction(id)

Aquí existen dos formularios creados con dos métodos privados, el de editar (archivo, principal y referencia) y el de eliminar.

updateAction(request,id)

Al editar, si existe otra imagen que esté como principal, esta pasará a no serlo con el método *unsetNoPrincipales(entity).* No se permite cambiar una imagen principal a no principal, ya que siempre tiene que existir una para cada inmueble.

deleteAction(request, id)

Al eliminar si no se trata de la fotografía principal se elimina directamente. En caso contrario hay do maneras de actuar: cuando se trata de la única fotografía, cuando hay más fotografías.

createFotoDefault()

En el primer caso se crea una nueva entidad con la fotografía base que tiene, se elimina la otra y se actualiza la base de datos.

ponerPrimeraFotoDefault(fotos, entity)

En el caso de haber más fotografías se coge la primera y se pone la principal, después se elimina la otra.

OfertasController

Este es el controlador con mayor complejidad tanto en su lógica como en los componentes que necesita. Se encarga de listar todas las ofertas, buscarlas por referencia, crearlas y eliminarlas.

indexAction(request)

Aquí se crea un formulario utilizando el método *createFormBuilder* con la entidad ofertas como entrada y añadiendo el campo referencia. Si llega la referencia por este formulario se ajusta la búsqueda, en caso contrario se listan todas.

createAction(request)

Un formulario utilizando el método *createCreateForm* con la entidad ofertas como entrada. Al recibir los datos en caso de que todo sea correcto:

createOfertasLangDefault(entity)

Crea con unos atributos por defecto en todos los idiomas que estén activos, la descripción, la ruta y el título.

createOfertasFotosDefault (entity)

Crea una fotografía principal por defecto para la oferta.

editAction(id)

Obtiene todas las fotografías, los formularios OfertasLang asociados a esa oferta y la propia oferta.

updateAction(request,id)

Al recibir todos los campos correctamente actualiza la base de datos. OfertasLang y Fotos son actualizados dentro de sus controladores.

deleteAction(request, id)

Eliminar una oferta hace que todo lo relacionado con la misma desaparezca, tanto en la base de datos como sus fotografías.

OfertasLangController

Este controlador solo recibe desde el controlador Ofertas un formulario para editar los campos título, ruta y descripción. Al actualizar los datos correctamente vuelve a la edición de la oferta dentro del controlador Ofertas, en caso contrario devuelve un flash error.

PueblosController

Ese controlador no sufre ningún cambio y se utiliza el que viene por default dentro del generador CRUD. Cuando se trata de entidades tan sencillas como ésta con una relación a provincias *many-to-one* no hay mayor problema.

Plantillas

Se ha incluido algo de código Javascript para mejorar el aspecto y comportamiento de los controladores, intentando ganar en usabilidad como en diseño. El generador CRUD de Symfony2 crea siempre 4 plantillas:

- edit.html.twig
- show.html.twig
- index.html.twig
- new.html.twig

Estas plantillas tienen un padre que al cambiarlo y ponerle la plantilla con los estilos de *bootstrap* ya adquiere un aspecto visual mucho mejor, pero aún hay que hacer ciertos retoques. Las tablas, los botones, formularios y la navegación pasan a tener las clases por defecto del *framework*, así como los mensajes flash de error que también son cambiados.

Formularios:

Normalmente un formulario se renderiza utilizando la función dentro de *twig form(form)*. En este caso y para mejorar el aspecto hay que entrar en cada una de las entradas para poder cambiar valores como la etiqueta (*label*) o la clase.

Para los inputs de tipo text, checkbox, select:

```
{{ form_label(form.ofertas, 'Referencia') }}

{{ form_widget(form.ofertas, { 'attr': {'class': 'btn-btnsuccess' }}) }}
```

Para los inputs de tipo submit :

```
{{ form_widget(form.submit, { 'attr': {'class': 'btn btn-success subir' }, 'label' :
'Actualizar'}) }}
```

Ofertas

En este caso la plantilla aumenta de complejidad ya que recibe diferentes entidades como son ofertas, ofertasLang con sus dos formularios y Fotos. Debido a tanto contenido su utiliza un panel para subdividir el contenido por entidades, y cada uno de esos paneles tiene un botón de *submit*.

Las descripciones de las ofertas tienen un campo *textarea* que se carga con el *bundle* CKEDITOR para crear contenido HTML y poder dar ciertos estilos CSS.

Conclusiones

Por parte del cliente he recibido un feedback bastante bueno. La persona que se dedica a gestionar las ofertas está muy satisfecha, ya que según ella el tiempo para hacer las típicas gestiones de edición y creación se ha reducido considerablemente.

- Encuentra el panel de control muy sencillo e intuitivo.
- El buscador por referencia tanto de ofertas como de fotografías evita hacer el scroll.
- El número de operaciones para hacer una oferta las ha reducido.

El front end también ha tenido un cambio que a ellos les ha parecido mejor que el anterior. Además de que tiene la capacidad de adaptarse para dispositivos móviles y tablets.

En cuanto al trabajo desde el punto de vista de programación me siento bastante satisfecho con el resultado, los retos que he tenido que asumir y la manera en que he podido solucionarlos. Siempre en función de un tiempo y dinero limitados.

No sólo hay una parte de programación y de conocimientos técnicos, también está la parte humana. Hay que captar las necesidades del cliente, analizar la problemática, presentar propuestas, etc. Básicamente gestionar un proyecto desde el punto cero hasta la entrega.

Dentro de un proyecto los imprevistos se pueden dar o siempre ocurren. Intentar organizar todo antes de ponerse a trabajar es una manera de gestionar el tiempo y la inversión de que se dispone, pero cuando te encuentras con este tipo de problemas existen dos posibles soluciones:

- Trabajar más para llegar al punto de calidad previsto
- Reducir la calidad del trabajo y dedicarle el tiempo asumido

La primera solución es viable siempre y cuando el cliente esté dispuesto a pagar más, cosa que no suele suceder. La segunda es la típica, pero aquí hay que lidiar con el cliente, ya que se le tiene que hacer entender que no se va a llegar al objetivo planteado. Tomando el trabajo con la segunda solución hace que la calidad disminuya, pero siempre puede haber otro momento en el que se pueda hacer.

Las partes del trabajo que necesitan de una revisión por lo comentado en el párrafo anterior son:

- El diseño del front end
- El diseño del back end

Mejoras:

En cuanto a posibles mejoras del front end:

- Cambio de idioma cuando el filtro está activo no guarda ese estado
- Añadir un “carrito” de viviendas, donde los usuarios puedan ir añadiendo esas ofertas y luego imprimir un PDF con todas sus características.
- Dentro de la visualización de la vivienda poder descargar la información en PDF.
- Tener un perfil de usuario para poner alertas cuando las viviendas bajen de precio.
- Poder poner mensajes entre el visitante y el gestor sobre una vivienda, una especie de chat.

En el back end:

- La parte de ofertas creo que estaría mucho mejor tener todo en una pantalla.
- Ordenar los destacados utilizando un drag-n-drop.
- Integrar las viviendas creadas dentro de las bases de datos de otras plataformas como idealista.

Futuros cambios en un proyecto pueden suceder y durante esos cambios se puede modificar algo involuntariamente que trastoque las funcionalidades. Esto puede no ser detectado en el entorno de desarrollo y acabar en el de producción, generando una situación difícil, ya que habría que depurar mientras no funciona la web. Evitar esto se podría hacer mediante control de versiones, para hacer un rollback y volver a la versión funcional, o algo mucho más interesante pero costoso a la vez: desarrollo con tests. Algo que todavía no conozco pero que suena muy interesante.