UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Desarrollo de una aplicación web para la gestión de recorridos virtuales formados por imágenes panorámicas

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Carlos Verdú Marco

Tutor: David Picó Vila

Curso académico 2013-2014

Resumen

En este proyecto se documenta el proceso de desarrollo de una aplicación web para la creación, edición y gestión de recorridos virtuales formados a partir de imágenes panorámicas (360º). La aplicación da la posibilidad de, entre otras cosas, añadir a éstas elementos denominados *Hotspots*, los cuales permiten al usuario navegar entre las distintas imágenes que componen el recorrido así como ver información o contenido multimedia relacionado con los lugares representados en la imagen panorámica como, por ejemplo: Comercios, museos, edificios emblemáticos, etc.

Palabras clave: Imagen, panorámica, recorrido, virtual, punto, aplicación, web

Abstract

This project documents the development process of a web application for the creation, edition and management of virtual tours formed by panoramic images (360º). The application allows the user to, among other things, add to these panoramic images elements called Hotspots, which allow the user to navigate between the different images that form the tour as well as see information or multimedia content related to the places shown in the image like, for example: Shops, museums, emblematic buildings, etc.

Keywords: Image, panorama, virtual, tour, hotspot, web, application

Tabla de contenidos

1. Introducción

1.1 Presentación del proyecto

En este proyecto, se documenta el proceso de desarrollo de una aplicación web a partir de los requisitos establecidos por un cliente. Esta herramienta permite al cliente subir imágenes panorámicas tomadas previamente a un servidor y, a partir de éstas, crear recorridos virtuales personalizados. La aplicación también permite, entre otras cosas, añadir elementos llamados *Hotspots* a las imágenes, los cuales pueden tener distintas funciones dependiendo del tipo añadido. Principalmente, se utilizan los tipo flecha, que permiten navegar entre las distintas escenas que forman el recorrido.

La herramienta esencial para llevar a cabo todo este proceso se llama "KRPano". Se trata de una utilidad ligera y de alto rendimiento para la creación y visualización de imágenes panorámicas y recorridos virtuales interactivos. Este visor está disponible tanto en *Flash* como *HyperText Markup Language 5* (HTML5) y puede ser usado también en dispositivos móviles. KRPano está compuesto de dos partes diferenciadas:

- *KRPano Viewer*: Se trata del visor, que permite al usuario navegar por las escenas de forma intuitiva.
- *KRPano Tools*: Es un conjunto de pequeñas herramientas que ayudan a preparar las imágenes panorámicas que sube el usuario para ser vistas correctamente con la aplicación.

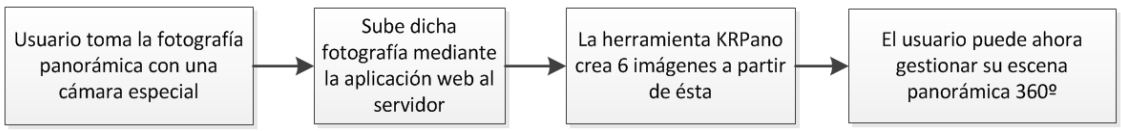

Figura 01: Diagrama del proceso general

Para mostrar al usuario las escenas, KRPano interpreta archivos de tipo *eXtensible Markup Language* (XML), los cuales contienen los parámetros u opciones necesarias configuradas, así como la definición de las seis imágenes que componen la escena (creadas por KRPano para un visionado correcto a partir de la que sube el usuario), *Hotspots* y las acciones asociadas a éstos (como, por ejemplo, cargar una nueva escena panorámica cuando se haga *click* en el punto).

La aplicación gira por tanto en torno a tres elementos principales:

- **Recorridos virtuales**: Formados por un conjunto de escenas panorámicas las cuales están conectadas mediante *Hotspots*, permitiendo así al usuario navegar entre éstas.
- **Escenas panorámicas**: Imágenes panorámicas 360º creadas por KRPano a partir de las fotografías tomadas con una cámara fotográfica especial por el usuario. Estas escenas pueden contener elementos *Hotspot*.
- ***Hotspots***: Se trata de elementos que pueden ser añadidos a la escena y que pueden realizar distintas funciones. Entre ellas, la más común es el salto a otra escena dentro de un mismo recorrido virtual pero también es posible añadir vídeos, elementos ampliables (por ejemplo, un cuadro colgado en una pared), globos de información, etc.

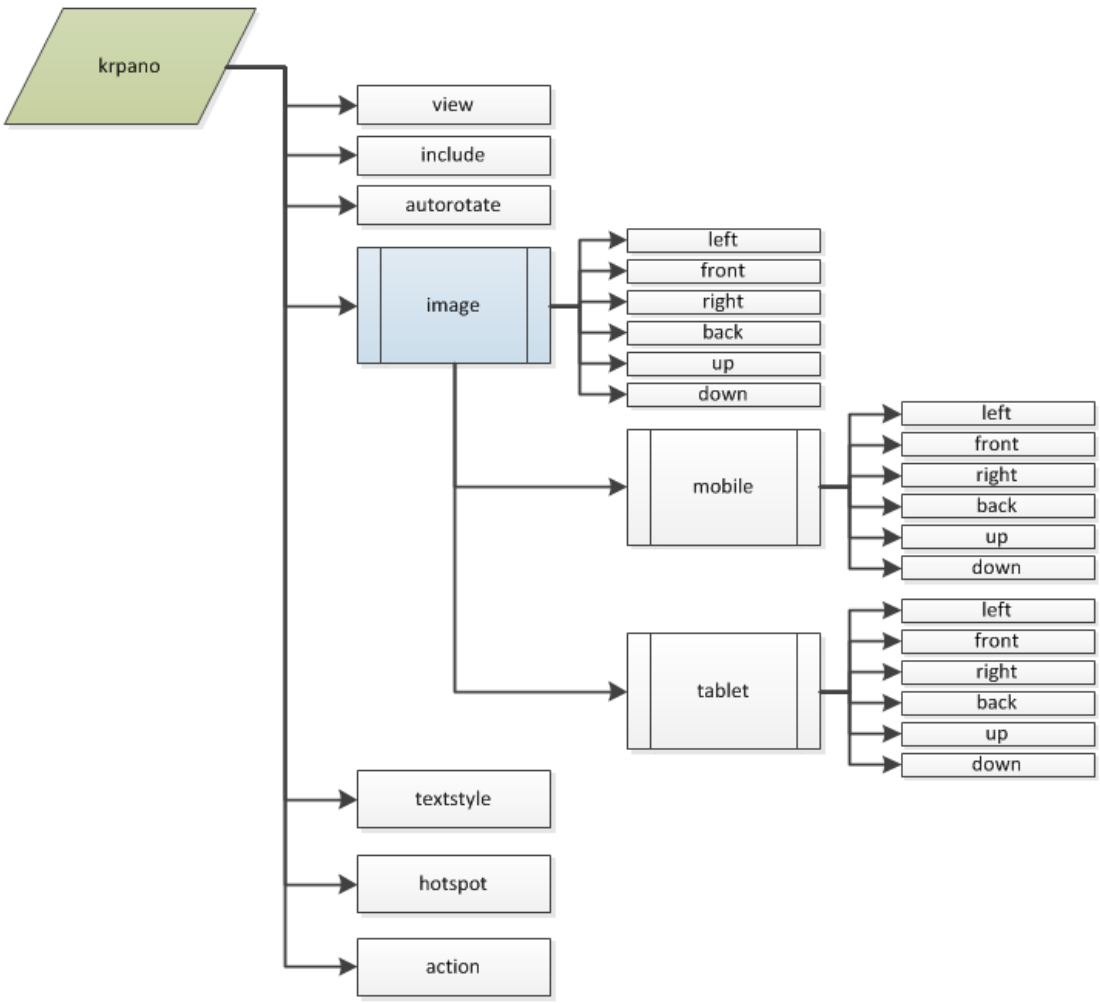

Figura 02: Ejemplo de la estructura del archivo XML que interpreta KRPano

Figura 03: Ejemplo oficial de *KRPano Viewer*

1.2 Estado de la cuestión

1.2.1 Contexto

La realización de este proyecto ha tenido lugar durante la estancia de prácticas en una empresa. El proyecto surge a partir de la necesidad de un cliente de contar con una herramienta que le permita incrementar la productividad de su trabajo. Antes de la realización de esta aplicación, el cliente tenía que contratar a un informático para crear y gestionar uno a uno y manualmente (puesto que no se contaba con una interfaz o aplicación específica para ello) los distintos archivos XML necesarios para que KRPano mostrase las escenas y elementos deseados. Debido a esto, la productividad que el cliente obtenía era muy baja, ya que conseguía finalizar una media de, tan solo, 5 recorridos por mes.

1.2.2 Punto de partida

Se parte de la siguiente idea principal: Automatizar el proceso de creación y gestión de los recorridos lo máximo posible mediante una aplicación web. Así mismo, se sabe que para la muestra y montaje de las escenas panorámicas y recorridos virtuales es esencial el uso de la herramienta KRPano.

Por otra parte, el cliente establece ciertos requisitos imprescindibles. Entre los más destacados se encuentran los siguientes:

- Posibilidad de descargar una versión *Offline* totalmente funcional de un recorrido virtual específico.
- Soporte para el visionado de los recorridos virtuales y escenas en dispositivos móviles.
- Soporte multilingüe (inglés, francés, alemán e italiano) en el visionado, tanto *Online* como *Offline*, de los recorridos y escenas.
- Contar con un mecanismo de gestión de clientes. Esto es necesario puesto que nuestro cliente vende su trabajo a terceros.
- En la creación y edición de recorridos debe requerirse, por lo menos:
    - Indicar un nombre.
    - Seleccionar, a partir de las escenas que lo componen, cuál será la principal. Es decir, la que se mostrará primero al visualizar el recorrido.
    - Elegir, desde una lista de localizaciones, la que corresponde al recorrido (por ejemplo, Valencia).
- En las escenas panorámicas tan solo es necesario requerir el nombre y la localización.
- Incluir una opción que permita marcar una escena como un punto de interés (*Point of Interest, POI*). Estos puntos deberán aparecer marcados en un mapa al pie del visor del recorrido virtual si este incluye una escena que tenga uno o más POI.
- Poder crear “ubicaciones”. Éstas deben estar formadas por un conjunto de recorridos virtuales y es necesario dar la posibilidad de indicar el recorrido por el que deben comenzar. Así mismo, se debe poder acceder a ellas a través de una dirección *Uniform Resource Locator* (URL).

La aplicación final debe contar, además, con todas las funciones necesarias para la gestión, creación y edición de los distintos elementos. Por ejemplo: Listas de elementos con funciones de filtrado según distintos parámetros, borrado de escenas y recorridos, etc. Para la presentación de la aplicación junto a las funciones anteriores, se creará un pequeño *Content Management System* (CMS) que será realizado mediante la herramienta *MODx*, la cual permite crear tu propio sistema de forma completamente personalizada.

1.3 Alcance

Este proyecto documenta las fases de diseño e implementación de la funcionalidad necesaria para cumplir los requisitos del cliente. Para ello, se cubre, desde el diseño e implementación de la base de datos que da la persistencia a la aplicación, hasta la implementación de la funcionalidad, pasando por la estructuración en capas del código. Sin embargo, no se adjunta información sobre la creación del CMS mediante la herramienta MODx, puesto que fue realizado por otra persona con experiencia previa y preferimos enfocar el proyecto a la parte funcional de la aplicación.

1.4 Objetivos

A partir de los requisitos del cliente, nos proponemos los siguientes objetivos principales:

- Diseñar e implementar la base de datos de forma que aporte persistencia a todos los elementos necesarios para el correcto funcionamiento de la aplicación final.
- Crear una estructura en tres capas para el desarrollo de la lógica y funcionalidad de la aplicación. Esto nos facilita el desarrollo, sobre todo en el caso en que haya que realizar cambios, ya que éstos estarán centralizados en una de las capas:
    - Capa de negocio.
    - Capa de datos/persistencia.
    - Capa de presentación.
- Crear las clases y funciones necesarias para cumplir los requisitos principales impuestos por el cliente, así como la funcionalidad esencial para la gestión de los recorridos, escenas, etc.

De forma secundaria y una vez alcanzados los objetivos principales, nos gustaría mejorar en todo lo posible la gestión de escenas panorámicas y recorridos virtuales mediante la adición de pequeñas funcionalidades adicionales que permitan una personalización y configuración aun más amplia y fácil. Algunas ideas:

- Escenas panorámicas:
    - Incluir la posibilidad de poder introducir un nombre distinto para la escena, que será el que se mostrará a los usuarios. Este nombre podrá ser multilingüe (alemán, inglés, francés e italiano).
    - Añadir una pequeña vista previa de la escena al formulario, permitiendo al usuario controlar la misma (rotar alrededor de la escena), de forma que la vista que deje definida el usuario sea el punto de entrada. Este punto es la vista inicial que se mostrará al cargar la escena. De esta forma el usuario puede apuntar inicialmente a una zona de especial interés o simplemente mostrar un espacio estéticamente agradable para dar una buena primera impresión.
    - Permitir escribir la dirección física a la que pertenece la escena.
    - Dar la posibilidad de categorizar la escena según los tipos de edificios u establecimientos que aparecen en la misma. Por ejemplo, algunas de las categorías pueden ser: Restaurantes, hoteles, bares, ópticas, garajes, etc.
        - Sólo escenas que estén clasificadas como POI podrán ser categorizadas.

- Recorridos virtuales:
    - Permitir seleccionar un cliente para relacionarlo con el recorrido. El cliente estará definido en la base de datos previamente gracias a la gestión de clientes que se va a implementar.
    - Crear un sistema de vista preliminar. Este sistema permitirá, tanto añadir una imagen estática preliminar que identifique al recorrido, como mostrar en una ventana flotante una versión completamente funcional de las escenas que componen el recorrido.
    - Añadir una entrada informativa en el formulario que indique el número de escenas que forman el recorrido actualmente.
    - Permitir clasificar un recorrido según tipo. Esto puede ser necesario en el caso de que las empresas con que nuestro cliente trabaja quisieran personalizar sus recorridos de alguna forma (por ejemplo, distintas imágenes para los *Hotspot*).
    - Proveer los enlaces relacionados con el recorrido: Enlaces para visionarlo, enlaces de descarga *Offline*, etc.

- Comunes a ambos:
    - Posibilitar la activación y desactivación de escenas y recorridos concretos.

En definitiva, éstas son solo algunas ideas iniciales para mejorar la experiencia del usuario, al finalizar la aplicación algunas de estas ideas se implementarán y otras tal vez no. Además, es muy posible que el cliente añada requisitos o nos de ideas adicionales a lo largo del desarrollo por lo que el resultado final puede llegar a ser distinto del planteado inicialmente.

1.5 Justificación

Se trata de un proyecto complejo, ya no solo por su envergadura sino también por el hecho de depender de un cliente en todo momento. Al tratarse de una situación real, los requisitos pueden cambiar en cualquier instante y sin previo aviso por lo que debemos adaptarnos a los deseos del cliente en todo momento, con lo que ello implica.

Parece por tanto una muy buena oportunidad para llevar a cabo este proyecto ya que no solo nos permite mejorar y expandir nuestros conocimientos informáticos durante el desarrollo del mismo sino que también proporciona una importante experiencia laboral para el futuro.

2. Desarrollo

2.1 Herramientas, recursos empleados y estructura del desarrollo

Para llevar a cabo un proyecto de esta envergadura correctamente es esencial buscar y utilizar las herramientas más adecuadas. El proyecto está desarrollado principalmente en PHP pero también usamos, lógicamente, HTML, CSS y JavaScript con su biblioteca JQuery. Teniendo esto en cuenta, a continuación exponemos las herramientas que usamos en este proyecto:

- NetBeans IDE 8.0: Se trata de un entorno de desarrollo integrado (*Integrated Development Environment*, IDE) gratuito y de código abierto. Aunque en principio cuenta con soporte para varios lenguajes de programación, nosotros utilizamos la versión *PHP: Hypertext Preprocessor* (PHP), que es más ligera. NetBeans nos parece un entorno de desarrollo perfecto para este proyecto debido a que su completa interfaz, herramientas y ayuda a la hora de programar facilitan el avance del desarrollo del mismo.

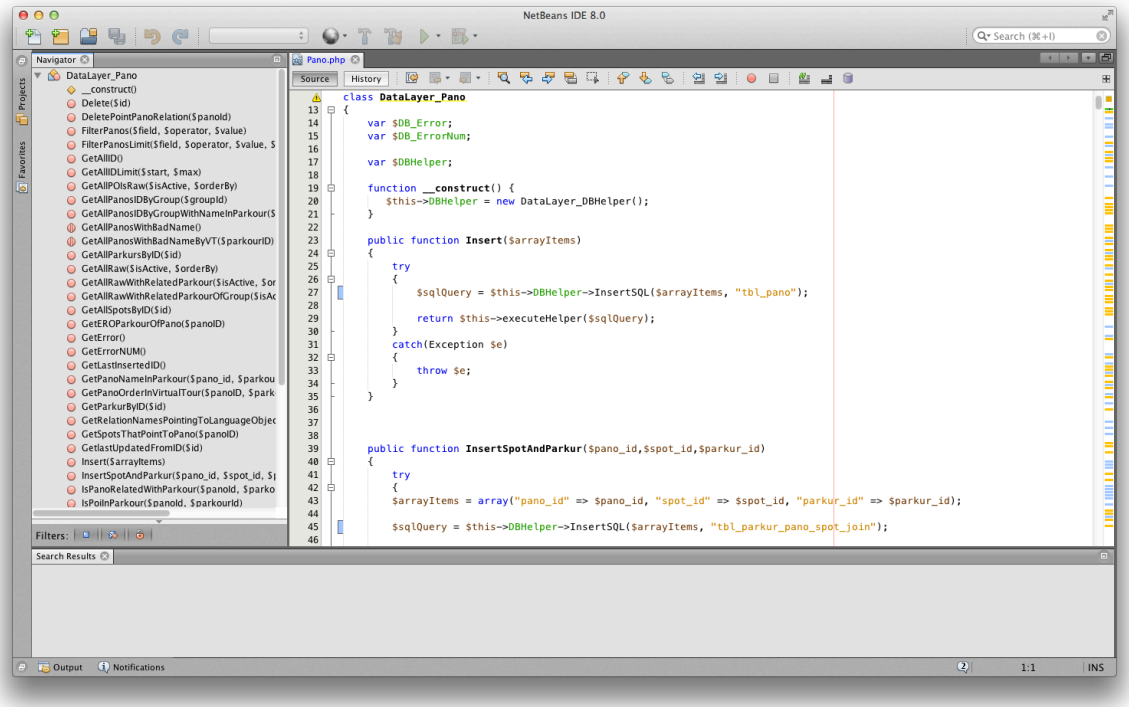

Figura 04: Herramienta de navegación de una clase en NetBeans

- Xdebug: Es una extensión para PHP que es esencial en este y cualquier otro proyecto de cierta complejidad puesto que facilita en gran medida la búsqueda y resolución de errores. Como su nombre indica, se trata de una herramienta de *Debug* (depurado). En nuestro caso, hemos aprovechado que puede integrarse con nuestro entorno de desarrollo NetBeans (la cual es otra de las razones para utilizarlo).
- XAMPP: Es una herramienta muy popular que proporciona lo esencial para trabajar de forma local a la hora de desarrollar páginas y aplicaciones web con PHP. En nuestro caso, hemos hecho uso tanto de su servidor de base de datos MySQL como de su servidor web Apache.

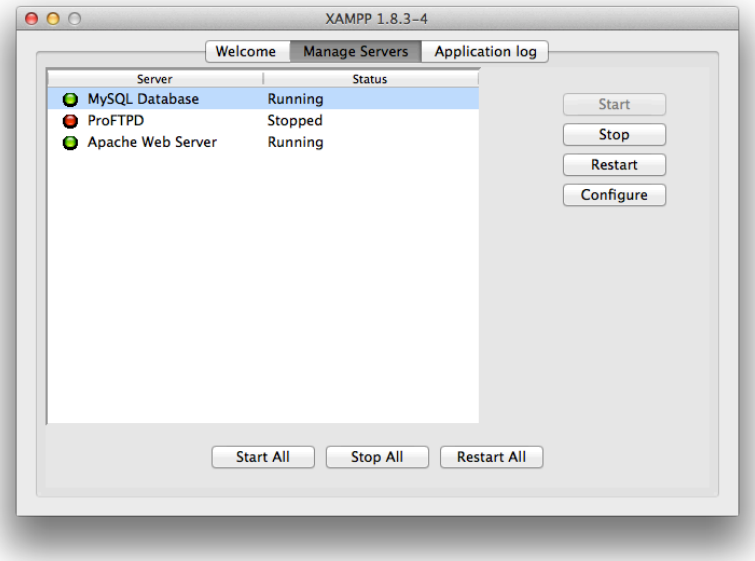

Figura 05: XAMPP y sus servidores en funcionamiento

- MySQL Workbench: Se trata de una aplicación de escritorio muy completa puesto que proporciona tanto herramientas de administración de bases de datos como de diseño y modelado de éstas. En nuestro caso, la utilizamos para trabajar de forma local junto al servidor MySQL proporcionado por XAMPP para diseñar, modelar, crear y administrar las distintas tablas que forman nuestra base de datos.

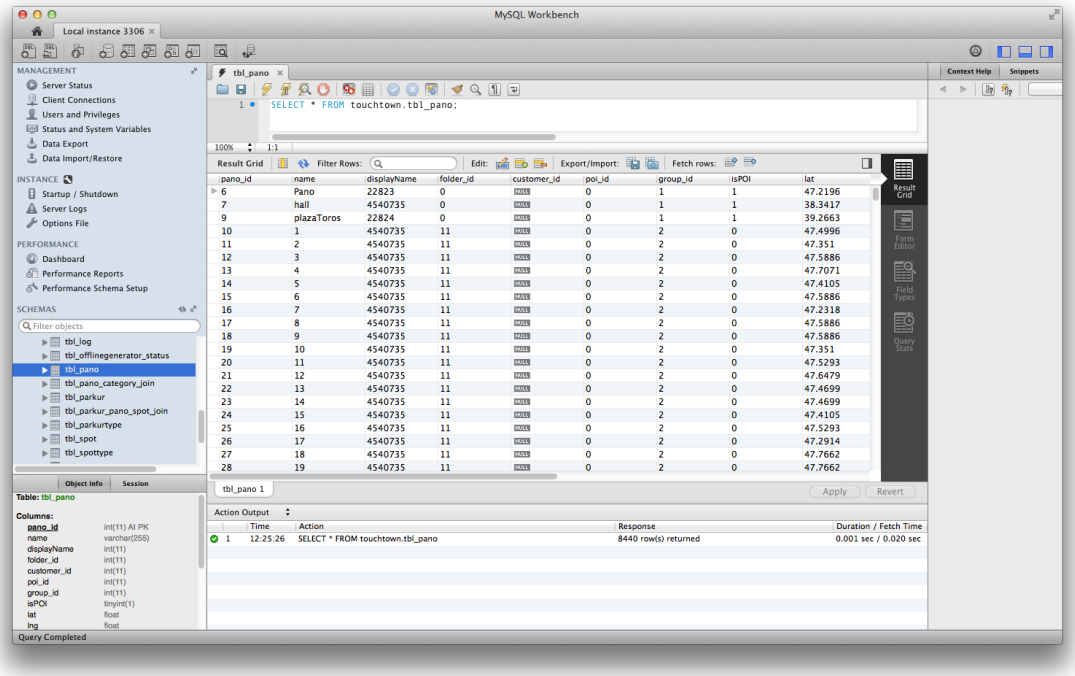

Figura 06: Administración de la base de datos con MySQL Workbench

- Versions: Es un cliente *Subversion* (SVN) para el control y gestión de versiones durante el desarrollo de *Software*. Esta aplicación en concreto es de pago (aunque cuenta con un periodo de prueba) y tan solo está disponible para el sistema operativo Mac OS X; sin embargo, hay gran cantidad de herramientas SVN disponibles para todos los sistemas operativos como, por ejemplo, *TortoiseSVN* o *Git.*

  El control de versiones es importante en este tipo de proyectos, especialmente si más de una persona trabaja en el mismo, ya que así se evitan pérdidas de información a la hora de juntar código proveniente de distintas fuentes.
- KRPano: Como se indicó en la introducción, la herramienta imprescindible y sobre la que se basa todo el proyecto es KRPano. Esta herramienta nos proporciona tanto el visor para las panorámicas (*KRPano Viewer*) como las utilidades para convertir las imágenes subidas por el usuario en dichas escenas panorámicas compatibles con el visor (*KRPano Tools*). Así mismo, también permite realizar conversión de tamaño de imágenes, lo cual aprovechamos para crear una versión de menor resolución y por tanto menos pesada de las mismas para los dispositivos móviles.

- TCPDF: Se trata de una clase para PHP que permite crear archivos *Portable Document Format* (PDF) dinámicamente. Es muy popular debido a su facilidad de uso y versatilidad.
- FileZilla: Es el cliente FTP mediante el cual nos conectamos al servidor real donde se alojan los archivos finales de la aplicación. FileZilla es gratuito y fácil de usar pero existen innumerables clientes FTP disponibles.
- Microsoft Office Visio: Utilizamos Visio para la creación de diagramas como los de las figuras 01 y 02. Es una herramienta muy útil para esquematizar ideas y transmitir grandes cantidades de información de forma visual.
- OmniGraffle Pro: Alternativa a Office Visio para el sistema operativo Mac OS X.

En cuanto a la estructura del desarrollo, contamos con tres partes diferenciadas:

- Entorno de desarrollo: Es nuestro propio equipo, donde trabajamos de forma local y realizamos todo el desarrollo con las herramientas citadas anteriormente.
- Entorno intermedio o de pruebas: Cuenta con un servidor de características lo más similar posibles al entorno de producción final de forma que, tras realizar satisfactoriamente las pruebas en el entorno de desarrollo local y antes de subir el proyecto al entorno de producción, se prueban en este para asegurar aun más el correcto funcionamiento en el entorno final.
- Entorno de producción: Se trata del servidor real donde se alojan los archivos de la aplicación, y es al que se accede al utilizar la misma a través del navegador web.

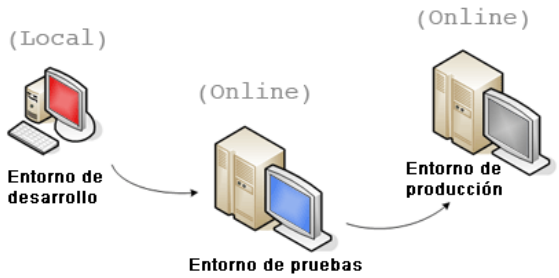

Figura 07: Estructura del desarrollo

Este tipo de estructura es ideal, ya que reduce al máximo el número de errores en el servidor real. De esta forma, el desarrollo se lleva a cabo de forma transparente para los usuarios y se evitan tiempos de mantenimiento o caídas del servicio innecesarias.

2.2 Estructura de la aplicación

Como hemos comentado brevemente en la introducción, para facilitar en gran medida el desarrollo del proyecto hemos decidido estructurar la lógica en tres capas. Esto es conocido como programación por capas. En este tipo de programación de la arquitectura cliente-servidor el objetivo principal es separar la lógica de las distintas capas, de forma que las tres sean completamente independientes. Esto nos proporciona una gran libertad a la hora de realizar cambios ya que, puesto que éstos no afectan a las demás capas, basta con modificar tan solo la que sea necesario. Otras ventajas pueden ser las siguientes:

- Aislado de la lógica de la aplicación en componentes separados, lo cual aumenta en gran medida la facilidad a la hora de realizar cambios.
- Posibilidad de distribuir las capas en diferentes máquinas o procesos.
- Posibilidad de desarrollar en paralelo: Muy útil para proyectos grandes que cuenten con numerosos desarrolladores.
- Distribución de recursos a cada una de las capas de forma independiente.
- Reutilización de código entre proyectos.

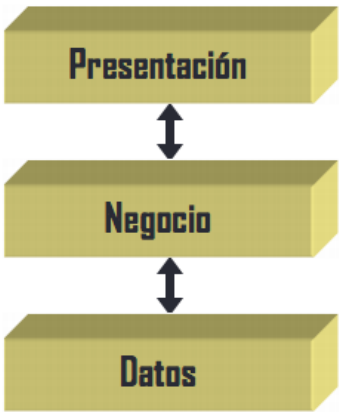

Figura 08: Arquitectura en tres capas

2.2.1 Capa de negocio

La capa de negocio o lógica de la aplicación es la parte del programa que implementa las reglas que determinan cómo se puede crear, mostrar, modificar y guardar la información. Generalmente, las clases de la capa de negocio están

relacionadas a través de sus funciones con la capa de datos puesto que hacen uso de ella para obtener o manipular la información de la base de datos.

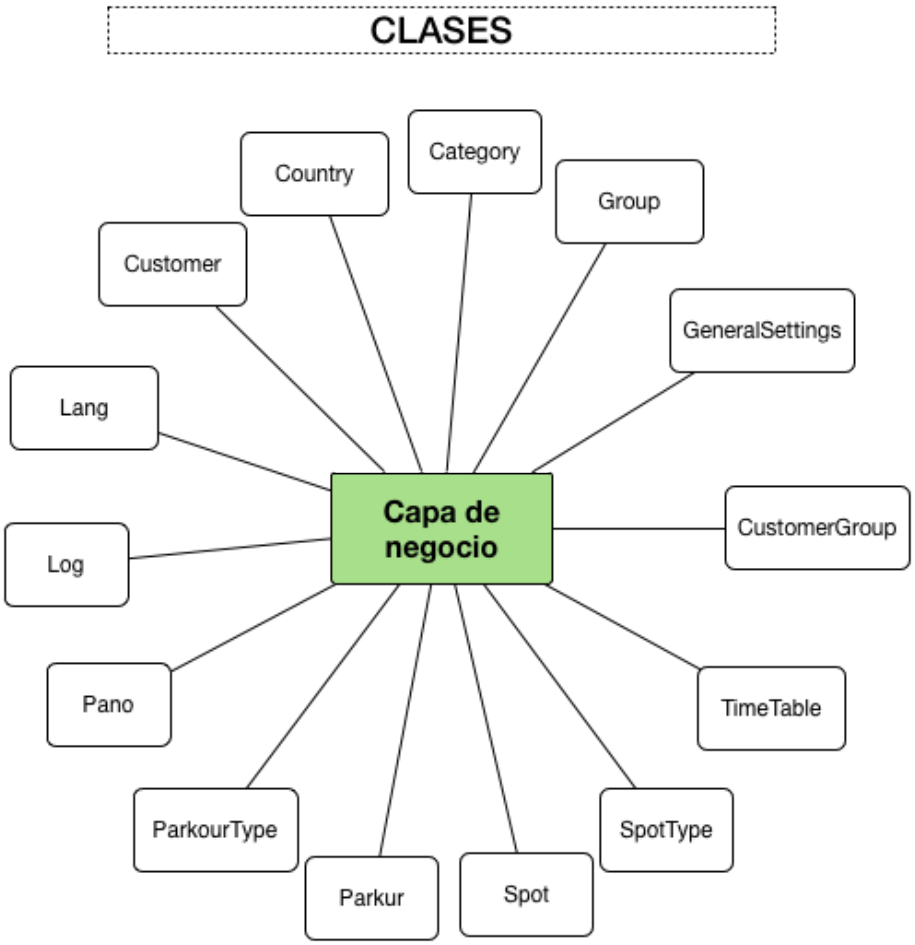

Figura 09: Clases que forman la capa de negocio

A continuación, exponemos detalladamente las clases que forman nuestra capa de negocio, así como la función que desempeñan en la aplicación final:

- Category: Al igual que las demás clases de la capa de negocio, contiene la definición de todas las funciones que necesitamos para obtener y modificar los datos. Si éstas necesitan acceder a la información de la base de datos, lo consiguen instanciando la clase correspondiente de la capa de datos (que normalmente tiene el mismo nombre que la de la capa de negocio) y obteniendo el resultado devuelto por la función correspondiente de la misma.
  Category es una sección accesible a través del CMS que nos permite gestionar las categorías en las que una escena panorámica marcada como POI se puede

clasificar por lo que, entre las distintas funcionalidades que ofrece, encontramos: Añadir, listar, editar y eliminar categorías.

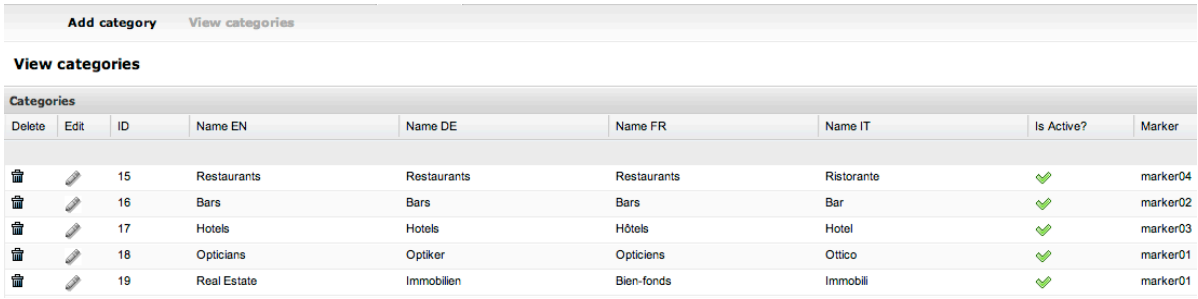

Figura 10: Lista de categorías en el CMS

- Country: Al igual que *Category*, *Country* es una sección accesible desde el CMS y cuenta también, además de con las funciones usuales (añadir, listar, editar y eliminar países), con un filtrado según el nombre del país. En la aplicación final utilizamos los países, principalmente, en dos situaciones:
    - A la hora de crear y editar *Locations* (ubicaciones) para asociarles el país en el que se encuentran.
    - A la hora de crear y editar *Customers* (clientes) para indicar el país del que provienen.

  Estos usos implican que las clases de la capa de negocio correspondientes a las que utilizan ésta deben instanciarla para obtener los datos necesarios. Por ejemplo, la clase *Customer* contiene una función llamada *GetCountry* que realiza esta acción, asignando el valor obtenido (si es que existe) al parámetro correspondiente de su objeto cliente; esto ocurre por tanto con todas las clases que se relacionan entre sí y manejan datos de la base de datos.

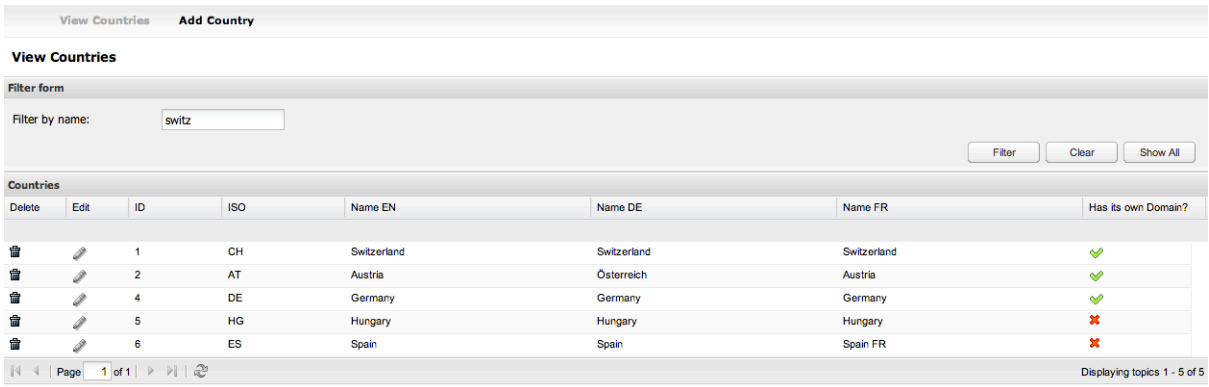

Figura 11: Lista de países en el CMS

- Customer: Es otra de las categorías del CMS y, al igual que la lista de países, también cuenta con un filtrado. En este caso este es algo más complejo ya que permitimos hacerlo según cuatro parámetros distintos: *Title, Address, Country* y *City*.

  Puesto que nuestro cliente vende su trabajo a otras personas (los clientes listados en la base de datos), en la creación y edición de recorridos virtuales debemos proporcionar la posibilidad de seleccionar un cliente entre todos los existentes. Por tanto, tiene que existir una función en la clase correspondiente a los recorridos que instancie a ésta clase para obtener la lista de clientes existentes. A su vez, ésta clase (de la capa de negocio), instancia a la clase clientes de la capa de datos para obtener o modificar la información necesaria de la base de datos.

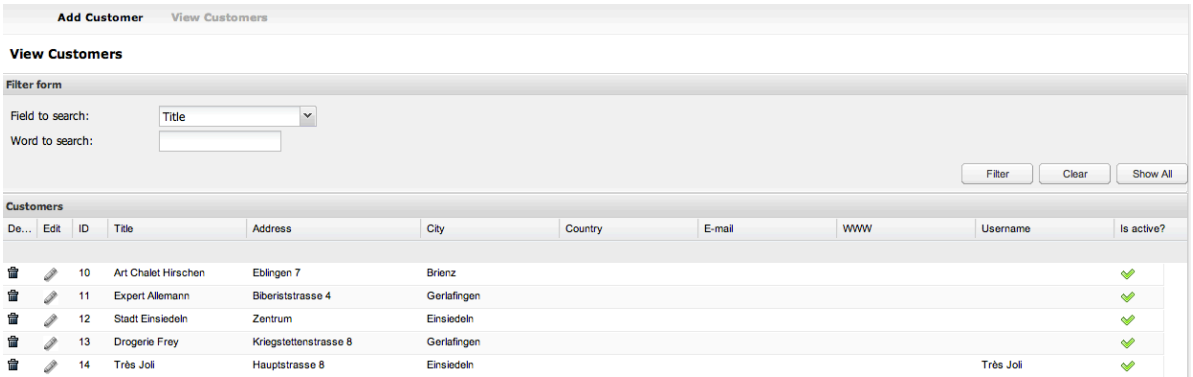

Figura 12: Lista de clientes en el CMS

- CustomerGroup: Implementa la funcionalidad de un campo obligatorio en la creación y edición de clientes que nos permite asociarlos a un grupo predefinido en la base de datos. La capa de negocio se encarga de implementar las funciones necesarias para la gestión de éstos. Al igual que en las demás clases de la capa de negocio, las funciones de ésta instancian a la clase correspondiente de la capa de datos y guardan en un objeto *array* los datos resultantes de aplicar las funciones de la misma.

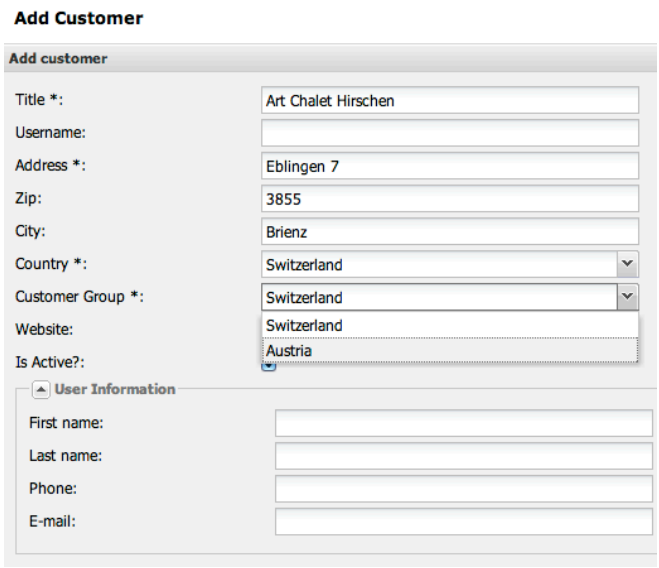

Figura 13: Selección del grupo al que pertenece un cliente

- GeneralSettings: Es otra de las pestañas del CMS. Ésta incluye tres opciones que nos permiten modificar el comportamiento del visor de KRPano de forma global. Una de ellas es *Autorotator* y, si está marcada, sustituye al valor individual de cada recorrido (es decir, se activa para todos sin importar el valor que tengan definido). Como su nombre indica, si está activado, cuando se visualicen las escenas de un recorrido éstas rotarán automáticamente sin que el usuario necesite interactuar con ellas.

  En cuanto a las otras opciones, "*Show upper left logo?*" activa o desactiva un logo que podemos mostrar en la esquina izquierda superior del visor de escenas de KRPano y "*Seconds in autoplay for each panorama*" configura la cantidad de tiempo que se muestra cada escena en dicho modo.

  Una vez más, la capa de negocio se encarga de obtener los datos necesarios para crear, cargar y actualizar las opciones haciendo uso de las clases y funciones de la capa de datos.

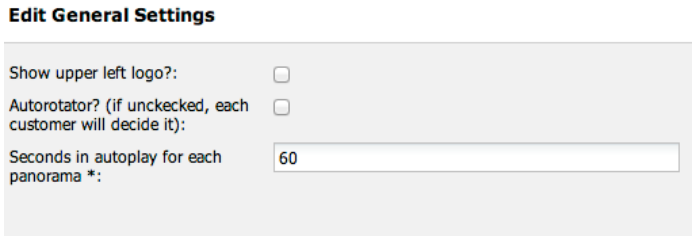

Figura 14: Opciones generales de la aplicación

- Group: Existe la posibilidad de crear *locations* (ubicaciones). Éstas están formadas por un conjunto de recorridos virtuales de forma que, por ejemplo, puede crearse una ubicación llamada "Valencia" que esté compuesta por todos los recorridos que contienen escenas de dicha ciudad. Además, al crear o editar una ubicación es posible indicar el recorrido por el cual queremos que comience.

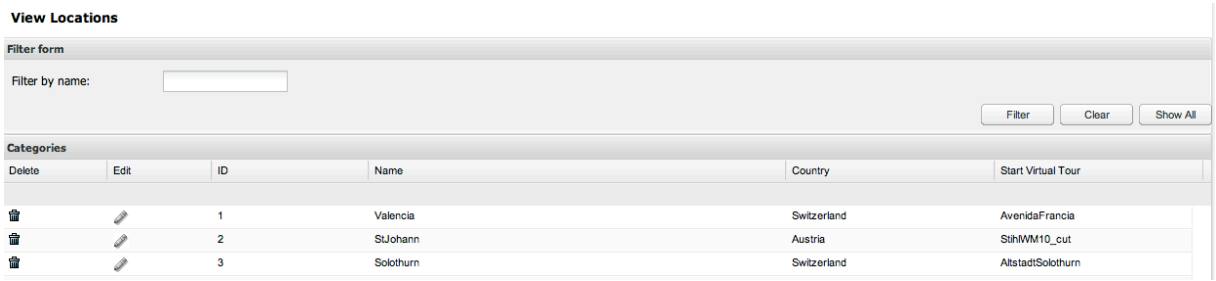

Figura 15: Lista de ubicaciones

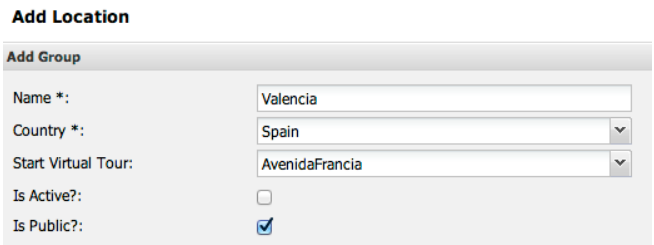

Figura 16: Creación de ubicaciones

- Lang: Esta clase es muy importante para conseguir que la aplicación esté disponible en varios idiomas puesto que contiene la traducción en cuatro lenguas (inglés, alemán, francés e italiano) de todos los textos que así lo necesiten. La capa de negocio controla la carga, creación y edición de estos textos. Por ejemplo, en la creación y edición de escenas es posible indicar el nombre de la misma en estos cuatro idiomas. Cuando introducimos y enviamos los datos a través del formulario, la capa de negocio utiliza su función de guardado (cuando es una nueva escena) o de actualización (cuando se está editando una existente) para instanciar la clase correspondiente de la capa de datos y utilizar sus funciones para acceder a la base de datos y realizar los cambios con los datos que hemos introducido.

- Log: Esta clase es simple pero muy importante. Básicamente, nos proporciona información sobre los eventos que tienen lugar durante el uso de la aplicación por los usuarios. La información obtenida se compone de:
    - Mensaje informativo: Qué ha ocurrido.
    - Origen: Indica qué ha provocado el evento o dónde ha ocurrido. Puede ser una clase, función, un paso o elemento de un formulario, etc.
    - *TIMESTAMP*: Fecha y hora en que ha ocurrido el evento. Este campo es rellenado automáticamente por la base de datos al crearse la entrada en la misma.
    - *Severity*: Se trata de un número (entre 0 y 4, incluidos) que indica el tipo de evento, siendo 0 el más importante. Normalmente, en la práctica los más comunes son los tipo 0, 2 y 4:
        - 0: Indica que el evento es un error que necesita ser solucionado.
        - 1: Significa que el evento es de tipo *Warning* (aviso).
        - 2: Contiene información sobre la ejecución.
        - 3: Significa *Tracing* (similar al depurado, sirve para seguir la ejecución).
        - 4: Indica *Debug* (mensajes de depurado, para ayudar a encontrar fallos durante la programación).

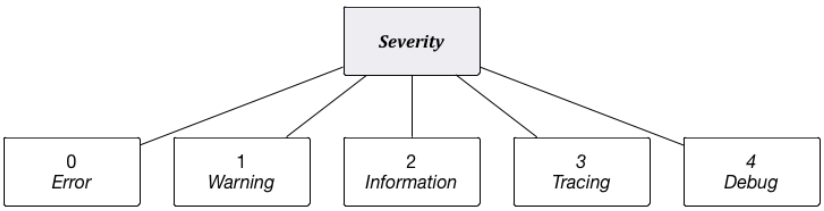

Figura 17: Tipos de evento

Esta clase contiene una función distinta para cada nivel, siendo la única diferencia entre ellas el valor del parámetro *Severity* pasado en la llamada a la función de inserción de la clase de la capa de datos instanciada.

- Pano: Es una de las tres clases más importantes (junto a *Parkur* y *Spot*) puesto que en ella implementamos la funcionalidad necesaria para la gestión de las escenas panorámicas. Cuenta por tanto con un gran número de funciones, siendo algunas simples variaciones de otras debido a la gran cantidad de

parámetros que existen y todas las relaciones con las que cuenta la tabla en la base de datos.

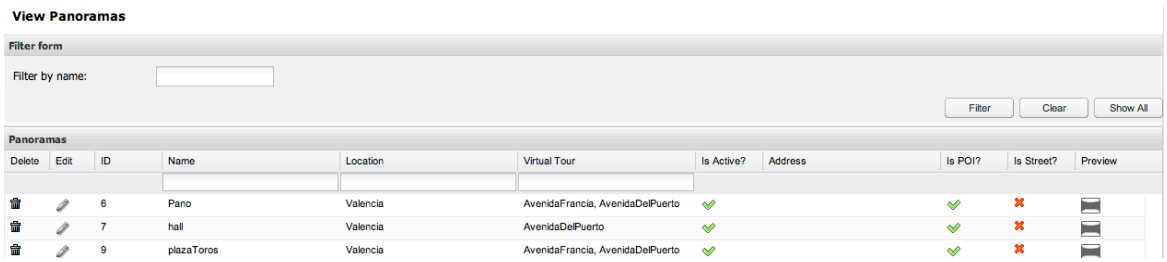

Figura 18: Lista de escenas

Para crear una nueva escena, el usuario debe subir mediante el formulario al servidor la imagen panorámica que ha tomado previamente. Este también le permite seleccionar si quiere incluir la escena en un nuevo recorrido virtual o bien en uno ya existente y, dependiendo de lo que elija, debe introducir una serie de datos distintos:

- Inclusión en un nuevo recorrido: Es necesario seleccionar la ubicación a la que se desea que pertenezca el recorrido así como darle un nombre a este.
- Inclusión en un recorrido existente: Para encontrar el recorrido al que se quiere que pertenezca, se debe seleccionar primero la ubicación y, tras esto, elegir entre los recorridos que ésta contiene. Por tanto, si queremos que nuestra nueva escena pertenezca al recorrido “Avenida de Francia”, primero debemos seleccionar la ubicación “Valencia”.

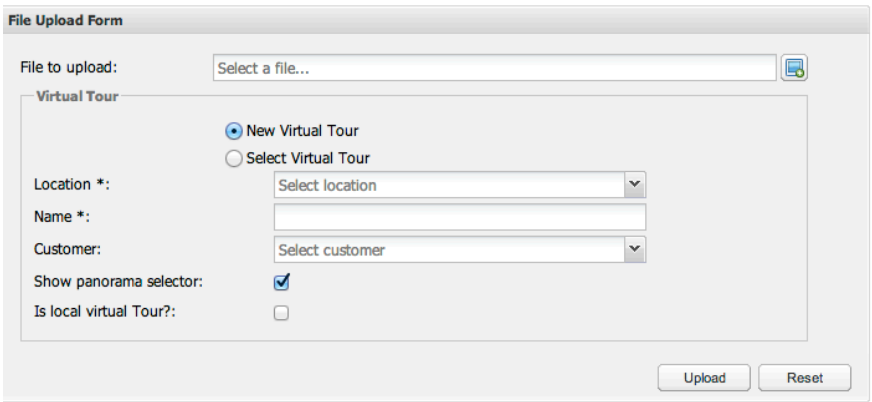

Figura 19: Creación de una escena e inclusión en un nuevo recorrido virtual

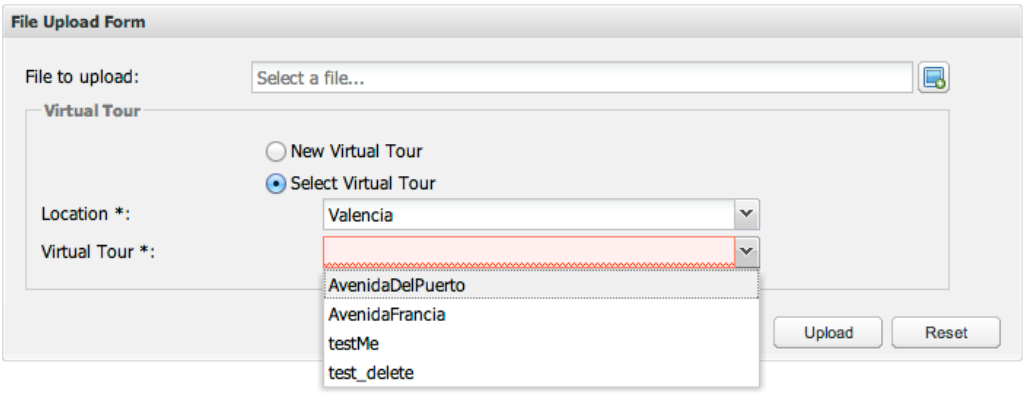

Figura 20: Creación de una escena e inclusión en un recorrido existente

Una vez más, todas las funciones que necesitan obtener datos de la base de datos lo consiguen instanciando y usando la clase de la capa de datos correspondiente. Algunas de las más destacadas nos permiten: obtener todos los *Hotspots* de una escena, recuperar los identificadores de los recorridos en que se incluye la escena, obtener una lista de los *Hotspot* que apuntan a esta escena, comprobar si está relacionada con un recorrido, …

- Parkur: Se trata de otra de las clases esenciales, ya que implementa la gestión de los recorridos virtuales. Al igual que la clase *Pano*, ésta cuenta con muchas funciones debido a la alta cantidad de relaciones con que cuenta la tabla en la base de datos.

  Como se puede ver (*Fig. 21*), cuando accedemos al listado de recorridos se nos presentan varias operaciones posibles para cada uno. En concreto, y a parte de las acciones comunes de gestión, encontramos un par de opciones que no existen en los demás listados: obtener URLs y descargar versión *Offline* (en varios idiomas). La primera acción abre una página en una nueva ventana que nos muestra los distintos enlaces con los que acceder al recorrido y a sus escenas individuales, así como el código necesario para insertar el mismo en una página. Las demás acciones nos permiten descargar un archivo comprimido que contiene todos los archivos imprescindibles para visualizar los recorridos y escenas sin necesidad de estar conectado a internet (modo *Offline*); cada una de estas acciones descarga las versiones de un idioma distinto.

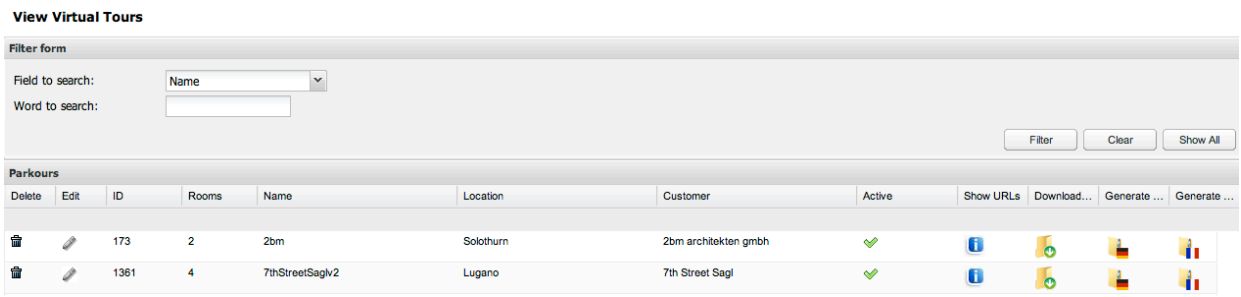

Figura 21: Lista de recorridos virtuales

Algunas de las funciones más destacadas son: añadir una escena a un recorrido, borrar un recorrido, filtrar según una propiedad determinada, obtener las escenas que forman cierto recorrido, obtener todos los recorridos de una categoría, …

- ParkourType: Como su nombre indica, esta clase gestiona los tipos de recorridos virtuales. Como explicamos previamente, los tipos de recorrido son necesarios debido a que nuestro cliente vende su trabajo a terceros y éstos pueden querer personalizar los recorridos que compran de algún modo.
  Puesto que en este caso solo necesitamos obtener la lista de tipos de recorridos para poder seleccionar uno durante la creación o edición, la clase de la capa de negocio únicamente cuenta con las funciones de carga y obtención. En concreto, en cuanto a este último, contamos con tres variantes: obtener todos, según identificador, y una tercera que obtiene el nombre de archivo dependiendo del tipo de recorrido. Ésta última es necesaria puesto que al generar la versión *Offline* de un recorrido se crea un archivo HTML *Index*, el cual tiene un nombre distinto según el tipo del recorrido.

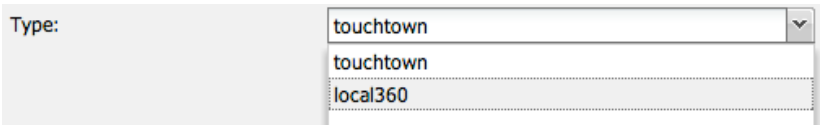

Figura 22: Selección de tipo de recorrido durante la creación u edición del mismo

- Spot: Es la tercera de las clases más importantes e implementa la funcionalidad necesaria para la gestión de los *Hotspots* en las escenas

panorámicas. La clase de la capa de negocio incluye varias funciones, entre las que se encuentran: crear, cargar, borrar y añadir *Hotspots* a una escena, obtener la etiqueta asociada al mismo (el texto descriptivo que puede aparecer cuando ponemos el ratón encima del punto), comprobar si pertenece a una escena concreta, etc.

Para añadir puntos a una escena o gestionar los existentes es necesario editar la misma (o crearla en el caso de que sea una escena nueva). Dentro del formulario contamos con un apartado muy importante donde se muestra la escena con una barra superior desde la que podemos seleccionar el tipo de punto a añadir (*Fig. 23*).

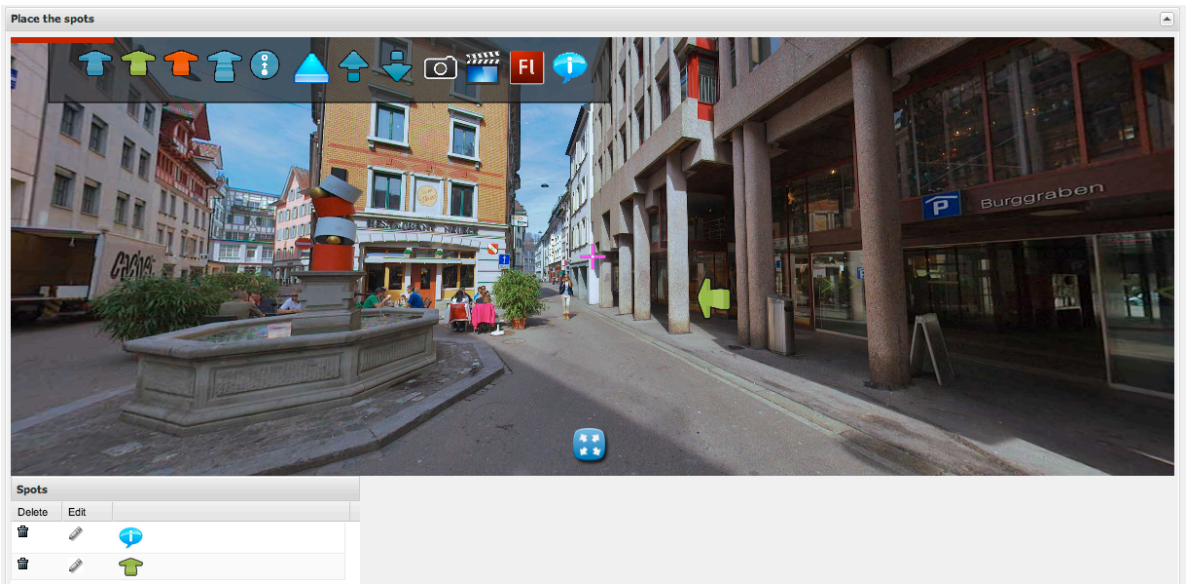

Figura 23: Gestión de los *Hotspots* de una escena panorámica

En el ejemplo, vemos que la escena seleccionada cuenta actualmente con dos puntos ya creados. Si editamos, por ejemplo, el segundo punto, nos aparecen todas las propiedades editables a la derecha (*Fig. 24*). Cabe destacar que las propiedades varían dependiendo del tipo de punto que añadamos. Por ejemplo, si tratamos de editar el otro (el globo de información), vemos que nos permite añadir todo tipo de información sobre una tienda o negocio (*Fig. 25*); este tipo de puntos se describen de forma más detallada en la explicación de la clase *Timetable*.

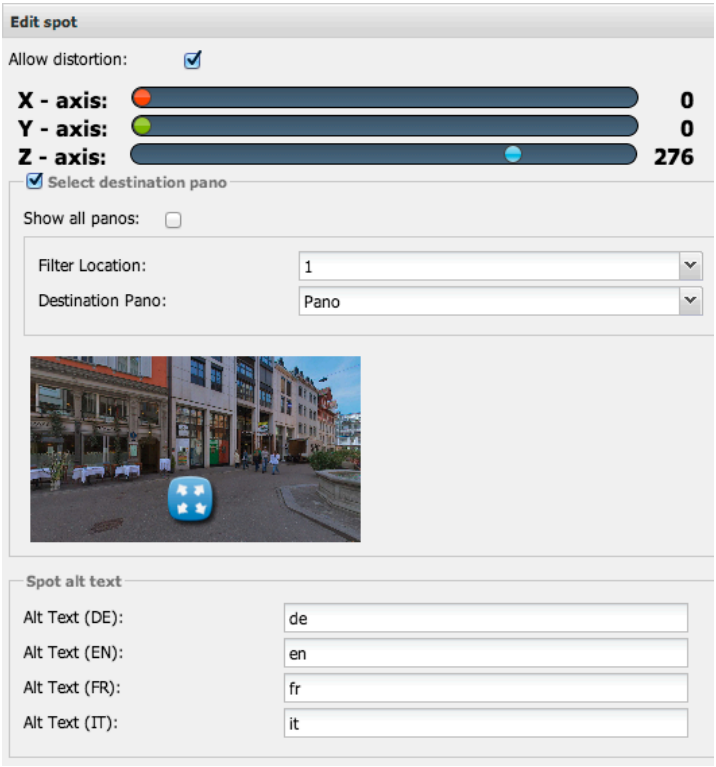

Figura 24: Parte de las propiedades de un punto tipo flecha

- SpotType: Esta clase contiene las funciones necesarias para obtener los datos de la tabla de la base de datos de mismo nombre haciendo uso de la clase correspondiente de la capa de datos y sus funciones. Concretamente, cuenta con tan solo dos: *Load* para cargar los datos a partir de un identificador y *GetTypeID* para obtener el identificador correspondiente al tipo de punto pasado. Algunos de los tipos de puntos son: *swf*, *picture*, *video*, *arrow*, *audio*, *info*, …

- TimeTable: Como hemos comentado en la explicación de la clase *Spot*, esta clase proporciona la funcionalidad para los *Hotspot* tipo globo de información. Al comienzo del proyecto, este tipo de puntos proporcionaban la posibilidad de añadir los horarios de apertura y cierre de un comercio (de ahí el nombre de la clase) pero conforme el proyecto estaba más avanzado, el cliente desechó esta idea debido a que esa información ya la proporcionaban las empresas a las que él vende el producto en las páginas donde éstas insertan los recorridos. Por tanto, actualmente, los globos pueden proporcionar más información,

como, por ejemplo: nombre de la tienda, dirección física, ciudad, teléfono, dirección de correo electrónico, página web, página de *Facebook*, etc.

Al igual que las demás, la clase cuenta con las funciones necesarias para poder crear, editar, cargar y guardar la información de este tipo de puntos en un *array*, el cual contiene los distintos parámetros correspondientes a cada campo. Este *array* es usado por las funciones de la clase de la capa de datos instanciada para realizar la acción correspondiente en la base de datos: añadir una entrada, obtener los datos de una existente, o modificarla y guardar los resultados en el mismo.

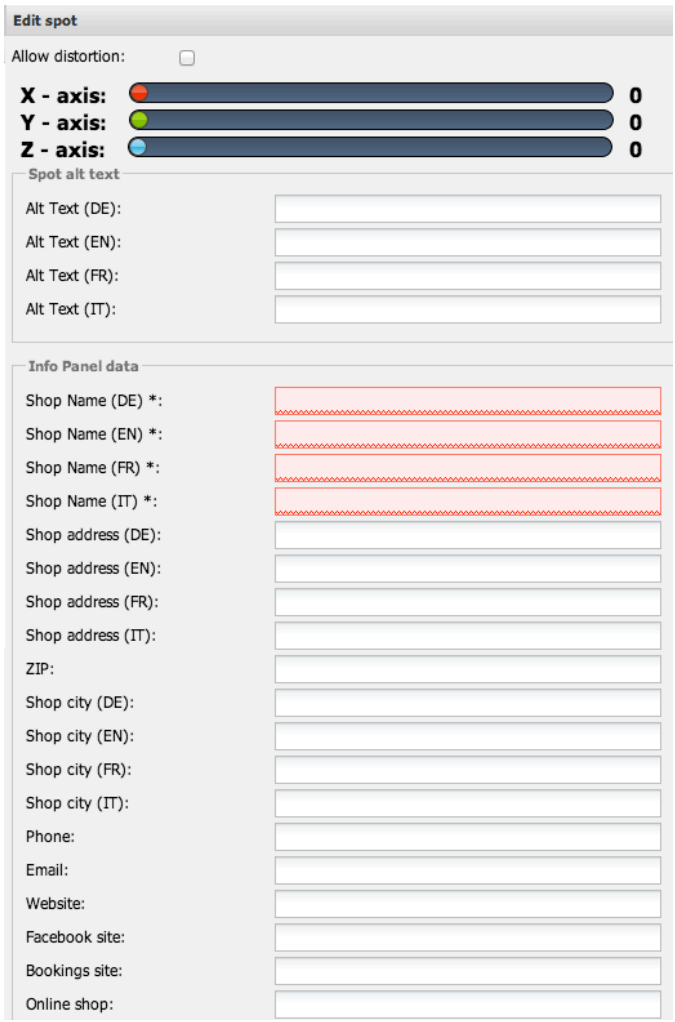

Figura 25: Parte de las propiedades de un punto tipo globo de información

En conclusión, el funcionamiento general de todas las clases de la capa de negocio consiste en:

1. Declaramos las variables correspondientes a los distintos parámetros que se manipulan (normalmente equivalen a los campos de la tabla de la base de datos).
2. Creamos una variable que contenga un *array* asociativo, correspondiendo cada posición del mismo al nombre y valor de uno de los parámetros declarados.
3. Los valores son obtenidos y guardados en el *array* mediante las distintas funciones que se implementan. Por ejemplo, una función que carga una escena o recorrido de la base de datos a partir de un identificador instancia la clase correspondiente de la capa de datos y usa una de sus funciones para realizar la consulta a la base de datos, guardando los valores de las distintas columnas y devolviéndolos a la capa de negocio, la cual almacena el resultado en el *array*.

El resultado obtenido puede luego ser usado, por ejemplo, por la capa de presentación, para mostrar los datos al usuario.

Figura 26: Resumen del proceso de negocio

2.2.2 Capa de datos

Esta capa da persistencia a la información mediante nuestra base de datos. La información se ve modificada según las acciones que realizamos a través del CMS. Esto es debido a que las clases de esta capa contienen funciones que realizan consultas a la base de datos. Estas consultas pueden ser para: obtener datos, añadirlos, borrarlos ó modificarlos. Por tanto, todas las clases de esta capa que interactúan con la base de datos cuentan, como mínimo, con las funciones *Delete, Insert, Select* y *Update*. Además, todas ellas tienen también una función constructor que instancia a la clase *DBHelper* (una clase auxiliar de la capa de datos que ayuda a realizar las consultas a la base de datos), así como otra función auxiliar *executeHelper* que recibe las consultas y, haciendo uso de la clase instanciada, las ejecuta y obtiene los resultados.

Como se puede ver en el diagrama siguiente, la capa de datos cuenta con las mismas clases que la de negocio y, a parte, otras dos clases más (*PanoCategoryJoin* y *DBHelper*):

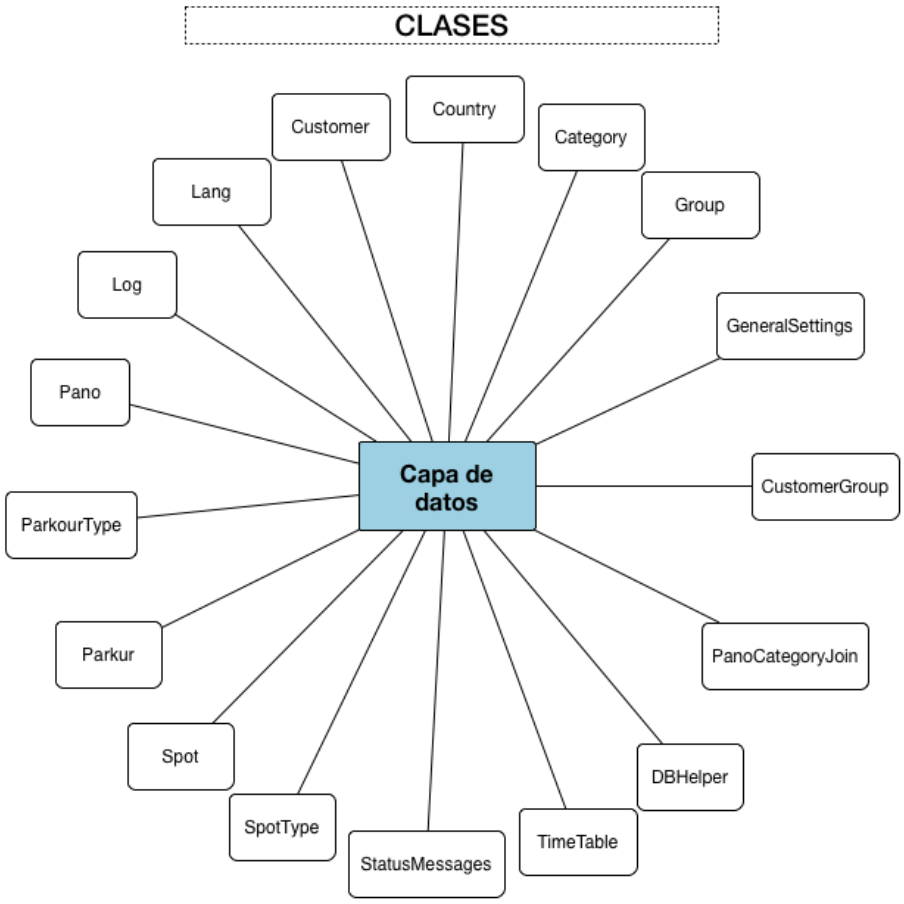

Figura 27: Clases que forman la capa de datos

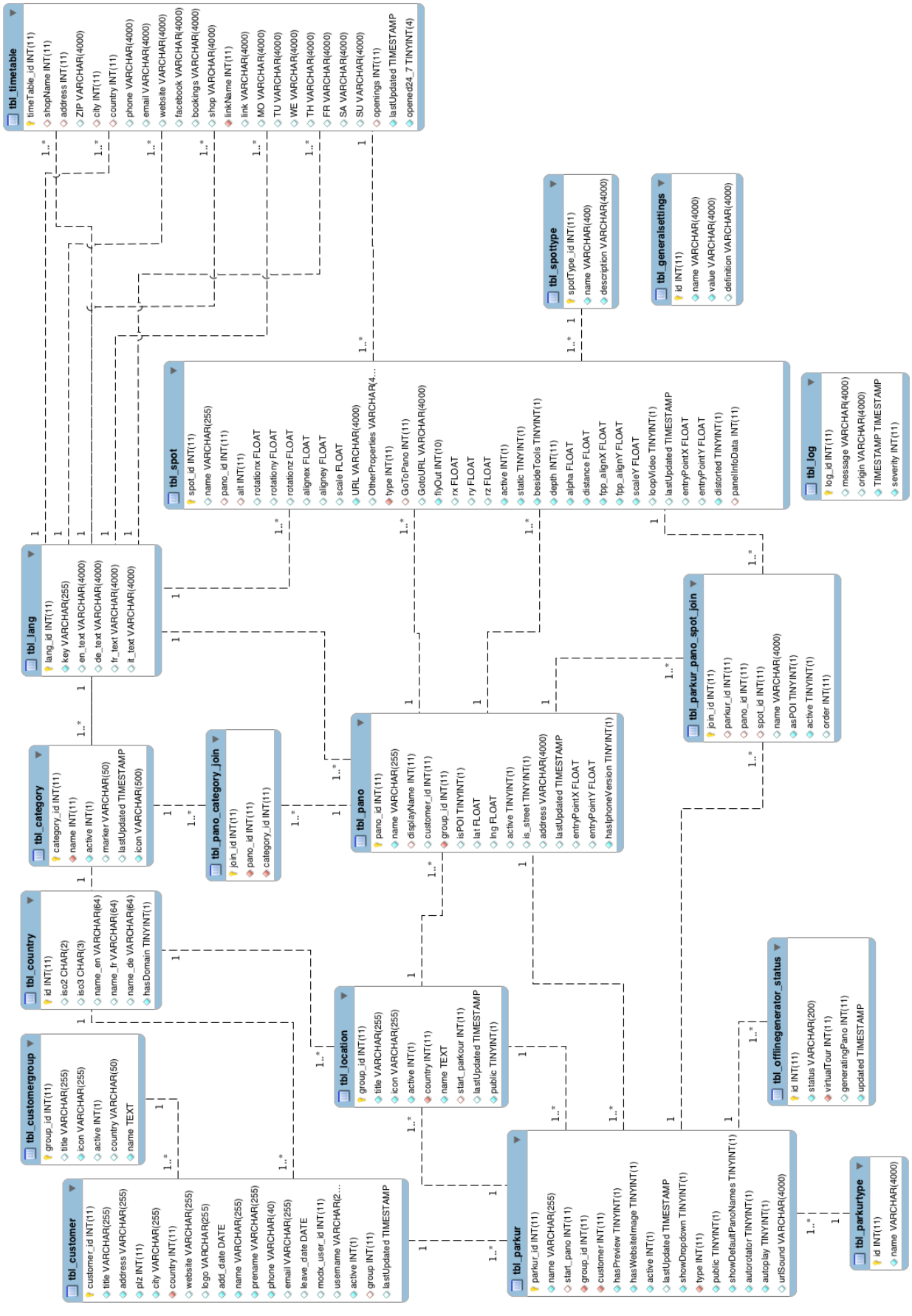

Figura 28: Diagrama de la base de datos

Para entender mejor la estructura y funcionamiento de la aplicación y los datos con los que se trabaja, vamos a exponer detalladamente cada una de las tablas de la Base de Datos (*Fig. 28*). Cabe destacar que algunas de las tablas cuentan con más campos de los que es posible introducir mediante el CMS. Esto es debido a que, durante el desarrollo de la aplicación, en algún instante el cliente ha cambiado los requisitos o ha querido añadir cosas a las que al final no se les ha dado uso alguno. Teniendo esto en cuenta, las tablas de la base de datos y los campos y propiedades más destacados e importantes de cada una son los siguientes:

- **Tbl_Customer**: Contiene la información sobre los clientes a los que el nuestro vende sus recorridos virtuales. Cuenta con los siguientes campos (los que no explicamos es porque no están en uso o no son importantes):
    - *Customer_id*: Se trata de un valor que la base de datos incrementa y añade automáticamente con cada nueva entrada, sirve para identificar a cada cliente puesto que es un valor único.
    - *Title*: Es el título que el usuario le ha asignado al cliente mediante el formulario.
    - *Address*: La dirección física donde reside o trabaja el cliente.
    - *Plz*: Código postal del cliente (“Plz” es una abreviatura alemana).
    - *City*: Ciudad en la que se encuentran el cliente y la dirección física indicada.
    - *Country*: País del cliente. En la base de datos se indica con un número, que es una clave foránea correspondiente a la tabla del mismo nombre. Por ejemplo, “6” corresponde al país con *id* igual a seis de la tabla *Country* (España).
    - *Website*: Página web del cliente.
    - *Add_date*: Fecha en que se ha añadido el cliente a la base de datos. El valor es puesto por la propia base de datos en el instante de añadir la entrada.
    - *Name*: Nombre del cliente.
    - *Prename*: Apellidos del cliente.
    - *Phone*: Teléfono de contacto del cliente.
    - *Email*: Dirección de correo electrónico del cliente.
    - *Username*: Nombre de usuario que se le asigna al cliente. Esto está relacionado con una posible futura ampliación en que los clientes puedan identificarse y acceder a una versión simplificada del CMS para gestionar los recorridos de su propiedad.

  - *Active*: Indica, mediante un "1" o "0", si el cliente está activo (si sigue habiendo una relación con él) o inactivo respectivamente.
  - *Group*: Grupo al que pertenece el cliente. Lo indicamos mediante un número identificador, que es clave foránea de la tabla *CustomerGroup*. Por ejemplo, el número "3" significa que el cliente pertenece al grupo de título "Austria" puesto que el identificador de ese grupo es "3".
  - *LastUpdated*: Fecha de la actualización más reciente realizada a los datos del cliente.

- **Tbl_CustomerGroup**: Se trata de una tabla en la que guardamos la información sobre los distintos grupos en que queremos clasificar a los clientes. Incluye los siguientes campos:
  - *Group_id*: Es el campo identificador al que la base de datos asigna un valor que se incrementa con cada nueva entrada, de forma que este nunca se repite.
  - *Title*: Contiene el nombre o título que le damos al grupo.
  - *Active*: Indica, mediante un "1" o "0" si el grupo está activo o inactivo respectivamente. Si un grupo está inactivo, no será posible seleccionarlo al crear o modificar un cliente.
  - *Country*: Indica el país al que pertenecen los usuarios clasificados en este grupo.

- **Tbl_Country**: En esta tabla mantenemos la información sobre cada país. En concreto, contamos con los siguientes campos:
  - *Id*: Es el campo identificador al que la base de datos asigna un valor que se incrementa con cada nueva entrada, de forma que este nunca se repite.
  - *ISO2*: Indica el código ISO oficial correspondiente al país de la entrada actual. Por ejemplo, el código para España es "ES".
  - *Name_en, Name_fr, Name_de*: Contiene el nombre oficial del país en inglés, francés y alemán respectivamente.
  - *hasDomain*: Con este campo indicamos si el país cuenta con su propio dominio web para acceder a la aplicación. Por ejemplo, Suiza cuenta con uno por lo que los usuarios que así lo deseen pueden acceder a través del dominio ".ch".

- **Tbl_Category**: Esta tabla contiene la definición de las distintas categorías en que podemos clasificar una escena panorámica siempre y cuando ésta sea un punto de interés. Contamos con los siguientes campos:
    - *Category_id*: Es el campo identificador al que la base de datos asigna un valor que se incrementa con cada nueva entrada, de forma que este nunca se repite.
    - *Name*: En vez de contener directamente el nombre de la categoría, este campo contiene un código identificador que es clave foránea de la tabla Lang. La tabla Lang contiene la traducción en los distintos idiomas de los textos más importantes que se muestran al usuario.
    - *Active*: Indica, mediante un "1" o "0" si la categoría está activa o inactiva respectivamente. Si está inactiva, no será posible seleccionarla al crear o modificar una escena virtual tipo POI ya que no se mostrará.
    - *Marker*: Contiene el nombre de una imagen marcador (*marker01, marker02,* …) que se utiliza para indicar en un pequeño mapa de *Google Maps* los distintos puntos de interés del grupo o localización a la que pertenece un recorrido. Este mapa se muestra en la parte inferior del visor cuando visualizamos dicho recorrido.
    - *LastUpdated*: Fecha y hora de la actualización más reciente realizada a la definición y propiedades de los marcadores.
    - *Icon*: Indica el nombre del archivo del icono que se ha asociado a la categoría al crearla o editarla. Este icono es representativo de la categoría (al contrario que el marcador, que puede estar compartido por varias categorías) y es elegido libremente por el usuario a partir de sus archivos.

category_id	name	active	marker	lastUpdated	icon
15	27417	1	marker04	2014-06-18 17:34:51	restaurant.png
16	27418	1	marker02	2014-06-18 17:35:32	bar.png
17	27419	1	marker03	2014-06-18 17:38:50	hotel.png
18	27420	1	marker01	2014-06-18 18:02:25	optician.png

Figura 29: Ejemplo de algunas de las categorías definidas en la tabla *Category*

- **Tbl_Location**: En esta tabla guardamos la información sobre las distintas ubicaciones que se crean a través del CMS. Las ubicaciones son conjuntos de recorridos virtuales de forma que, por ejemplo, podemos asignar la ubicación "Valencia" a todos los recorridos de esta ciudad. Estas ubicaciones pueden ser

luego accedidas mediante una URL de forma que se muestran al usuario uno tras otro los distintos recorridos que la componen.

Cuenta con los siguientes campos (nuevamente, los que no explicamos es porque no están en uso o no son importantes):

- *Group_id*: Es el campo identificador al que la base de datos asigna un valor que se incrementa con cada nueva entrada, de forma que este nunca se repite.
- *Active*: Señala, mediante un "1" o "0", si la ubicación está activa o inactiva respectivamente. Si está inactiva, no se podrá acceder a ella.
- *Country*: Indica, mediante un número, el identificador del país al que pertenece la ubicación. Se trata por tanto de una clave foránea de la tabla *Country*.
- *Name*: Se trata del nombre que le damos a la ubicación a través del formulario.
- *Start_parkour*: En este campo se indica, mediante un número identificador, el recorrido virtual en el que empieza la ubicación al cargarla. Este identificador es una clave foránea de la tabla *Parkur*. Cabe destacar que para saltar de recorrido en recorrido dentro de una misma ubicación, se muestra en la parte superior del visor una barra de selección de recorridos de entre los que la forman.
- *LastUpdated*: Fecha y hora de la actualización más reciente realizada a la ubicación.
- *Public*: Señala, mediante un "1" o "0", si la ubicación es pública o no respectivamente. Si no lo es, sólo los administradores pueden acceder a ella.

group...	title	icon	active	country	name	start_parkour	lastUpdated	public
1			0	1	Valencia	1	NULL	1
2			1	2	StJohann	301	NULL	1
3			1	1	Solothurn	247	NULL	1
4	Bern		1	1	Bern	45	NULL	1
5	Biel		1	1	Biel	58	NULL	1

Figura 30: Ejemplo de algunas ubicaciones definidas en la tabla *Location*

- **Tbl_Lang**: Esta tabla contiene la definición, en los cuatro idiomas requeridos (inglés, alemán, francés e italiano), de todas las palabras clave que se deben mostrar de forma multilingüe. Contamos con estos campos:

- *Lang_id*: Es el campo identificador al que la base de datos asigna un valor que se incrementa con cada nueva entrada, de forma que este nunca se repite.
- *Key*: Título identificador opcional que le podemos dar a cada palabra clave. Por ejemplo: *copyright*.
- *En_text*: La palabra clave en inglés. Por ejemplo: *All rights reserved*.
- *De_text*: La palabra clave en alemán. Por ejemplo: *Alle Rechthen vorbehalten*.
- *Fr_text*: La palabra clave en francés. Por ejemplo: *Tous droits réservés*.
- *It_text*: La palabra clave en italiano. Por ejemplo: *Tutti i diritti riservati*.

lang_id	key	en_text	de_text	fr_text	it_text
500		Middle room	Mittlerer Raum	Salle de milieu	Camera centrale
503		Background room	Hinterer Raum	Salle de fond	Camera sfondo
509		Wine cellar	Weinkeller	Cave	Cantina di vini
520		Room	Zimmer	Chambre	Camera

Figura 31: Ejemplo de entradas existentes en la tabla L*ang*

- **Tbl_OfflineGenerator_Status**: Esta tabla es especial, puesto que no sirve para dar persistencia a información que se muestra en la aplicación sino que la usamos como herramienta de depuración para el proceso de generación de las versiones *Offline* de los recorridos. Los siguientes campos nos proporcionan la información necesaria para asegurarnos de que el proceso se ha llevado a cabo con normalidad y, en caso de fallo, saber en qué fase de la generación ha ocurrido el mismo:
  - *Id*: Es el campo identificador al que la base de datos asigna un valor que se incrementa con cada nueva entrada, de forma que este nunca se repite.
  - *Status*: Indica el estado en que en que se encontraba la generación (*generating, waiting*, …) en el instante indicado por el campo *Updated*.
  - *VirtualTour*: Contiene el identificador del recorrido virtual del que se está generando la versión *Offline*. Es por tanto una clave foránea de la tabla *Parkur*.
  - *GeneratingPano*: Indica, con un número, la escena (de las que componen el recorrido) que se está generando en el momento de añadir la entrada a la base de datos. Es decir, no es el identificador de la escena, sino el

número de orden con que ésta cuenta dentro del recorrido. Este campo es muy útil ya que nos permite controlar que se han generado todas las escenas antes de cambiar el estado.

- *Updated*: Fecha y hora de la actualización más reciente realizada al estado.

id	status	virtualTour	generatingPano	updated
12734	GENERATING	1348	1	2014-07-02 09:39:33
12976	WAITING	526	0	2014-07-02 08:16:16
12975	WAITING	1344	0	2014-07-02 08:15:05
12974	WAITING	1352	0	2014-07-02 08:10:35

Figura 32: Ejemplo de entradas existentes en la tabla *Offline Generator Status*

- **Tbl_GeneralSettings**: Esta tabla es bastante simple ya que tan solo contiene tres entradas, correspondientes a las opciones de configuración global que hemos explicado previamente (*pág. 22*). Contamos por tanto con los siguientes campos:
  - *Id*: Es el campo identificador al que la base de datos asigna un valor que se incrementa con cada nueva entrada, de forma que este nunca se repite.
  - *Name*: Contiene el nombre de las opciones de configuración.
  - *Value*: Indica el valor actual de cada opción. En el caso de *ACTIVATE_UPPER_LEFT_LOGO* y *AUTOROTATOR* es un "1" o un "0" según si se ha marcado o no en el CMS. Mientras que la tercera opción (*AUTOPLAY_SECONDS*) contiene un valor numérico que indica los segundos que cada escena panorámica se muestra cuando el modo de reproducción automático está activado.
  - *Definition*: Este campo actualmente tan solo sirve para que el administrador identifique para qué sirve cada una de las opciones pero también puede usarse en un futuro para mostrar al usuario de la aplicación una pequeña ayuda.

id	name	value	definition
1	ACTIVATE_UPPER_LEFT_LOGO	0	It activates or deactivates the logo in the panorama viewer
2	AUTOROTATOR	0	NULL
3	AUTOPLAY_SECONDS	60	it sets the time that each panorama is shown in the autoplay mode

Figura 33: Contenido de la tabla *General Settings*

- **Tbl_Log**: Como su nombre indica, esta tabla registra todos los mensajes de información, error, depurado, etc. Que se generan durante la ejecución de la aplicación. Estos mensajes, introducidos por nosotros durante la programación de la aplicación, nos permiten identificar posibles errores y saber dónde han ocurrido, qué clases los han generado, etc. Es por tanto muy parecida a la tabla *Offline Generator Status* pero en este caso cubre toda la aplicación y no solo dicho proceso.
    - *Log_id*: Es el campo identificador al que la base de datos asigna un valor que se incrementa con cada nueva entrada, de forma que este nunca se repite.
    - *Message*: Se trata del mensaje que indica, dependiendo del tipo de evento, el error, información, aviso, etc. Que se ha generado, permitiéndonos así localizar los errores y seguir la ejecución.
    - *Origin*: Indica la clase o función que ha generado el evento e insertado el mensaje.
    - *Timestamp*: Contiene la fecha y hora en que se insertó el mensaje en la base de datos.
    - *Severity*: Como se explicó en la capa de negocio, contiene un número entre 0 y 4 que indica el tipo de mensaje, siendo 0 un error y 4 un mensaje de depuración.

log_id	message	origin	TIMESTAMP	severity
600	Failed Loading Parkur for id 1415: Parkour doesn't exist	BusinessLayer...	2014-07-27 12:47:42	0
599	Failed Loading Parkur for id 505: Parkour doesn't exist	BusinessLayer...	2014-07-23 13:19:01	0
598	Error in panoXML file	panoXML	2014-07-23 13:19:01	0

Figura 34: Ejemplo de entradas existentes en la tabla *Log*

- **Tbl_TimeTable**: Esta tabla contiene todos los campos necesarios para guardar la información que se le permite añadir al usuario en los *Hotspot* tipo globo de información.
    - *TimeTable_id*: Es el campo identificador al que la base de datos asigna un valor que se incrementa con cada nueva entrada, de forma que este nunca se repite.

- *ShopName*: Contiene un identificador que es clave foránea de la tabla Lang. Este identificador corresponde a una entrada de dicha tabla, la cual contiene el nombre del edificio asociado al *Hotspot* en los cuatro idiomas.
- *Address*: Igual que el campo anterior, solo que en este caso la clave foránea identificador indica la dirección física en que se encuentra.
- *ZIP*: Indica el código postal (ZIP) que corresponde a la ubicación de la tienda.
- *City*: Al igual que *ShopName* y *Address*, es una clave foránea de la tabla Lang. En este caso, indica el nombre de la ciudad.
- *Country*: Indica, mediante un número, el identificador del país al que pertenece la tienda. Se trata por tanto de una clave foránea de la tabla con el mismo nombre.
- *Phone*: Contiene el número de teléfono para contactar con la tienda.
- *Email*: Indica la dirección de correo electrónico con la que podemos contactar con la tienda.
- *Website*: Lista la dirección de la página web de la tienda.
- *Facebook*: Contiene la dirección al perfil o página de la tienda en la red social con dicho nombre.
- *Bookings*: Nos permite indicar la dirección web desde la que podemos realizar reservas o pedir cita (útil para, por ejemplo, restaurantes u hoteles).
- *Shop*: En este campo podemos indicar la dirección web de la tienda *Online* del establecimiento.
- *LinkName*: Está relacionado con el siguiente campo (*Link*) puesto que indica el nombre que le damos al enlace. Al igual que otros de los campos anteriores, este también se representa con un identificador que es clave foránea de la tabla *Lang*.
- *Link*: Es el enlace que se crea al subir el archivo con documentación sobre la tienda. Este enlace indica la dirección en que se guarda el archivo en el servidor.
- *MO, TU, WE, TH, FR, SA, SU*: Campos libres para cada día de la semana en que podemos indicar, por ejemplo, el horario de apertura de cada uno.
- *Openings*: Se trata de otro campo que es clave foránea de la tabla *Lang*. En el formulario que se presenta al usuario este campo corresponde a "*Season*" y sirve para indicar la temporada de apertura durante el año del establecimiento.
- *LastUpdated*: Fecha y hora de la actualización más reciente realizada a la información sobre el negocio.

- *Opened24_7*: Indica, mediante un “1” o un “0”, si el establecimiento está abierto las 24 horas del día o no respectivamente.

- **Tbl_Parkur**: Es la primera de las tres tablas principales y contiene toda la información necesaria sobre cada uno de los recorridos virtuales que creamos mediante el formulario correspondiente. Cuenta con los siguientes campos:
  - *Parkur_Id*: Es el campo identificador al que la base de datos asigna un valor que se incrementa con cada nueva entrada, de forma que este nunca se repite.
  - *Name*: Contiene el nombre que le asignamos al recorrido mediante el formulario. Por ejemplo: “AvenidaFrancia”.
  - *Start_Pano*: Indica mediante un identificador la escena panorámica por la que comienza el recorrido. Este identificador es una clave foránea de la tabla *Pano*.
  - *Group_Id*: Contiene un identificador que es clave foránea de la tabla *Location* y que indica la ubicación en la que se incluye el recorrido.
  - *Customer*: Es otro identificador, en este caso clave foránea de la tabla *Customer*, que define cuál es el cliente al que le pertenece el recorrido.
  - *HasPreview*: Indica, mediante un “1” o un “0”, si el recorrido cuenta con una imagen de vista preliminar del recorrido.
  - *HasWebsiteImage*: Similar al campo anterior, solo que en este caso la imagen mostrada es de mayor tamaño.
  - *Active*: Indica, mediante un “1” o “0”, si el recorrido está activo o inactivo respectivamente. Si está inactivo, no será posible acceder a él, tan solo editarlo.
  - *LastUpdated*: Fecha y hora de la actualización más reciente realizada a las propiedades del recorrido.
  - *ShowDropdown*: Indica, mediante un “1” o “0”, si al abrir el recorrido con el visor de KRPano se muestra o no en la parte superior el menú desplegable que nos permite seleccionar la escena que queremos ver de las que forman el recorrido.
  - *Type*: Contiene un identificador que es clave foránea de la tabla *ParkurType*. Este identificador indica de qué tipo es el recorrido. Como hemos explicado anteriormente, es importante poder clasificar los recorridos en distintos tipos ya que algunos de los clientes a los que el nuestro vende los recorridos pueden querer personalizarlos.

- *Public*: Indica, con un “1” o un “0”, si el recorrido es público o no respectivamente. Si un recorrido no es público, tan solo los usuarios con los permisos adecuados podrán acceder a él.
- *ShowDefaultPanoNames*: Puesto que al editar un recorrido permitimos la edición de las escenas que lo componen y, dentro de la edición de éstas, tenemos la opción de indicar un nuevo nombre para ellas que sea específico para el recorrido, es útil tener la posibilidad de activar o desactivar en cualquier momento esos nombres personalizados para el recorrido. Por tanto, este campo indica mediante un “1” o “0” si mostramos los nombres por defecto o los personalizados respectivamente.
- *Autorotator*: Nos permite indicar, mediante un “1” o un “0”, si esta propiedad de KRPano está activada o no para el recorrido. Si está activada, cuando el usuario termine de cargar el recorrido, este comenzará a rotar automáticamente sin que él necesite controlarlo con el ratón. KRPano nos permite, además, modificar la velocidad con que se produce la rotación (grados por segundo) así como el tiempo de espera (en segundos) desde que termina de cargar el recorrido hasta que comienza la misma. En nuestro caso hemos fijado estos valores en 2.0 y 1.5 respectivamente.
  Si se usa un valor negativo para la velocidad de rotación, el giro se produce hacia la izquierda en vez de hacia la derecha.
- *Autoplay*: Indica, mediante un “1” o un “0” si la reproducción automática del recorrido está activada. La reproducción automática permite al usuario visualizar todas las escenas panorámicas del recorrido virtual sin necesidad de tener que clicar los *Hotspots* manualmente.
- *URLSound*: Contiene, en caso de que el recorrido cuente con uno, la dirección URL del archivo de sonido que podemos reproducir al visualizarlo.

- **Tbl_ParkurType**: Esta tabla contiene los campos necesarios para definir los tipos de recorridos. Recordamos que la clasificación de los recorridos en tipos es necesaria porque alguno de los clientes puede querer que sus recorridos virtuales sean diferentes en algún aspecto como, por ejemplo, un set de iconos distinto para representar los distintos tipos de puntos. El clasificar éstos recorridos en tipos nos facilita en gran medida la programación de estos cambios.
  - *Id*: Es el campo identificador al que la base de datos asigna un valor que se incrementa con cada nueva entrada, de forma que este nunca se repite.
  - *Name*: Incluye los nombres que le damos a cada uno de los tipos de recorrido.

- **Tbl_Pano**: Esta tabla es otra de las principales y contiene todos los campos necesarios para dar persistencia a los datos de cada escena panorámica. Estos campos son los siguientes:
    - *Pano_Id*: Es el campo identificador al que la base de datos asigna un valor que se incrementa con cada nueva entrada, de forma que este nunca se repite.
    - *Name*: Aloja el nombre que le asignamos a la escena mediante el formulario de creación u edición.
    - *DisplayName*: Se trata de un identificador único, clave foránea de la tabla *Lang*, en la cual se indica el nombre de la escena que se mostrará a los usuarios en los cuatro idiomas distintos.
    - *Group_Id*: Contiene el identificador de la ubicación a la que pertenece la escena panorámica. Es por tanto clave foránea de la tabla *Location*.
    - *isPOI*: Indica, mediante un “1” o un “0”, si la escena es un punto de interés o no respectivamente. Como hemos explicado previamente, si una escena es POI se puede categorizar, sino no.
    - *Lat*, *Lng*: Estos dos campos contienen, respectivamente, la latitud y longitud en que se ha situado mediante el mapa interactivo el POI (solo si la escena lo es).
    - *Active*: Indica, mediante un “1” o un “0”, si la escena está activa o inactiva respectivamente. Si está inactiva, no será posible acceder a ella.
    - *Is_Street*: Si la escena representada es una calle, se indica un “1”, en cualquier otro caso, se indica un “0”. Es necesario identificar estas escenas puesto que el cliente desea que éstas cuenten con ciertos elementos distintos a los demás tipos de escenas.
    - *Address*: Contiene la dirección física donde se encuentra lo representado en la escena panorámica.
    - *LastUpdated*: Fecha y hora de la actualización más reciente realizada a las propiedades de la escena.
    - *EntryPointX*, *EntryPointY*: Es un conjunto de valores numéricos que representan una desviación en grados respecto a la escena. Contamos con la posibilidad por tanto de añadir valores para el eje horizontal y vertical lo que significa que podemos definir un punto concreto de la escena a partir de éstos. En nuestro caso, este es un punto de entrada, que es lo que se muestra al usuario nada más cargar la escena.
    - *HasIphoneVersion*: Indica, mediante un “1” o un “0”, si la escena cuenta con una versión móvil ya creada o no respectivamente. La versión móvil

consiste simplemente en una copia de las imágenes de la escena pero a menor tamaño y peso.

- **Tbl_Pano_Category_Join**: Se trata de una tabla importante puesto que relaciona el identificador de una escena con el de cada una de las categorías a las que ésta pertenece. Por ejemplo, la escena con identificador 6702 aparece cuatro veces en la tabla por lo que está clasificada en cuatro categorías distintas (*Fig. 35*). Cabe recordar que una escena sólo puede clasificarse en una o más categorías si es un punto de interés (POI).
  Los campos de esta tabla son los siguientes:
  - *Join_id*: Es el campo identificador al que la base de datos asigna un valor que se incrementa con cada nueva entrada, de forma que este nunca se repite.
  - *Pano_id*: Clave foránea de la tabla *Pano* que identifica a una escena.
  - *Category_id*: Clave foránea de la tabla *Category* que identifica a una categoría.

join_id	pano_id	category_id
327	6702	32
329	6702	24
332	6702	33
381	6702	28

Figura 35: Ejemplo de la relación entre escenas y categorías

- **Tbl_Spot**: Es la tercera de las tablas principales y contiene una gran cantidad de campos debido a que la mayor parte de éstos corresponden a cada uno de los parámetros de un Hotspot que KRPano nos permite configurar. Aunque muchos de éstos campos no los estemos usando en la actualidad, es importante tenerlos en la base de datos en el caso de que sea necesario utilizarlos en el futuro; de esta forma nos ahorramos el tener que realizar cambios mayores. Los campos más importantes son los siguientes:
  - *Spot_Id*: Es el campo identificador al que la base de datos asigna un valor que se incrementa con cada nueva entrada, de forma que este nunca se repite. Es muy importante puesto que nos permite usar este identificador a la hora de implementar las acciones de cada punto. Por ejemplo, para que

se cargue una nueva escena al clicar en un punto, definimos y utilizamos la acción "*loadpano[Spot_id]"* en el archivo XML que interpreta KRPano.

- *Name*: Es el nombre que le asignamos de forma automática al punto. Es muy importante puesto que identifica de forma única a cada uno en el archivo XML que interpreta KRPano.
- *Pano_Id*: Clave foránea de la tabla *Pano* que identifica a una escena panorámica en particular, a la cual el punto actual pertenece.
- *Alt*: Clave foránea de la tabla *Lang* que contiene, en los cuatro idiomas, el texto que se muestra al pasar el ratón por encima de un *Hotspot*.
- *AligneX*, *AligneY*: Estos valores indican la distancia desde el punto de alineamiento o anclaje del elemento al punto del borde (*edge point*). Para entender bien esta propiedad, incluimos una ilustración oficial de KRPano que lo explica detalladamente (*Fig. 38).*
- *Scale*: Se trata de un valor, entre 0.0 y 1.0, que indica el tamaño del elemento (*Hotspot*).
- *URL*: Contiene la dirección física donde se encuentra la imagen icono correspondiente al punto actual en el servidor. Por ejemplo: "/panos/hotspots/arrow.png".
- *OtherProperties*: Es un campo auxiliar que podemos utilizar en un futuro si surge la necesidad de añadir alguna propiedad adicional.
- *Type*: Se trata de un identificador, clave foránea de la tabla *Spot type*, que indica el tipo del punto (*Hotspot*). Por ejemplo, un "4" significa que es un punto tipo flecha mientras que un "2" indica que es una imagen.
- *GoToPano*: Indica, mediante un identificador clave foránea de la tabla *Pano*, cuál es la escena destino a la que se llega si se presiona el punto.
- *GoToURL*: Puesto que contamos con la posibilidad de añadir hipervínculos a los puntos, es necesario un campo como este, en el que se guarda la directriz *Javascript* que abre la dirección contenida en el hipervínculo.
- *flyOut*: Contiene un valor, "1" o "0", que indica si el punto realiza o no la acción de "vuelo", la cual consiste en presentar de forma maximizada un elemento de la escena. Por ejemplo, puede mostrarse en la escena un libro sobre una mesa y, si el usuario selecciona el libro, se le presenta tras una pequeña animación de entrada una vista completa del mismo. En las figuras 36 y 37 mostramos un ejemplo detalladamente.
- *Rx*, *Ry*, *Rz*: Estos tres campos contienen los valores, en grados, de rotación del punto en cada eje. Puesto que hay tres valores, existen tres ejes, lo que implica que es una rotación en tres dimensiones (3D). El valor

por defecto para todos es 0.0. Cabe destacar que esta propiedad solo se aplica al punto cuando el campo *Distorted* está activado.

- *Active*: Indica, mediante un "1" o un "0", si el punto está activado o no. Si no está activado, no se muestra en el visor de escenas de KRPano, pero es posible seguir editándolo desde el panel de administración.
- *Alpha*: Es un valor, propiedad de los elementos de KRPano, que nos permite indicar el nivel de transparencia de los mismos. Por ejemplo, un *Hotspot* con valor alfa de 0.0 es completamente transparente y por tanto no visible mientras que uno con valor 1.0 es totalmente visible. Eso sí, aunque un elemento sea completamente transparente aun está presente por lo que, en el caso de los puntos, es posible seguir seleccionándolos.
- *LoopVideo*: Es una propiedad exclusiva de los puntos tipo *Shockwave Flash* (SWF), utilizados para mostrar pequeños videos en una escena, que, mediante un "1" o un "0", indica si el video se reproduce continuamente en bucle o tan solo una vez respectivamente.
- *LastUpdated*: Fecha y hora de la actualización más reciente realizada al punto y sus propiedades.
- *EntryPointX*, *EntryPointY*: Similar a los valores de punto de entrada de las escenas pero, en este caso, de forma individual para cada punto. Indica, por tanto, la vista que se muestra al clicar en el punto.
- *Distorted*: Indica, mediante un "1" o un "0", si el punto tiene dicha propiedad activada o no respectivamente. Solo es posible rotar el punto en un espacio de tres dimensiones si cuenta con esta propiedad activada.
- *PanelInfoData*: Si el punto es de tipo globo de información, este campo contiene una clave foránea de la tabla *Timetable* que identifica a la entrada de la misma que contiene la información que se quiere mostrar al presionar en el punto. En el caso de que no sea un globo de información, este campo es nulo.

En las siguientes dos figuras se muestra un ejemplo de los elementos que cuentan con la propiedad *Fly Out* en una escena. El usuario puede identificar fácilmente estos elementos ya que destacan sobre los demás y, al pasar el puntero del ratón por encima, el cursor cambia. Cuando el usuario clica en uno de estos elementos, se le muestra este en mayor tamaño tras una pequeña animación de forma que pueda leerlo o visualizarlo fácilmente. En el ejemplo de la figura se le muestran al usuario tres

elementos destacados en el tablón de anuncios y al clicar, por ejemplo, en el del medio, se le muestra un documento con información sobre cursos de idiomas.

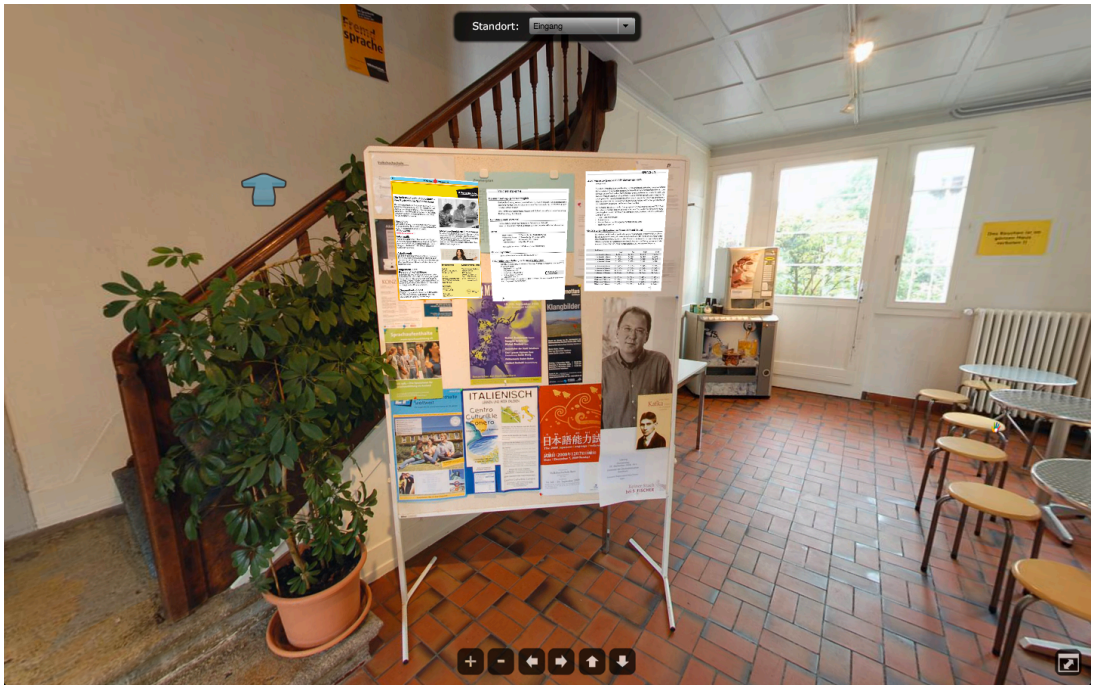

Figura 36: Los tres papeles blancos del tablón de anuncios cuentan con la propiedad *Fly Out*

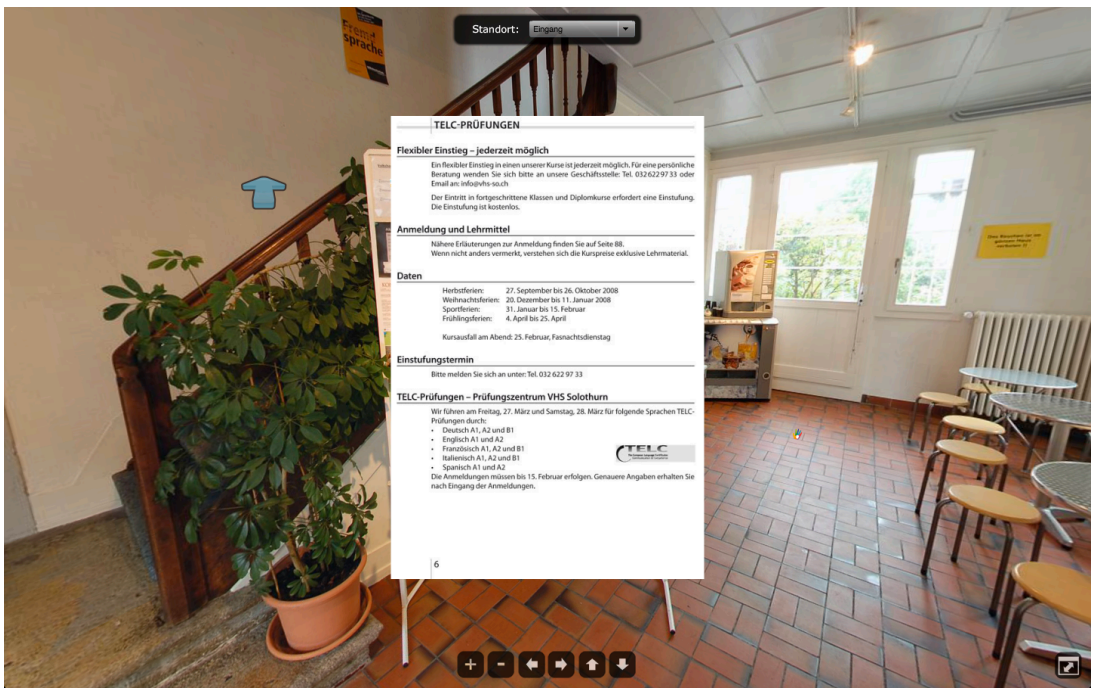

Figura 37: Resultado de clicar en uno de los elementos

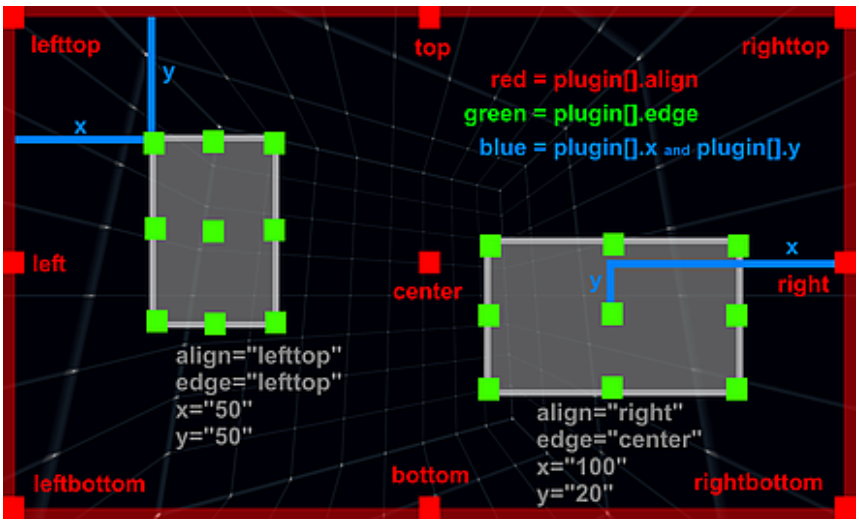

Figura 38: Ilustración explicativa de la propiedad de alineamiento de un elemento

- **Tbl_SpotType**: Mediante esta tabla definimos los distintos tipos de puntos que se pueden añadir a una escena y asociamos a éstos un identificador. Los campos son los siguientes:
    - *SpotType_id*: Es el campo identificador al que la base de datos asigna un valor que se incrementa con cada nueva entrada, de forma que este nunca se repite.
    - *Name*: Contiene el nombre que le damos al tipo de punto. Por ejemplo: "swf", "picture", "video", …
    - *Description*: Es un campo opcional y sin mayor utilidad que poder añadir una descripción o explicación para identificar más fácilmente la función de cada tipo de punto.

- **Tbl_Parkur_Pano_Spot_Join**: Esta tabla es de alta importancia puesto que determina, entre otras cosas, si una escena está relacionada con un recorrido (es decir, si forma parte del recorrido), qué nombre se le da a cada escena dentro del recorrido (ya que puede ser distinto del nombre propio de la escena) y cuál es el orden de cada escena dentro de un recorrido.

  Las entradas de la tabla en que se define un recorrido y una escena, el identificador del *HotSpot* (*Spot_id*) es nulo y cuentan con un orden (columna *Order*) definido, representan la relación recorrido-escena (*Fig. 39*). Mientras que las que sí contienen un *HotSpot* definido indican la relación ternaria recorrido-escena-punto. Por tanto, es esencial que para que una escena esté incluida dentro

de cierto recorrido, exista una entrada con *spot_id* nulo que relacione a la escena con el recorrido y cuente con el orden definido.

Todo esto es necesario debido a que en cierto instante durante el desarrollo del proyecto el cliente nos pidió que existiese la posibilidad de que una escena pueda estar presente en más de un recorrido (es decir, que los recorridos puedan "cruzarse"). Los campos que forman la tabla son los siguientes:

- *Join_id*: Es el campo identificador al que la base de datos asigna un valor que se incrementa con cada nueva entrada, de forma que este nunca se repite.
- *Parkur_id*: Clave foránea de la tabla *Parkur* que identifica a un recorrido.
- *Pano_id*: Clave foránea de la tabla *Pano* que identifica a una escena panorámica.
- *Spot_id*: Clave foránea de la tabla *Spot* que identifica a un punto (*HotSpot*).
- *Name*: Clave foránea de la tabla *Lang* que relaciona un identificador con el nombre de la escena panorámica en cuatro idiomas distintos. En este caso, el nombre es el definido dentro del recorrido para esa escena, el cual puede ser distinto del nombre propio de la escena.
- *AsPOI*: Indica, mediante un "1" o un "0", si la escena es un punto de interés dentro del recorrido y si, por tanto, puede categorizarse.
- *Active*: Indica, mediante un "1" o un "0", si la relación está activa. Si no lo está (*active* = 0), la escena no estará incluida dentro del recorrido (aunque la relación exista en la base de datos).
- *Order*: Define el orden de una escena concreta dentro de un recorrido. Puesto que este está compuesto por más de una escena, al definir un orden conseguimos que aparezcan de forma ordenada en el selector de escenas del mismo así como que, cuando la reproducción automática está activada, se muestren las escenas en el orden que deseamos. Sin embargo, cabe destacar que la primera escena que se nos muestra al visualizar un recorrido no tiene por qué ser la misma que, según el orden, debería serlo. Esto es debido a que el campo "Start_Pano" (escena inicial) de la tabla *Parkur* (tabla de los recorridos) se tiene en cuenta antes que el orden definido en esta tabla.

join_id	parkur_id	pano_id	spot_id	name	asPOI	active	order
29560	1	6	NULL	4540735	1	1	8
29964	1	6	17112	NULL	0	1	NULL
12046	1	6	43	NULL	0	1	NULL
28744	299	6	NULL	4540735	0	1	3

Figura 39: Ejemplo de una escena incluida en más de un recorrido

En cuanto a las clases de la capa de datos, la estructura, funcionamiento y proceso de ejecución es el siguiente:

1. Declaramos las variables necesarias para instanciar la clase auxiliar *DBHelper* y creamos la función constructora, dentro de la cual se instancia dicha clase. Esta función se ejecuta automáticamente cada vez que se hace uso de la clase en que se ha definido, de forma que cada vez que se use la clase, se instanciará *DBHelper* automáticamente. Definimos también la otra función auxiliar, *ExecuteHelper*, que ejecuta la lógica para realizar la consulta y obtener los resultados mediante una función de *DBHelper*.
2. Creamos el resto de funciones de la clase, que contienen los distintos tipos de consultas necesarias como, por ejemplo: inserción, obtención, modificación/actualización, borrado, etc. Estas funciones reciben de la capa de negocio las distintas variables necesarias para su ejecución y, usando los valores de dichas variables, crean la consulta, la asocian a una nueva variable, y ejecutan la función auxiliar *ExecuteHelper* con ella, obteniendo los resultados, y devolviéndolos a la capa de negocio.
3. La capa de negocio obtiene el resultado y, por tanto, la capa de presentación puede ahora hacer uso de ésta para mostrar los datos al usuario. Se crea por tanto una especie de jerarquía en la que la capa de datos y la de presentación no interactúan entre ellas puesto que tan solo se “comunican” con la de negocio, que desempeña una función de intermediario en la obtención y comunicación de los datos.

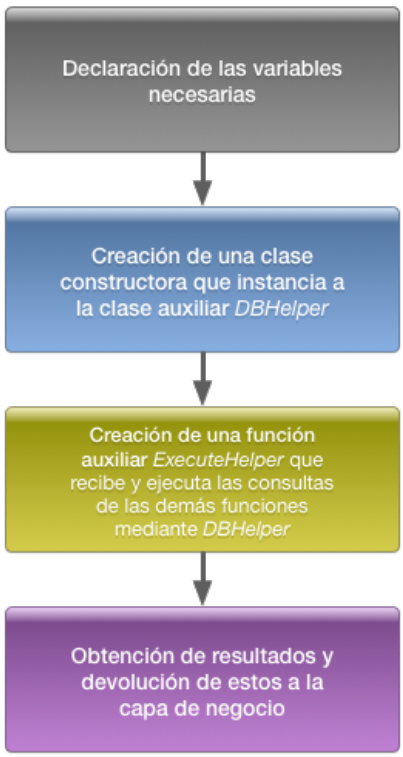

Figura 40: Resumen del proceso de la capa de datos

2.2.3 Capa de presentación

En cuanto a la capa de presentación, como ya comentamos al principio el CMS ha sido realizado con la ayuda de la herramienta MODX puesto que proporciona total libertad a la hora de crearlo. Es decir, no partimos de ningún modelo predefinido o estructurado en bloques. MODX está desarrollado en PHP y es ideal para nuestro proyecto puesto que se adapta perfectamente a comunicaciones ágiles con Ajax las cuales son muy útiles cuando tratamos con grandes cantidades de datos ya que podemos realizar búsquedas y filtrados en tiempo real, entre otras cosas. Sin embargo, el presente proyecto se centra sobretodo en la parte funcional o lógica de la aplicación por lo que no vamos a entrar en el desarrollo del CMS.

Nuestra capa de presentación no está formada tan solo por el CMS, también contamos con otros elementos de alta importancia como, por ejemplo, la lógica necesaria para generar los ficheros XML que interpreta KRPano para mostrar al usuario los recorridos, escenas y puntos así como los archivos HTML que controlan la presentación de dichos datos e incluso la generación de los archivos PDF y ZIP

(versión *Offline*). A continuación, exponemos dichas clases y funciones de forma más detallada:

- **XMLGenerator**: Es una de las clases más importantes de la aplicación ya que, como su nombre indica, se encarga de generar el código de los distintos ficheros XML con que cuenta cada recorrido. En concreto, genera los siguientes ficheros:
    - **Buttons.xml**: Contiene el código necesario tanto para mostrar los distintos botones que conforman el visor de escenas como para implementar la funcionalidad de éstos (las acciones que se ejecutan al clicar en ellos). La documentación oficial de KRPano proporciona ejemplos para la implementación de dichos botones ya que, al igual que la propiedad *Fly Out*, también se trata de distintas extensiones o *plugin* oficiales de la aplicación.

      Además de los botones de la parte inferior (*Fig. 41*) también contamos con un menú desplegable en la parte superior donde el usuario puede seleccionar a qué escena de las que forman parte del recorrido acceder (*Fig. 42*). De esta forma, no es necesario que el usuario utilice los *Hotspot* flecha para pasar de una escena a otra sino que puede llegar directamente a la que él desee. Si se ha definido un orden mediante el campo *Order* de la tabla que relaciona de forma ternaria los recorridos, escenas y puntos, este será el orden en que se listen las escenas en el menú siempre y cuando la escena inicial definida para el recorrido sea la misma que dicta según el orden. Si no es así, la escena inicial definida para el recorrido tiene preferencia sobre el orden definido.

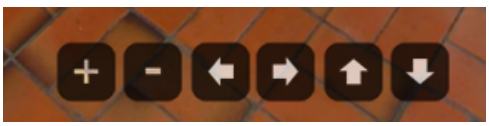

Figura 41: Botones de navegación de un recorrido (botones de zoom y navegación)

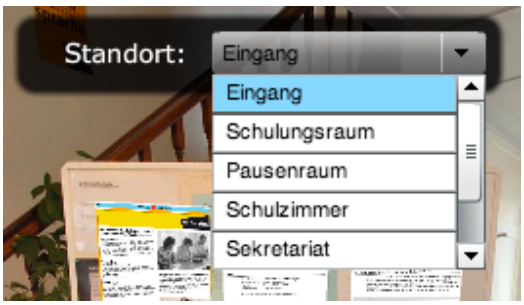

Figura 42: Botones de navegación de un recorrido (selección de escena)

- **ButtonsAutoplay.xml**: Similar al anterior fichero, sólo que con una pequeña modificación para adaptarse a los recorridos con reproducción automática. En concreto, añade un botón de reproducción que al ser pulsado y tras un pequeño intervalo comienza la reproducción automática. Además, es cambiado por un botón de pausa/detención al estar activado.

Figura 43: Botones de navegación de un recorrido con reproducción automática

En ambos casos se incluye en la esquina inferior derecha un botón que permite acceder al modo pantalla completa (*Figs. 36 y 37*), donde se muestra una visión integra de la escena, sin los elementos adicionales como la selección de idioma o la descarga de los ficheros PDF con los códigos de inserción.

Dependiendo del tipo en que esté clasificado el recorrido y si dicho tipo requiere un conjunto de botones distinto (requisitos del cliente), la clase *XMLGenerator* se encarga de generar un fichero XML con los botones específicos para ese tipo de recorrido.

El tercer archivo que se encarga de generar la clase y uno de los más importantes es el XML correspondiente a cada escena del recorrido. Este archivo XML es creado una vez por cada idioma: alemán, inglés, italiano y francés; así como en dos versiones por cada uno de éstos: normal y iPhone (para dispositivos móviles).

- **panoXML_[DE, EN, FR, IT].xml**: Este archivo es esencial para la aplicación puesto que es el archivo XML que interpreta KRPano (usa una sintaxis propia de KRPano) para mostrar al usuario la escena panorámica 360º, los puntos (*Hotspot*) asociados a ella y la funcionalidad de éstos (salto a otras escenas del recorrido). La estructura de este archivo es la mostrada al comienzo del documento (*Fig. 02*), aunque algunos elementos son opcionales o varían según la plataforma. Los elementos esenciales que se incluyen en todas las escenas son las imágenes, los puntos, y las acciones (asociadas a los puntos definidos).

  La clase obtiene los datos para construir estos archivos a partir de las funciones de las distintas clases de la capa de negocio, las cuales como se ha comentado previamente obtienen éstos de la capa de datos, la cual interactúa con nuestra base de datos. Las tablas más importantes de la misma para este caso son: *Pano* (escenas), *Parkur* (recorridos), *Spot* (puntos) y la tabla ternaria que relaciona éstas tres.

  - **panoXML_[DE, EN, FR, IT]_iphone.xml**: La versión para dispositivos móviles difiere levemente respecto a la normal. Las principales diferencias son las direcciones URL o físicas (en el caso de la versión *Offline* del recorrido) a las distintas imágenes que componen la escena panorámica 360º ya que apuntan a los ficheros de la versión móvil (de menor resolución y peso). Además, pequeños efectos como la muestra de información al poner el puntero encima de un *Hotspot* (*tooltips*) no es posible en dispositivos móviles por lo que queda descartada en este fichero. Así mismo, otra gran diferencia es la ausencia de botones; puesto que los dispositivos móviles cuentan con una pantalla táctil, no es necesario añadir botones de navegación ya que todas las acciones se pueden realizar mediante ésta.

- **HTMLGenerator.php**: Genera los archivos HTML de los recorridos virtuales, uno por cada idioma. Estos archivos HTML incluyen:
  - Los *scripts* necesarios para la correcta ejecución del visor (hojas de estilo, archivos *javascript*, etc.).
  - La declaración y definición de las funciones imprescindibles para cargar las escenas panorámicas correspondientes al recorrido y mostrarlas al

usuario, incluyendo los botones (uso de los archivos XML creados por la clase anterior).

- Elementos adicionales de la interfaz del visor de recorridos, situados en la parte inferior del mismo (pie de página):
  - Mapa de Google con indicadores de los distintos POI incluidos no solo en el recorrido, sino en todos los recorridos que forman parte del grupo (localización) del que forma parte el recorrido. Por ejemplo, si el usuario está visualizando el recorrido de la Avenida de Francia, se le mostrará un mapa con todos los puntos de interés del grupo al que pertenece (Valencia).
  - Selección de idioma entre los cuatro disponibles.
  - Información adicional (logos, *copyright*,…).

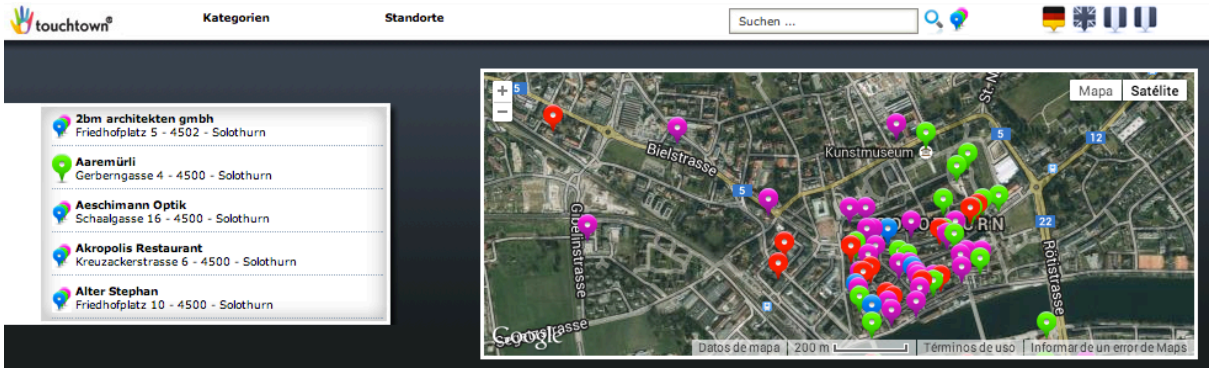

Figura 44: Pie de página de la interfaz del visor de recorridos virtuales

- **OfflineGenerator.php**: Esta clase se encarga, principalmente, de crear las versiones *offline* de los recorridos (y por tanto escenas que este incluye) pero también genera los elementos que se muestran a pie de página al visualizar cada escena panorámica (la clase anterior se encargaba de generar lo que se muestra al visualizar una localización, no una escena individual, pero la idea es la misma).

  Estas versiones son descargadas por el usuario como un fichero comprimido en formato ZIP, el cual contiene todos los archivos necesarios para poder visualizar el recorrido sin necesidad de estar conectado a Internet. La clase cuenta con distintas funciones, cada una de las cuales se encarga de copiar los distintos ficheros necesarios del servidor y guardarlos en una carpeta temporal, siguiendo una estructura determinada. Así mismo, también se encarga de crear

la versión sin conexión de los ficheros HTML ya que, por ejemplo, es posible que algunos de los *scripts* u hojas de estilo de la versión en red estuvieran alojados en servidores públicos remotos (lo que se suele conocer como *Content Distribution Networks, CDN,* o redes de entrega de contenidos) y por tanto es necesario descargar esos archivos y cambiar la dirección en el archivo HTML a la local. Una vez tenemos todos los archivos necesarios copiados o creados dentro de sus carpetas correspondientes formando la estructura correcta, se crea el fichero ZIP mediante las funciones nativas de PHP, se le envía al usuario y se borra del servidor una vez este finaliza la descarga.

En cuanto al pie de página, que se muestra cuando la visualización en pantalla completa no está activada, la clase no solo genera el de la versión *offline* sino también la versión *online*. Hay una pequeña diferencia entre ambas y es que la versión *offline* no cuenta ni con el formulario de contacto para los clientes ni con los enlaces para generar los archivos PDF y ZIP, lógicamente (*Figs. 45 y 46*).

Figura 45: Pie de página del visor individual de escenas (versión *online*)

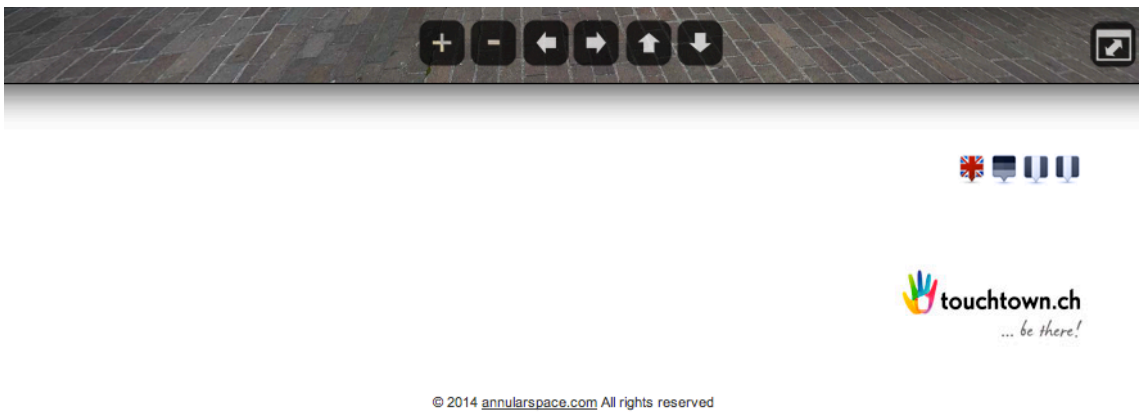

Figura 46: Pie de página del visor individual de escenas (versión *offline*)

- En cuanto a la **creación de los ficheros PDF,** como comentamos en la sección de recursos utilizados, hacemos uso de la herramienta gratuita TCPDF, la cual nos proporciona una clase PHP de fácil uso para crear dichos ficheros. Estos archivos cuentan con una estructura bastante simple ya que el contenido consiste tan solo de pequeños trozos de código que permiten a los clientes insertar las escenas y recorridos en sus propias páginas web. Esta inclusión se realiza a través de un *iframe* HTML en el caso de los recorridos, lo cual permite insertar una página HTML dentro de otra, mientras que en el caso de las escenas individuales simplemente se cargan directamente en la propia página.

Codes for Virtual Tour
2bm architekten gmbh

Codes

```
<iframe id="panoramaFrame" frameborder="no" width="100%" title="panoramaFrame" framespacing="0"
border="0" name="panoramaFrame" src="http://www.touchtown.ch/Solothurn/2bm" style="display: block; width:
100%; float: left; height: 100%;"></iframe>
```

Aussenansicht

```
<div id="panoViewer" style="height: 100%; ">
  <div id="panorama">
  </div>
  </div>
<script type="text/javascript" src="http://cf-360.local.ch/pano/swfkrpano.js"> </script>
<script type="text/javascript" src="http://cf-360.local.ch/pano/touchtownUtils_cloud.js"> </script>
<script>loadPano('2bm', '2bm_2bm_00', 'EN');</script>
```

Büro Solothurn

```
<div id="panoViewer" style="height: 100%; ">
  <div id="panorama">
  </div>
  </div>
<script type="text/javascript" src="http://cf-360.local.ch/pano/swfkrpano.js"> </script>
<script type="text/javascript" src="http://cf-360.local.ch/pano/touchtownUtils_cloud.js"> </script>
<script>loadPano('2bm', '2bm_2bm_01', 'EN');</script>
```

Figura 47: Contenido de los ficheros PDF que se generan

Con esto concluimos la descripción y explicación de los elementos más importantes que forman parte de la presentación de la aplicación de cara al usuario.

En lo que se refiere al funcionamiento de la capa de presentación, ésta obtiene todos los datos necesarios a partir de la capa de negocio (la cual los pide a la capa de datos, que se comunica con la base de datos para obtenerlos), los procesa, y los muestra al usuario final a través del CMS, el visor de KRPano, etc. De esta forma se completa el ciclo de obtención e intercambio de datos de las aplicaciones *software* programadas por capas.

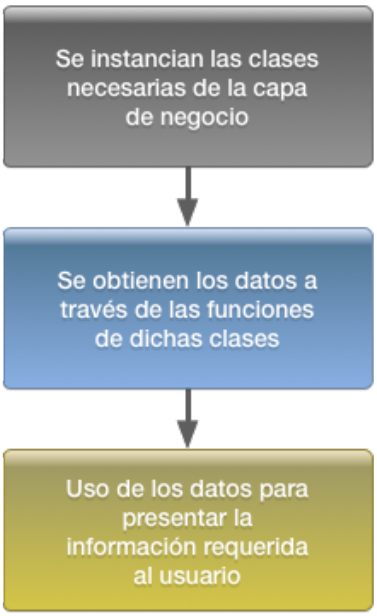

Figura 48: Resumen del proceso de la capa de presentación

2.3 Pruebas y control de errores

La realización de pruebas o *testing* es una de las partes más importantes de cualquier proyecto de desarrollo de aplicaciones pu que nos permite encontrar y corregir los posibles errores con que, en otro caso, se encontraría el usuario final. Esto implica un ahorro en costes de desarrollo y brinda un mayor nivel de satisfacción al usuario final.

En nuestro caso, hacemos uso de una técnica llamada “unit testing”, que consiste en realizar pruebas de los distintos módulos de código que componen nuestra aplicación. Es importante, por tanto, probar todas las nuevas funciones y clases conforme las programamos ya que así nos evitamos futuros dolores de cabeza tratando de buscar dónde puede estar el error o errores que provocan cierto comportamiento incorrecto por parte de nuestra aplicación, lo cual es especialmente problemático en proyectos de cierta complejidad. Algunos de los inconvenientes más destacados de realizar este tipo de pruebas son:

- No nos permiten evaluar todos los distintos casos de ejecución posibles (a no ser que la aplicación sea muy simple).
- Tan solo prueban la propia unidad o módulo de código y no la aplicación en conjunto (relaciones entre los distintos módulos, etc.).

- Aunque los distintos módulos que componen la aplicación funcionen correctamente, mediante este tipo de pruebas no podemos comprobar cuál será el rendimiento de la aplicación final por lo que es posible que tengamos módulos de código que funciona correctamente pero que cuentan con un alto grado de optimización posible y, si no se optimizan, ralentizarán la ejecución de la aplicación final.

Cabe destacar que el *unit testing* puede realizarse tanto de forma automática (existen *frameworks* para ello, como PHPUnit para PHP, aunque algunos lenguajes cuentan con soporte nativo) como manual. En conclusión, esta técnica es útil para comprobar que los distintos módulos de código realizan las acciones que nosotros deseamos, pero no nos permiten saber cómo se van a comportar esos módulos en relación con los demás en la aplicación final por lo que realizar *unit testing* no nos exime de testear el producto final en profundidad.

Aparte del *unit testing* y como ya hemos comentado previamente en la descripción de la capa de datos, también hemos implementado un sistema de *log* que nos facilita en alta medida el mantenimiento y búsqueda de errores en la aplicación final. Este sistema es esencial para nosotros y pensamos que todas las aplicaciones de cierto nivel deberían contar con él ya que ahorra mucho tiempo y dinero y no es para nada complicado de implementar e incluso nos permite reforzar los puntos débiles con los que cuenta el *unit testing*.

3. Producto final

Aunque en este caso el producto nunca se da por terminado, puesto que nuestro cliente desea realizar cambios o adiciones constantemente, una vez finalizada la primera iteración de la aplicación y hechas todas las pruebas necesarias, el resultado obtenido ha sido satisfactorio teniendo en cuenta la complejidad del proyecto y la experiencia previa con que se contaba.

Al tratarse de una aplicación que maneja una alta cantidad de datos de distintas fuentes, el rendimiento no es el mejor que se puede esperar pero resulta complicado optimizar un puzle que cuenta con tantas piezas relacionadas entre sí. Además, en algunos casos el rendimiento obtenido depende altamente de las prestaciones con que cuenta el usuario final. Por ejemplo, a la hora de cargar las escenas, es importante contar con una buena velocidad de descarga puesto que las imágenes utilizadas son de alta calidad (para evitar en lo posible problemas de pérdida de calidad al realizar zoom en la imagen mediante el visor de KRPano) y lo mismo ocurre por tanto al descargar los archivos ZIP con la versión *offline* de los recorridos virtuales, especialmente si el recorrido está formado por muchas escenas.

Algunos aspectos que podríamos mejorar en un futuro son los relacionados con la presentación al usuario, sobretodo en el caso de los formularios con muchos campos ya que a veces queda todo muy apelotonado o algunos campos no queda del todo claro qué realizan a no ser que conozcas muy bien la aplicación, lo cual no es un problema en este caso ya que la aplicación tan sólo es usada por el propio cliente pero en cualquier otro caso sería buena práctica dejar clara la función de cada elemento del formulario. En concreto, este problema viene dado, en gran medida, por las distintas modificaciones o adiciones que el propio cliente nos pide cuando le hacen falta o le surge una idea y, puesto que hay que cumplir cierta fecha de entrega, no es fácil mantener una estructura ordenada y clara para tantos elementos.

En las siguientes figuras mostramos algunas de las páginas más destacadas de la aplicación. En concreto, podemos ver la página de bienvenida, que se muestra al usuario tras identificarse, donde se resumen los últimos cambios y se muestran las distintas pestañas que permiten acceder a las diferentes secciones que forman la aplicación (*Fig.* 49). Así mismo, también vemos un ejemplo del formulario de creación de recorridos (*Fig.* 50*)*, así como un buen ejemplo de las posibilidades que ofrece la

aplicación a la hora de navegar por las distintas escenas (*Fig.* 51). En esta última figura se ve como la escena cuenta con, en la posición capturada en la imagen, cinco *hotspots,* de estos cinco contamos con:

- Dos globos de información: proporcionan información sobre un establecimiento, edificio, monumento, etc. En este caso si clicamos en ellos vemos como la información es referente a los edificios a los que accedemos si pinchamos en las flechas de navegación verdes.
- Dos flechas de navegación verdes: las flechas de navegación de color verde indican que el destino es el interior de un edificio en vez de una zona exterior.
- Una flecha de navegación azul: las flechas de navegación de color azul indican que el destino es una zona exterior. Estas flechas son las más comunes.

Estos son solo algunos de los ejemplos de los distintos puntos que el usuario puede añadir a sus escenas. Algunos de los demás puntos que no incluimos en estos ejemplos son imágenes, vídeos o simplemente otros tipos de flechas de navegación.

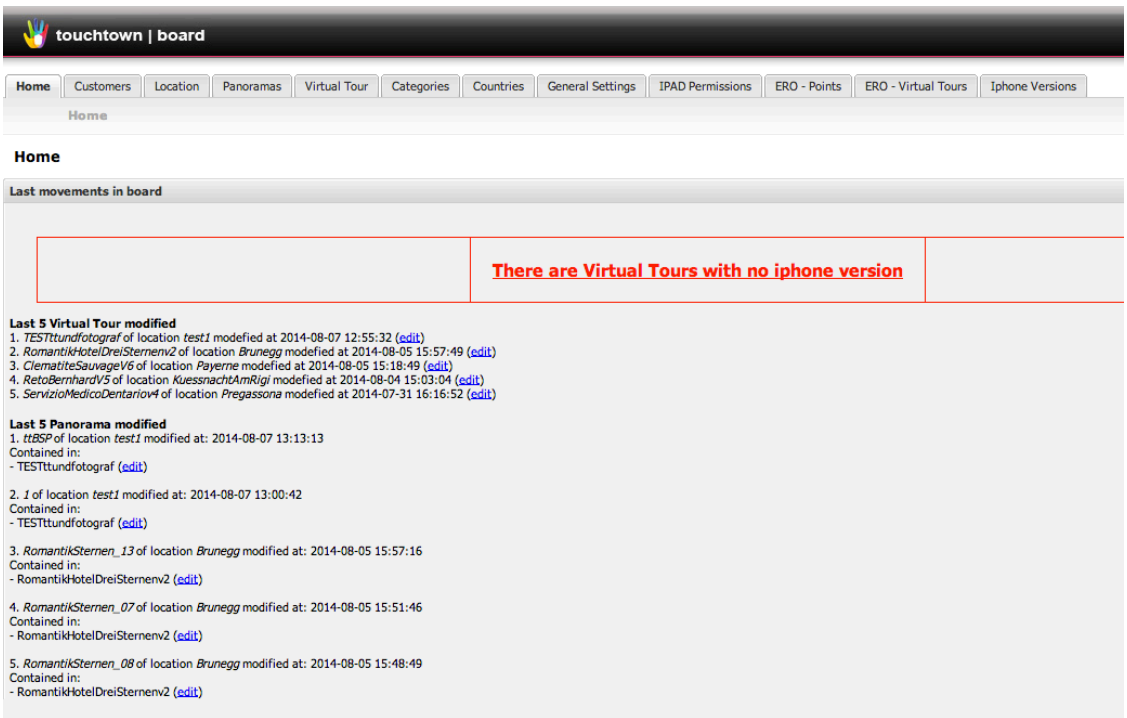

Figura 49: Pantalla de bienvenida del CMS

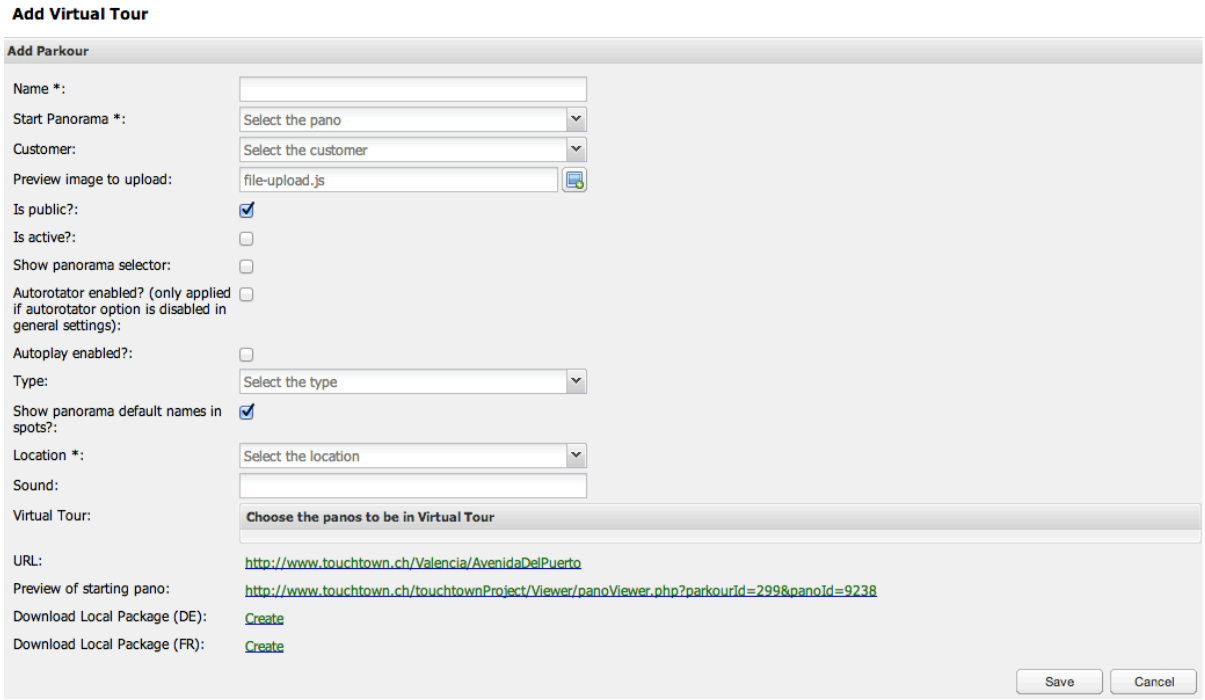

Figura 50: Formulario de creación de un recorrido

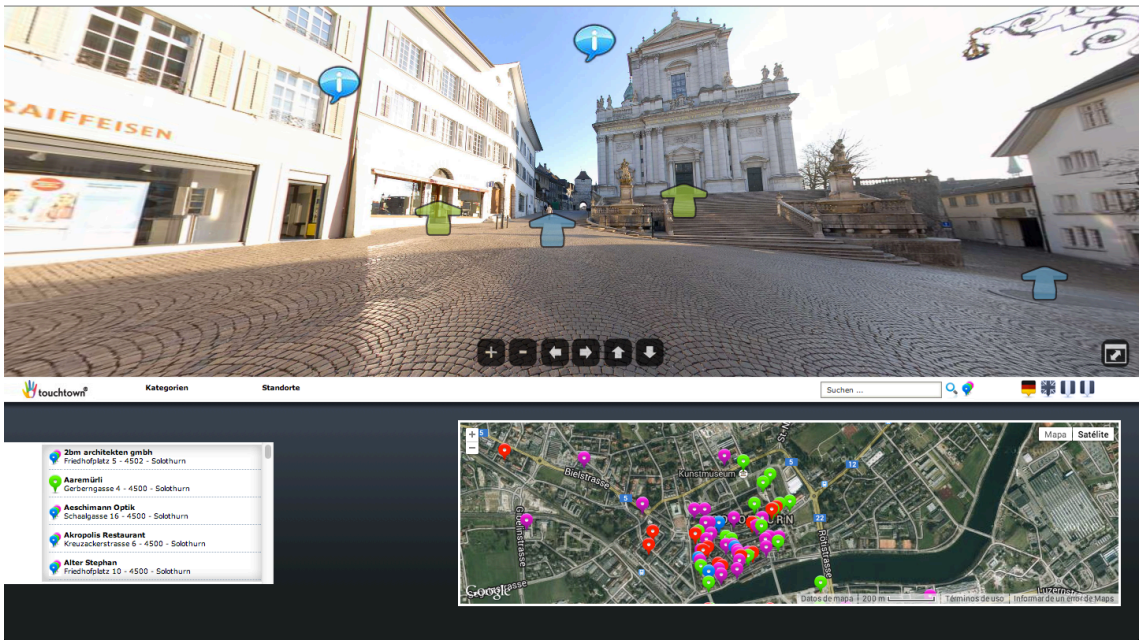

Figura 51: Visualización de una escena perteneciente a un recorrido de la ubicación Solothurn (Suiza)

4. Conclusión

Como se puede observar, una vez finalizada la versión inicial de la aplicación el resultado obtenido es altamente concordante con lo establecido en los requisitos e ideas iniciales del proyecto. A pesar de ello, y como ya hemos comentado previamente, debido a los plazos de entrega y las numerosas modificaciones y adiciones que se han realizado a lo largo del desarrollo, el resultado de la presentación de cara al usuario no ha quedado tan limpio como nos hubiera gustado pero es algo que se puede solucionar poco a poco.

En lo referente al desarrollo del proyecto, durante este nos hemos encontrado con distintos problemas de variada índole como, por ejemplo:

- Problemas con la instalación y configuración del servidor Apache en Mac OS X para trabajar de forma local, debido a incompatibilidades con el servidor nativo con el que cuenta el sistema así como los permisos de usuario.
- Pequeños inconvenientes con la manipulación de ficheros por parte de la aplicación debido a los permisos del sistema.
- La optimización del rendimiento del producto final.
- Control de las versiones de los ficheros del proyecto durante el desarrollo: a pesar de utilizar aplicaciones para ello (como comentamos al comienzo) no deja de ser una aplicación formada por muchos ficheros y directorios distintos a los que se realizan muchos cambios diariamente por lo que hay que ir con mucho cuidado a la hora de crear nuevas versiones y reemplazar ficheros, especialmente cuando más de una persona trabaja en el proyecto.
- Controlar la carga a la que somete la aplicación al servidor del entorno de producción. Especialmente importante para una aplicación de este tipo en la que se están generando y reemplazando ficheros constantemente en grandes volúmenes.

En cuanto a futuras ampliaciones y mejoras de la aplicación, pensamos que sería interesante realizar una versión muy simplificada del CMS y más adecuada para un usuario con menor experiencia mediante, por ejemplo, el *framework* “Bootstrap” de Twitter, la cual estaría dirigida a los terceros que compran los recorridos a nuestro cliente. De esta forma, éstos podrían acceder a un listado de los recorridos que les pertenecen y realizar cambios directamente ellos mismos fácilmente. Así mismo, y

como ya hemos comentado, nos gustaría mejorar la presentación de la aplicación, estructurando mejor el contenido (especialmente los formularios) y ofreciendo información que pueda ayudar al usuario final a utilizarla fácilmente.

Apéndice

A. Índice de figuras

B. Índice de siglas

En orden de aparición:

- HyperText Markup Language 5 (*HTML5*)
- eXtensible Markup Language (*XML*)
- Point of Interest (*POI*)
- Uniform Resource Locator (*URL*)
- Content Management System (*CMS*)
- Integrated Development Environment (*IDE*)
- PHP: Hypertext Preprocessor (*PHP*)
- Subversion (*SVN*)
- Portable Document Format (*PDF*)
- Shockwave Flash (*SWF*)
- Content Distribution Network (*CDN*)

C. Referencias y recursos

Referencias bibliográficas y electrónicas consultadas:

- Doyle, M. *Beginning PHP 5.3*. 1st edition. (2010). Indianapolis, IN (U.S.A): Wiley Publishing, Inc.
- krpano Gesellschaft mbH. (2014). *KRPano Documentation*. Obtenido de: http://krpano.com/docu/
- The PHP Documentation Group. (1997-2014). *Manual de PHP*. Obtenido de: http://php.net/manual/es/
- The Stack Exchange Network. (2014). *Stackoverflow*. http://stackoverflow.com/
- Sawyer McFarland, D. *JavaScript & Jquery: The Missing Manual*. 2nd edition. (Oct. 2011). U.S.A: O'Reilly Media, Inc.

Recursos *software* utilizados durante la realización del proyecto:

- NetBeans: https://netbeans.org/downloads/index.html
- XDebug: http://xdebug.org/
- XAMPP: https://www.apachefriends.org/es/index.html
- MySQL Workbench: http://www.mysql.com/products/workbench/
- Versions: http://versionsapp.com/
- KRPano: http://krpano.com/
- TCPDF: http://www.tcpdf.org/
- FileZilla: https://filezilla-project.org/
- Microsoft Office Visio: http://office.microsoft.com/es-es/visio/

- OmniGraffle: http://www.omnigroup.com/omniGraffle