Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Desarrollo de una aplicación web para la gestión de anuncios en múltiples portales inmobiliarios

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Vicente Ferrándiz Pardo

Tutor: Francisco Rodríguez Ballester

2014 / 2015

Resumen

Hoy en día las agencias inmobiliarias suelen disponer de un portal web en el que publicar las ofertas de venta y alquiler de sus inmuebles. Sin embargo, la aparición y el éxito de portales web inmobiliarios como Fotocasa, Idealista o Milanuncios han provocado que estas agencias también publiquen algunos de sus inmuebles en este tipo de portales inmobiliarios. Tanto el alta como la modificación y baja de estos anuncios supone un trabajo y esfuerzo muy alto para la agencias, puesto que no es posible realizar estas tareas de forma automática.

Este trabajo se centra en el diseño y desarrollo de una aplicación de escritorio que permita realizar de manera automática la publicación de los anuncios en una serie de portales objetivo accediendo directamente a los inmuebles almacenados en la base de datos de la agencia. Además de mantener la información sincronizada entre todos ellos, de manera que el usuario de la aplicación pueda saber de manera rápida y sencilla que inmuebles están anunciados en estos portales, en que portales está publicado y poder realizar modificaciones en todos ellos de manera simultánea.

Palabras clave: aplicación escritorio, java, http, MySQL.

Abstract

Nowadays, real estate agencies usually have a web portal where offer their real estates. However, the emergence and success of the real estate web portals like Fotocasa, Idealista or Milanuncios have caused these real estate agencies publish some of his real estates in such web portals. The creation, modification and elimination of these ads are a very high effort job for the real estate agencies, because it is not possible to perform these tasks automatically.

This work focuses on the design and development of a desktop application that allows users automatically posting ads in a number of target portals accessing directly to the real estates stored in the database of the real estate agency. In addition, it is responsible for maintaining information synchronized between them so that the users of the application can quickly and easy know which estates are published, in which portals and make changes to all of them simultaneously.

Keywords: desktop application, java, http, mysql.

Índice general

Índice de figuras

1. Introducción

En este primer capítulo se van a explicar cuáles han sido las motivaciones para la realización del proyecto así como la utilidad y necesidad de su desarrollo y los objetivos generales que este debe cumplir.

Por último se expondrá la estructura de la memoria indicando un pequeño resumen de cada uno de ellos.

1.1 Motivación

Las inmobiliarias han sufrido grandes cambios en su forma de vender sus inmuebles con la aparición de Internet. En un primer momento con páginas web propias donde poder exponer todos sus inmuebles en un catalogo accesible a todo el mundo; posteriormente con la creación de páginas web de anuncios inmobiliarios donde cualquier usuario puede poner a la venta sus propiedades. Es por esto que las inmobiliarias también hacen uso de este tipo de páginas debido a su popularidad y la difusión que esta conlleva.

Este tipo de páginas de anuncios suelen ser gratuitas, pero, normalmente limitan el número total de anuncios que se pueden publicar por cada cuenta. Para superar esta limitación, suelen existir cuentas especificas para las empresas que permiten publicar anuncios adicionales, pagando por cada uno de ellos.

Debido a esto, las inmobiliarias no pueden mantener en este tipo de páginas los anuncios de todos los inmuebles que tienen en su haber. Por ello, suelen seleccionar solamente un pequeño conjunto de inmuebles escogidos mediante una serie de criterios como los meses donde hay un crecimiento de venta de inmuebles, las zonas con más actividad inmobiliaria o inmuebles que por algún motivo especifico deben ser vendidos en un breve periodo de tiempo y se quiere aumentar su visibilidad.

Es por esto que las inmobiliarias realizan un gran número de cambios en que inmuebles están anunciados en este tipo de páginas de manera continua. El proceso de publicar, eliminar y modificar los anuncios es un proceso sencillo cuando existen un número reducido de páginas de anuncios y de inmuebles. Sin embargo, cuando esto no es así, la gestión de los anuncios puede volverse tediosa y consumir excesivo tiempo para los empleados de la inmobiliaria.

1.2 Objetivos

El trabajo tiene como objetivo principal diseñar y desarrollar una aplicación de escritorio que permita gestionar de una manera rápida, sencilla y centralizada todos los anuncios publicados en las páginas de anuncios objetivo.

La aplicación debe implementar las siguientes funcionalidades.

1. La aplicación debe permitir publicar y eliminar el anuncio de un piso concreto de manera sencilla y automática.
2. Los anuncios deben poderse actualizar en función de los cambios que se hayan producido en la base de datos de la inmobiliaria.
3. La aplicación debe ser extensible a cualquier página de anuncios que se desee.
4. La aplicación debe permitir añadir fotos a los anuncios de los inmuebles. Tanto localmente como procedentes de la web de la inmobiliaria.

Además de ser capaz de realizar estas funcionalidades, la aplicación debería presentar una interfaz amigable para el usuario y que permita gestionar todas las operaciones de manera intuitiva.

1.3 Estructura de la memoria

El presente trabajo se estructura en siete capítulos. A continuación se expondrá un breve resumen de cada uno de ellos.

- **Capítulo 1**. Se expone la motivación por la cual se ha llevado a cabo la aplicación, así como las funcionalidad y objetivos que debe abarcar.
- **Capítulo 2**. Se analizan las diferentes tecnologías que han sido empleadas para implementar la aplicación, así como las diferentes alternativas que fueron barajadas previamente.
- **Capítulo 3**. En este capítulo se realiza un análisis teórico de las características del proyecto, analizando los diferentes actores que pueden actuar sobre la aplicación y los diferentes casos de uso a los que se pueden enfrentar. Por último se describen los diferentes campos de formulario que han tenido que ser gestionados para realizar las operaciones.
- **Capítulo 4**. Este capítulo describe el diseño (la arquitectura software) empleada para implementar la aplicación.
- **Capítulo 5.** Describe con detalle la implementación realizada a partir de las tecnologías, el análisis y el diseño de la aplicación.
- **Capítulo 6.** En este capítulo se desarrolla de manera teórica el funcionamiento de la aplicación, describiendo el flujo de ejecución que se realiza a la hora de ejecutar las diferentes funciones que ofrece la aplicación. Finalmente se presentara un ejemplo práctico de la ejecución de la aplicación.

- **Capítulo 7.** El último capítulo recoge las conclusiones obtenidas a partir del desarrollo del trabajo; analizando si los objetivos planteados han sido completados y la utilidad de la aplicación en un entorno real. Además, se comentarán las posibles mejoras y trabajos que puedan derivar del mismo.

2. Tecnologías empleadas

En este capítulo expondremos las diferentes tecnologías que han sido empleadas para realizar el proyecto. Para ello se analizaran las características de estas y los motivos que han provocado su elección. Para acabar el apartado analizaremos otras posibles alternativas que finalmente fueron descartadas.

2.1 Lenguajes de programación

El lenguaje de programación que ha sido escogido para la implementación de la aplicación es el lenguaje orientado a objetos Java.

Este lenguaje ha sido escogido, entre otros factores, por tratarse de un lenguaje multiplataforma, es decir, por permitir a los desarrolladores escribir el código del programa una sola vez y que este pueda ser ejecutado en cualquier dispositivo sin necesidad de ser recompilado. Las aplicaciones java son ejecutadas directamente sobre la máquina virtual Java (JVM) sin importar la arquitectura de la computadora subyacente.

El programa desarrollado en este trabajo es una aplicación de escritorio, por lo que poder generar una aplicación única independiente al sistema operativo y hardware de la computadora nos decanto por un lenguaje como Java.

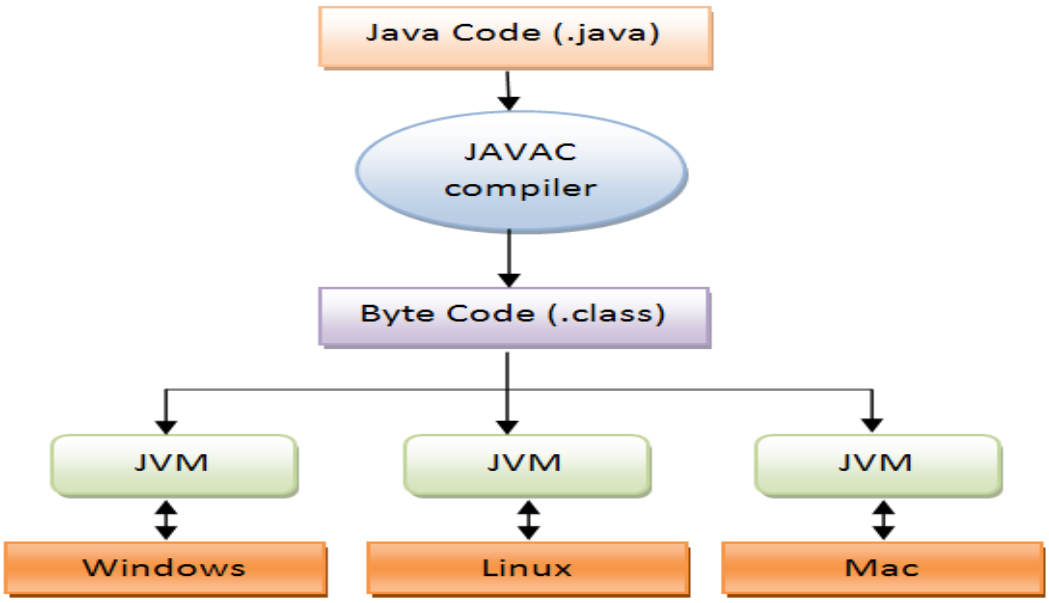

Figura 2.1 Funcionamiento de la máquina virtual Java

El otro motivo fundamental es que Java, pese a no ser un lenguaje de código libre, si permite desarrollar con el sin necesidad de adquirir una licencia o pagar por un software o entorno de desarrollo especifico. Este hecho lo diferencia de otros lenguajes de características similares como donde si tenemos que hacerlo.

2.2 Entorno de desarrollo

A la hora de seleccionar el entorno de desarrollo, se planteo la duda de decidir entre los dos entornos de desarrollo más conocidos y completos para desarrollar programas en Java, estos son Eclipse y Netbeans.

Como tanto Eclipse como Netbeans son de código libre, la elección se decantó atendiendo a características funcionales, ya que es ahí donde encontramos las principales diferencias. Netbeans presenta un entorno más agradable e intuitivo, además, proporciona por defecto una larga lista de *plugins* y módulos instalados que permiten al desarrollador empezar a trabajar inmediatamente una vez instalado el programa. Sin embargo, esto también afecta negativamente al rendimiento, a la personalización del espacio de trabajo y al control total del código, ya que al haber muchos elementos pre configurados no permite modificar al 100% el código del programa. Eclipse si nos permite configurar el ambiente de desarrollo con base a nuestras necesidades y modificar todo el código que el desarrollador genere. Además podemos añadir los *plugins* que se deseen como módulos independientes, permitiendo añadir solo aquello que vayamos a utilizar, afectando positivamente en el rendimiento final del proyecto.

El otro elemento principal que hace a Netbeans diferenciarse de Eclipse es su diseñador visual de interfaces de usuario para Java Swing. Sin embargo, existe un *plugin* para Eclipse llamado Window Builder que permite la creación de interfaces de usuario para aplicaciones Java de manera casi idéntica al diseñador visual que ofrece Netbeans en su instalación.

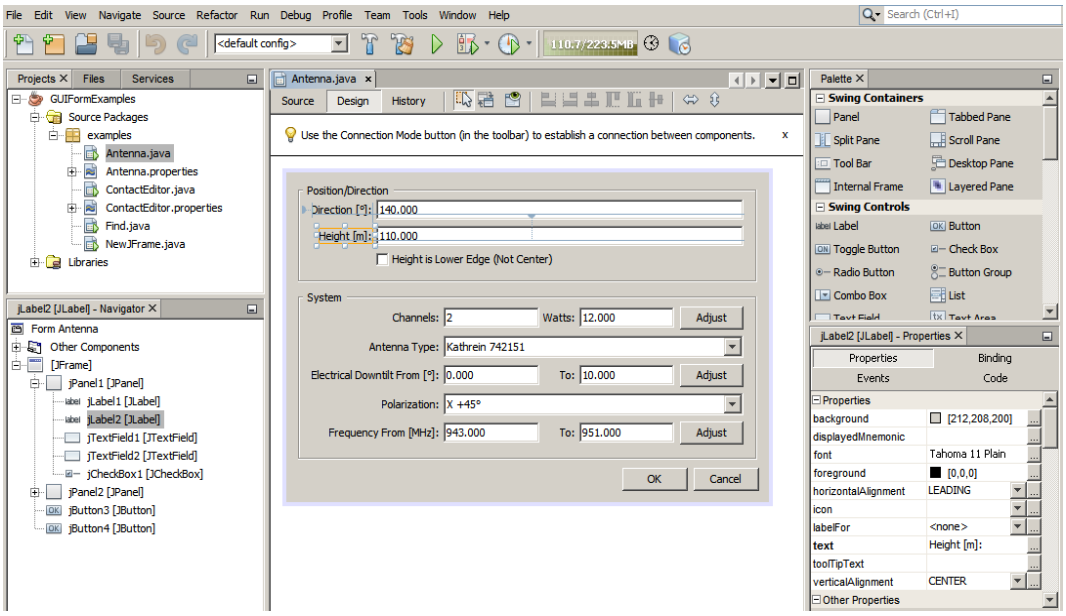

Figura 2.2 Diseñador de interfaces Netbeans

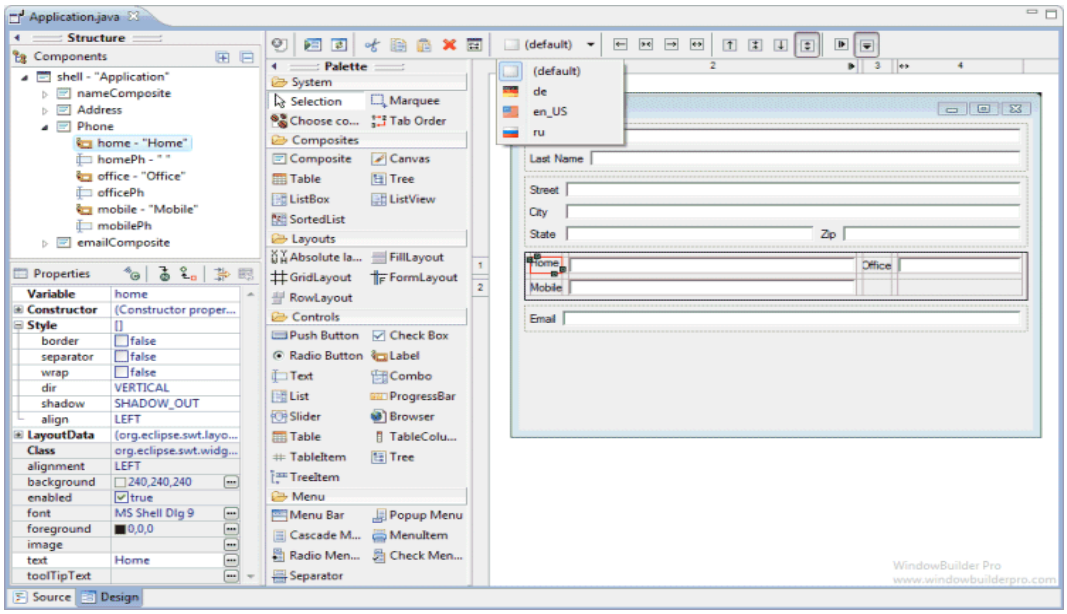

Figura 2.3 Diseñador de interfaces Window Builder para Eclipse

2.3 Base de datos

Para almacenar toda la información persistente que se utiliza en las diferentes operaciones que ejecuta la aplicación, se ha decidido utilizar una base de datos relacional.

Entre los diferentes gestores de bases de datos existentes, escogimos MySQL. MySQL es un software de gestión de bases de datos con licencia GNU y con unos requisitos muy asequibles para cualquier computador actual, lo que evita que el hardware donde esté instalado sea una limitación de la instalación.

Para instalar la base de datos MySQL se ha utilizado una instalación de XAMPP. XAMPP es un servidor web que consiste fundamentalmente en un servidor Apache, una base de datos MySQL e intérpretes para los scripts escritos en PHP y PERL.

Los motivos que decantaron la elección de XAMPP se deben a la facilidad de instalación y configuración de todos sus componentes, esto nos permitían tener instalada y configurada de una manera muy sencilla nuestra base de datos.

Al igual que con MySQL, XAMPP es un software gratuito y además multiplataforma. Esto nos permitía al igual que la aplicación, desplegarla independientemente del sistema operativo con la que trabajen los usuarios de la aplicación. Además, XAMPP proporciona una interfaz visual para la creación y modificación de las tablas, esto también facilita el trabajo respecto a tener que realizar las operaciones y consultas a través de la consola.

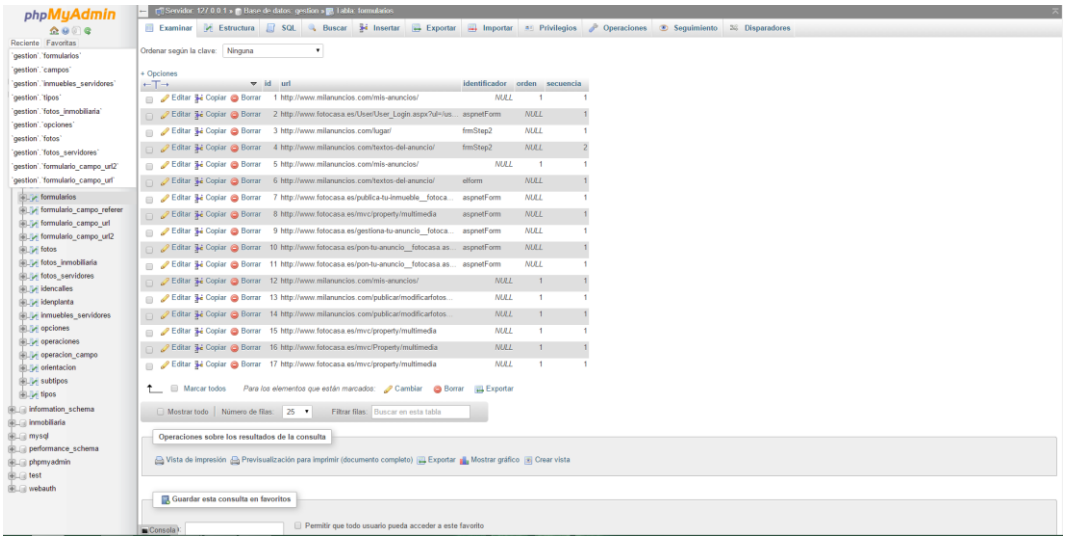

Figura 2.4 Interfaz de gestión de MySQL en XAMPP.

2.4 Posibles alternativas

En este apartado vamos a hablar de las diferentes alternativas que se barajaron respecto al lenguaje de programación con el que se ha desarrollado la aplicación, el tipo de aplicación, es decir, el porqué de una aplicación de escritorio frente una aplicación web y por último el porqué de una base de datos frente a otro tipo de métodos de proporcionar información persistente como ficheros XML o JSON.

En primer lugar vamos a hablar de la decisión de implementar una aplicación de escritorio frente a una aplicación web. Tal y como describe el titulo del trabajo la aplicación inicialmente iba a ser una aplicación web. La principal ventaja que ofrece esta solución es que permitía no tener que instalar o desplegar ningún tipo de software en el ordenador o ordenadores del usuario. Todos los usuarios podrían acceder a la web en la que se encontrase la aplicación y gestionar toda la información sobre los inmuebles y sus anuncios.

Existen tres motivos por los que finalmente no se desarrollo una aplicación web para la solución del trabajo.

El primer motivo se debe a la gestión de las operaciones HTTP y el envío de contenido multimedia al servidor. Pese a la existencia de una extensión llamada cURL que se encarga precisamente de la transferencia en PHP de contenido multimedia, no ofrece las mismas funcionalidades que ofrecen librerías como Java Net o Apache HTTP Client. Que ofrecen muchas más operaciones de transferencia de una forma más sencilla y robusta.

El segundo motivo es el mantenimiento de la sesión abierta entre peticiones, ya que en PHP no se encontró una extensión que solucionara de una manera sencilla este problema, sin embargo, de nuevo la librería Java Net lo soluciona de forma transparente y sencilla.

El tercer motivo es el análisis de las diferentes paginas de anuncios para encontrar valores en campos de formulario ocultos, imágenes o otra serie de elementos de la web. La librería JSOUP también soluciona el problema de una manera muy sencilla, permitiendo hallar elementos por etiquetas HTML, valores o por expresiones regulares *regex*.

Por último, vamos a hablar de la decisión de conservar toda la información persistente en una base de datos frente a ficheros XML o JSON. Pese a que los ficheros XML o JSON son más ligeros y permiten crear estructuras de datos que habrían suplido nuestra necesidad de almacenar información de control para completar los diferentes formularios requeridos para añadir un nuevo anuncio en las páginas web objetivo, se decidió el uso de la base de datos por dos factores.

- En primer lugar, ya que la mayoría de información para publicar los anuncios es consultada automáticamente de la tabla de inmuebles dentro de la base de datos de la inmobiliaria, era muchos más sencillo combinar esos datos con datos propios de la base de datos de la aplicación que utilizar otro tipo de medios como los mencionados anteriormente.
- En segundo lugar, al tener un servidor de base de datos, los datos de esta podrían ser consultados y editados por múltiples usuarios de manera simultánea, sin embargo, almacenar los datos en ficheros XML o JSON supondría tener copias de esta información en cada computadora utilizada.

3. Análisis

Una vez planteado el problema a resolver e identificadas las tecnologías que vamos a emplear para su implementación, en este capítulo analizaremos los diferentes actores que pueden intervenir en la aplicación , plantearemos un diagrama con los diversos casos de uso planteados a partir de la descripción del problema y describiremos en detalle cada uno de los casos de uso. Por último se analizaran los distintos tipos de campos utilizados en los formularios para adquirir la información. Esto permitirá comprender de manera más detallada el diseño de las tablas que se explicará en el capítulo 4.

3.1 Actores

En primer lugar vamos a identificar y describir el perfil del usuario o usuarios que van a interactuar con nuestra aplicación.

Dado que el objetivo de la aplicación es gestionar de forma automática los anuncios de los inmuebles en diversas páginas web, el actor principal que vamos a encontrar en nuestra aplicación es el de un empleado de la empresa.

No podemos distinguir otros actores o perfiles de usuario, ya que la aplicación está enfocada a un solo objetivo, es decir, no existen diferencias entre las funcionalidades que ofrece la aplicación entre un empleado o algún cargo superior de la empresa.

Por todo esto, podemos concluir que el actor principal y único actor de la aplicación es el empleado de la empresa.

Para la utilización de la aplicación, deben estar introducidas en la base de datos el nombre de usuario y las contraseñas de las diferentes cuentas de las páginas de anuncios. Además dichas cuentas deben estar validadas de manera manual previamente cuando sea preciso.

A partir de este momento, cualquier empleado de la inmobiliaria con acceso al ordenador o ordenadores donde este instalada la aplicación pueden insertar, borrar o modificar los anuncios de cada inmueble.

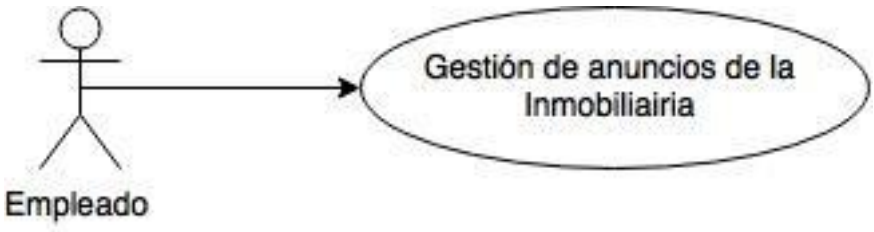

Figura 3.1 Diagrama general de la aplicación

En la Figura 3.1 podemos observar un diagrama general de la aplicación, como vemos, cualquier empleado puede acceder a la aplicación de gestión.

3.2 Diagrama de Casos de Uso

A partir de los requisitos planteados en las funciones que debe realizar la operación, se han analizado un conjunto de casos de uso que podemos observar en forma de diagrama en la figura 3.2.

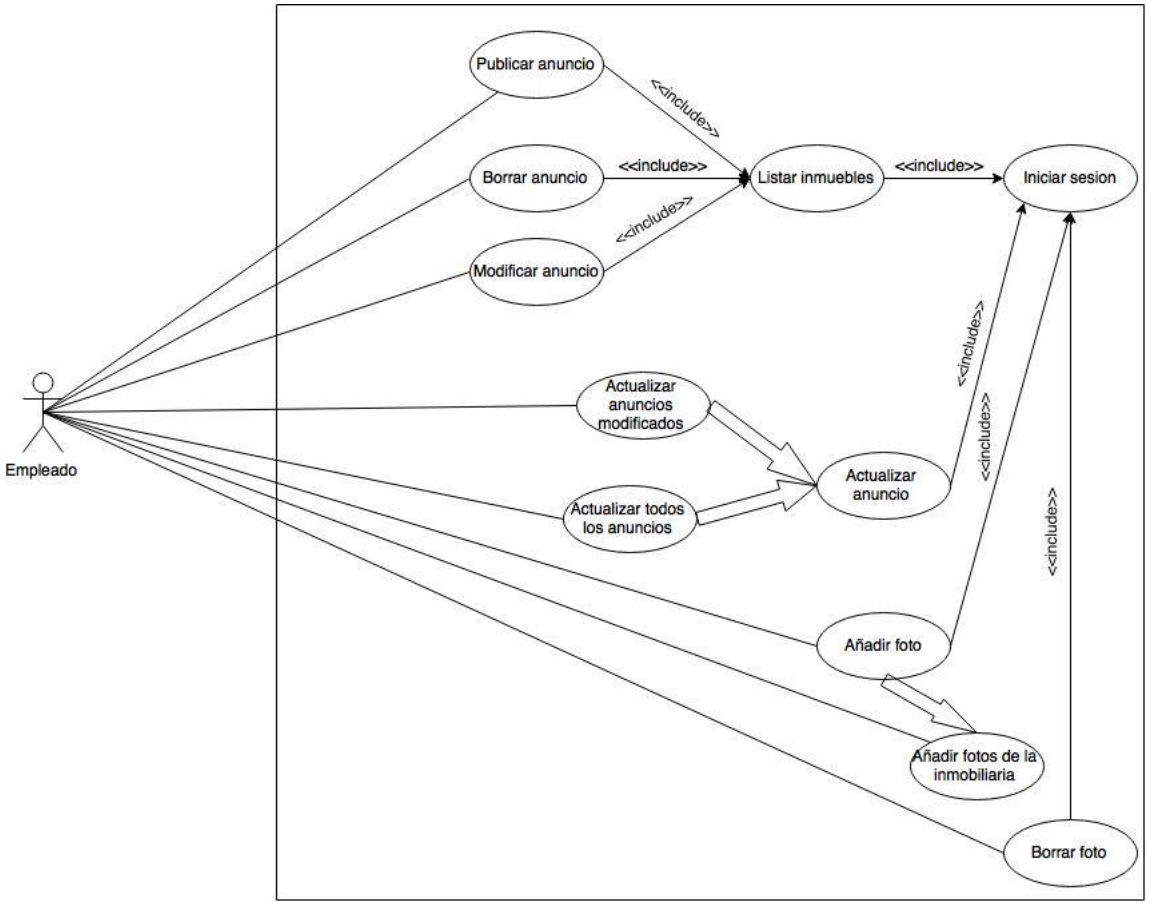

Figura 3.2 Diagrama de clases de la aplicación

3.3 Casos de Uso

3.3.1 Inicio de sesión

Actores	Empleado
Resumen	La aplicación envía las peticiones GET y POST pertinentes para iniciar sesión en las diferentes páginas web objetivo.
Precondiciones	-
Postcondiciones	El sistema mantiene la sesión iniciada en las diferentes páginas web.
Incluye	-
Extiende	-
Hereda de	-
Flujo de eventos	1. El empleado inicia la aplicación. 2. La aplicación envía las peticiones para iniciar sesión en las diferentes paginas.

3.3.2 Listar inmuebles

Actores	Empleado
Resumen	En la aplicación aparece un listado con todos los inmuebles almacenados en la base de datos de la inmobiliaria.
Precondiciones	-
Postcondiciones	-
Incluye	Iniciar sesión
Extiende	-
Hereda de	-
Flujo de eventos	1. La aplicación lista los inmuebles almacenados en la base de datos y los muestra en su interfaz.

3.3.3 Publicar anuncio

Actores	Empleado
Resumen	El empleado tiene una lista con los diferentes pisos de la inmobiliaria. Para un inmueble seleccionado, el empleado activa la casilla de verificación y el anuncio es publicado en la pagina indicada.
Precondiciones	La aplicación debe estar autenticada en las diferentes páginas web de anuncios que tomamos como objetivo.
Postcondiciones	El identificador del anuncio se almacena en la base de datos del sistema.
Incluye	Listar inmuebles
Extiende	-
Hereda de	-
Flujo de eventos	1. La aplicación inicia sesión en las diferentes páginas de anuncios. 2. La aplicación lista los inmuebles en la interfaz de la aplicación. 3. El empleado marca la casilla de verificación de una de las páginas de anuncios. 4. La aplicación obtiene la información del inmueble y publica el anuncio en la página seleccionada. 5. La aplicación almacena el identificador del anuncio en su base de datos.

3.3.4 Eliminar anuncio

Actores	Empleado
Resumen	El empleado tiene una lista con los diferentes pisos de la inmobiliaria. Para un inmueble seleccionado, el empleado desactiva la casilla de verificación y el anuncio es eliminado en la pagina indicada.
Precondiciones	La aplicación debe estar autenticada en las diferentes páginas web de anuncios que tomamos como objetivo.
Postcondiciones	El identificador del anuncio se almacena en la base de datos del sistema.
Incluye	Listar inmuebles

Extiende	-
Hereda de	-
Flujo de eventos	1. La aplicación inicia sesión en las diferentes páginas de anuncios. 2. La aplicación lista los inmuebles en la interfaz de la aplicación. 3. El empleado desmarca la casilla de verificación de una de las páginas de anuncios. 4. La aplicación obtiene el identificador del anuncio y lo elimina de la página web. 5. La aplicación elimina el identificador del anuncio en su base de datos.

3.3.5 Modificar anuncio

Actores	Empleado
Resumen	El empleado modifica algún campo o campos de uno de los inmuebles desde la interfaz de la aplicación, al pulsar el botón Entrar la información se actualiza en la base de datos de la inmobiliaria.
Precondiciones	-
Postcondiciones	-
Incluye	Listar inmuebles
Extiende	-
Hereda de	-
Flujo de eventos	1. El empleado edita un campo del inmueble. 2. La aplicación actualiza el inmueble en la base de datos.

3.3.6 Actualizar anuncio

Actores	Empleado
Resumen	El empleado actualiza los datos del anuncio con la nueva información del inmueble.
Precondiciones	La aplicación debe estar autenticada en las diferentes páginas web de anuncios que tomamos como objetivo.
Postcondiciones	-

Incluye	Iniciar sesión
Extiende	-
Hereda de	-
Flujo de eventos	1. El sistema obtiene la nueva información del inmueble de la base de datos y actualiza el anuncio.

3.3.7 Actualizar anuncios de inmuebles modificados

Actores	Empleado
Resumen	El empleado actualiza los anuncios de aquellos inmuebles cuyos valores han sido modificados desde la aplicación.
Precondiciones	Algún inmueble de la lista debe haber sido modificado previamente. La aplicación debe estar autenticada en las diferentes páginas web de anuncios que tomamos como objetivo.
Postcondiciones	-
Incluye	-
Extiende	-
Hereda de	Actualizar anuncio
Flujo de eventos	1. El empleado pulsa el botón de actualizar. 2. El sistema obtiene un listado de los inmuebles que han sido modificados. 3. El sistema obtiene la nueva información de cada inmueble de la base de datos y actualiza los anuncios.

3.3.8 Actualizar todos los anuncios

Actores	Empleado
Resumen	El empleado actualiza todos los anuncios publicados.
Precondiciones	La aplicación debe estar autenticada en las diferentes páginas web de anuncios que tomamos como objetivo.
Postcondiciones	-
Incluye	-
Extiende	-

Hereda de	Actualizar anuncio
Flujo de eventos	1. El empleado pulsa el botón de actualizar todos los inmuebles. 2. La aplicación selecciona todos los inmuebles, obtiene su información de la base de datos y actualiza uno a uno todos los anuncios existentes.

3.3.9 Añadir foto

Actores	Empleado
Resumen	El empleado puede añadir una nueva foto a los anuncios publicados.
Precondiciones	El anuncio tiene que estar publicado.
Postcondiciones	La foto se añade a la interfaz de gestión de fotos.
Incluye	Iniciar sesión
Extiende	-
Hereda de	-
Flujo de eventos	1. El empleado pulsa el botón subir fotos. 2. El empleado pulsa el botón añadir. 3. La aplicación abre un dialogo para elegir una imagen del sistema. 4. El empleado selecciona una fotografía almacenada en el sistema. 5. La aplicación almacena una copia de la imagen en su sistema de ficheros. 6. La aplicación sube la foto al servidor del anuncio y añade la foto al anuncio.

3.3.10 Borrar foto

Actores	Empleado
Resumen	El empleado puede borrar una foto de la lista de fotos del anuncio.
Precondiciones	El anuncio tiene que estar publicado. El anuncio debe tener al menos una foto.
Postcondiciones	La foto se elimina de la interfaz de gestión de fotos.
Incluye	Iniciar sesión
Extiende	-
Hereda de	-
Flujo de eventos	1. El empleado pulsa el botón subir fotos. 2. El empleado selecciona la foto que desea eliminar. 3. El empleado pulsa el botón borrar. 4. La aplicación elimina la copia de la imagen en su sistema de ficheros. 5. La aplicación elimina la foto del anuncio.

3.3.11 Añadir fotos de la inmobiliaria

Actores	Empleado
Resumen	El empleado puede añadir al anuncio, todas las fotos alojadas en la web de la inmobiliaria pertenecientes al
Precondiciones	El anuncio tiene que estar publicado.
Postcondiciones	Las fotos se añaden a la interfaz de gestión de fotos.
Incluye	-
Extiende	-
Hereda de	Añadir foto
Flujo de eventos	1. El empleado pulsa el botón añadir fotos de la inmobiliaria. 2. La aplicación descarga todas las imágenes del inmueble en la web de la inmobiliaria. 3. La aplicación almacena una copia de todas las imágenes descargadas en su sistema de ficheros.

	4. El sistema añade las fotos al anuncio.

3.4 Tipos de campos

A la hora de completar los formularios necesarios para realizar las diferentes operaciones planteadas a lo largo de la memoria, nos encontramos con diferentes tipos de campos, en este apartado se van a describir los que han sido necesarios para completar los formularios.

3.4.1 Campo de entrada de texto

Este tipo de campo requiere una entrada de texto por parte del usuario. En nuestra aplicación los textos se adquieren de valores almacenados en la base de datos de la inmobiliaria.

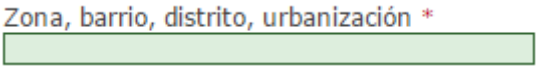

Figura 3.3 Ejemplo de campo de entrada de texto

3.4.2 Campos de lista de selección o cajas de control.

Los campos de lista o caja de control se rellenan con un valor numérico que equivale al texto seleccionado por el usuario desde el navegador. En nuestra aplicación las equivalencias numéricas están almacenadas en la base de datos de gestión de la aplicación.

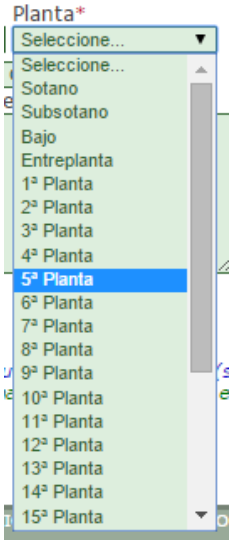

Figura 3.4 Ejemplo de lista de selección

```
<select class="inputs" name="playa" style="width:135px" tabindex="7">
<option value="60000">No está cerca</option><option value="60000">No está cerca</option>
<option value="10">unos 10 metros</option>
<option value="50">unos 50 metros</option>
<option value="100">unos 100 metros</option>
<option value="200">unos 200 metros</option>
<option value="300">unos 300 metros</option>
<option value="400">unos 400 metros</option>
<option value="500">unos 500 metros</option>
<option value="700">unos 700 metros</option>
<option value="900">unos 900 metros</option>
<option value="1000">un kilometro</option>
<option value="1500">1,5 kilometros</option>
<option value="3000">unos 3 kilometros</option>
<option value="6000">unos 6 kilometros</option>
<option value="10000">unos 10 kilometros</option>
<option value="20000">unos 20 kilometros</option>
<option value="30000">unos 30 kilometros</option>
<option value="40000">unos 40 kilometros</option>
<option value="50000">unos 50 kilometros</option>
```

Figura 3.5 Código fuente de una lista de selección

3.4.3 Campo ocultos

Los campos ocultos adquieren su valor por parte del navegador o por un script que lo genera. En nuestra aplicación el valor de estos campos suele ser extraido y no introducido, sin embargo, cuando hay que introducir el valor suele provenir de valores fijos almacenados en la base de datos de gestión de la aplicación.

```
<input type="hidden" name="c" value="4195484912">
```

Figura 3.6 Ejemplo de campo de formulario oculto

3.4.4 Campos generados en script

Los campos generados en script no forman parte como tal del formulario, pero suelen almacenar identificadores o redirecciones para formularios posteriores. En nuestra aplicación suelen ser campos consultados para adquirir el identificador de los inmuebles o fotografías.

```
var ifr = document.getElementById('ifr');
if(ifr) ifr.src = svr+'/fotos'+ad+'.php?esnuevo=s&idanuncio=165873254&clave=R148';
```

4. Diseño

En este capítulo vamos a describir los diferentes aspectos relacionados con el diseño tanto de la aplicación, donde explicaremos que arquitectura se ha utilizado y el diagrama de clases resultante a partir de las especificaciones, y el diseño de la base de datos.

Es esta sección vamos a hablar del diseño de la aplicación, es decir, la arquitectura software que vamos a emplear en el desarrollo de nuestra aplicación.

Según Philippe Kruchten, la arquitectura de software tiene que ver con el diseño y la implementación de estructuras de software de alto nivel. Es el resultado de ensamblar cierto número de elementos arquitectónicos de forma adecuada para satisfacer la mayor funcionalidad y requerimientos de desempeño de un sistema, así como requerimientos no funcionales, como la confiabilidad, escalabilidad, portabilidad y disponibilidad.

La arquitectura escogida para el diseño de la aplicación es la arquitectura multinivel en 3 capas. La arquitectura en 3 capas consiste en separar la lógica, la persistencia y la parte visible de la aplicación.

Este tipo de arquitectura permite la distribución de las diferentes capas en diferentes maquinas o procesos. También permite el desarrollo en paralelo en equipos de trabajo dada la independencia de cada una de las capas y como elemento más destacable la posibilidad de reutilizar todo el código desarrollado para una de las capas.

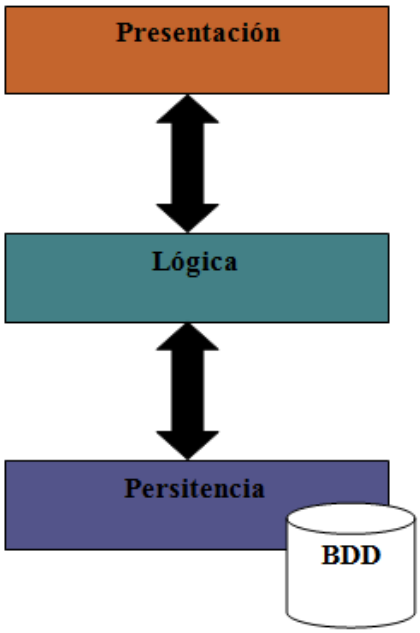

Figura 4.1 Esquema de la arquitectura en tres capas

4.1 Capa de presentación

La capa de presentación es la capa visible para el usuario. Se encarga de presentar y capturar la información que maneja el usuario. Como aparece en la figura 4.1 es la capa que interactúa con la capa de lógica.

La capa de presentación de nuestra aplicación se corresponde con la interfaz gráfica a través de la cual el empleado interactúa con las diferentes funciones que puede realizar la aplicación.

La librería que se ha utilizado para implementar la interfaz es la librería Java Swing. Swing incluye una serie de *widgets* especialmente creados para implementar interfaces gráficas de usuario como por ejemplo cajas de texto, listas desplegables, casillas de verificación, botones y tablas.

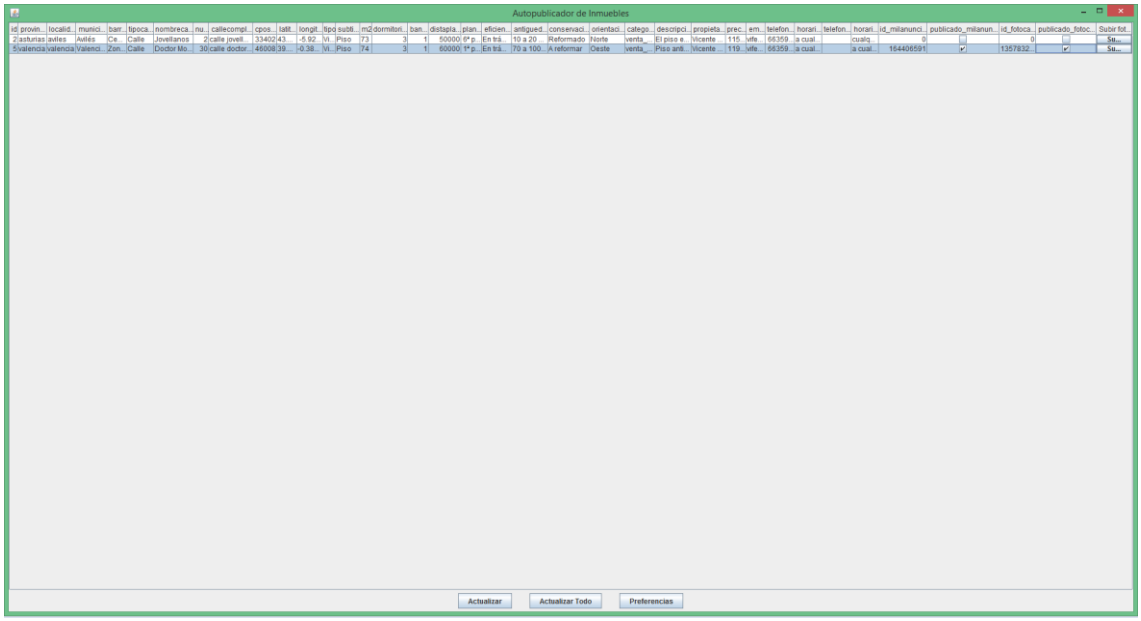

Figura 4.2 Ventana principal

En la figura 4.2 podemos observar la ventana principal de la aplicación. Los elementos principales que podemos observar en esta son:

Una tabla que ocupa la gran parte de la aplicación donde aparece un listado con los diferentes inmuebles que tiene almacenados en su base de datos la inmobiliaria que utilice la aplicación. Cada columna de la tabla se corresponde con un campo del inmueble en la base de datos, a excepción de las ultimas columnas. Por cada web de anuncios que este activada en la ventana de preferencias, aparecen dos columnas nuevas, una columna con una casilla de verificación que indica si el piso esta anunciado en la página de anuncios correspondientes, y una segunda columna con el identificador que tendrá el anuncio del inmueble en dicha pagina de anuncios. Además, la última columna incluye un botón para ir a la ventana de gestión de fotos de dicho inmueble.

En la parte inferior de la aplicación, encontramos tres botones, cada uno de ellos ofrece una funcionalidad nueva. En el caso del botón actualizar actualizamos todos aquellos anuncios de los inmuebles que hayan sido modificados. El botón actualizar todo permite volcar toda la información de los inmuebles almacenada en la base de datos en sus respectivos anuncios en el caso que los tengan. Esto permite actualizar todos los anuncios en caso de que haya habido grandes modificaciones en los inmuebles de la empresa modificados de manera externa a la aplicación para gestionar los anuncios.

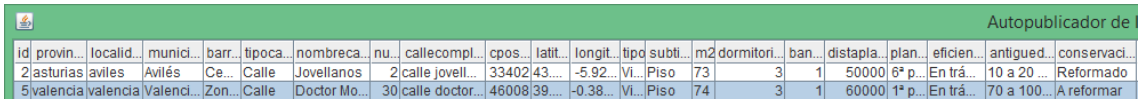

Figura 4.3 Detalle de la ventana principal 1

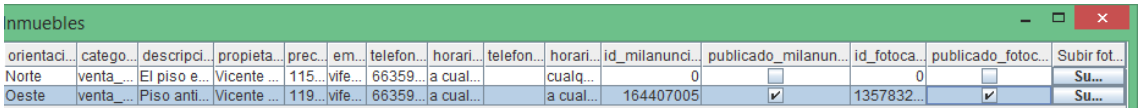

Figura 4.4 Detalle de la ventana principal 2

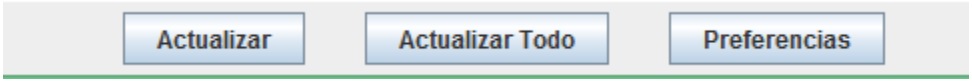

Figura 4.5 Detalle de la ventana principal 3

En las figuras 4.3, 4.4, 4.5 podemos observar detalles más cercanos de la ventana principal de la aplicación, podemos ver las diferentes columnas de la tabla, las columnas de las web de anuncios y el botón para la gestión de fotografías.

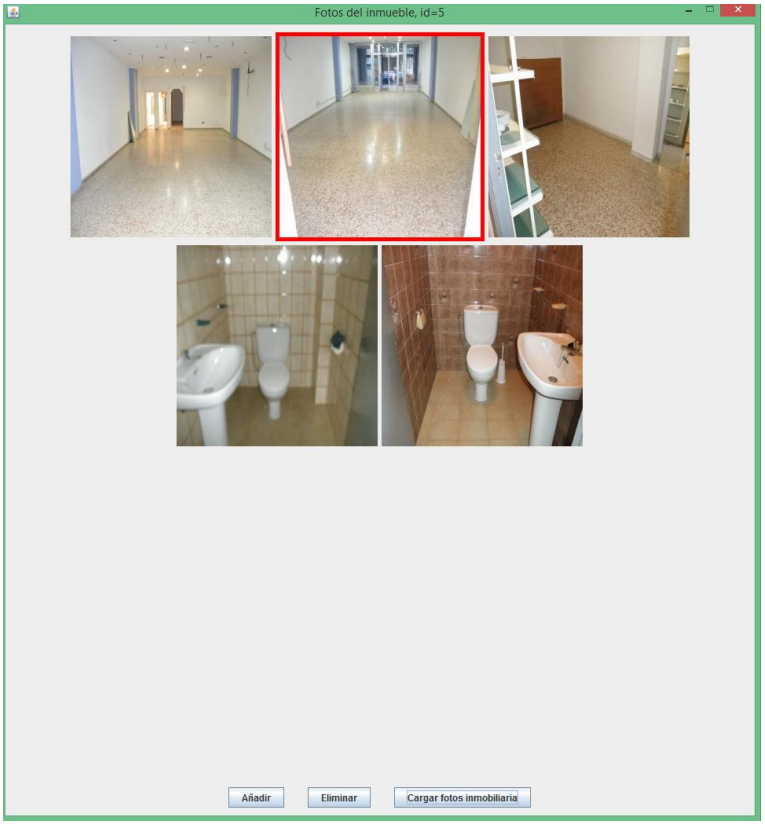

Figura 4.6 Ventana de gestión de fotografías

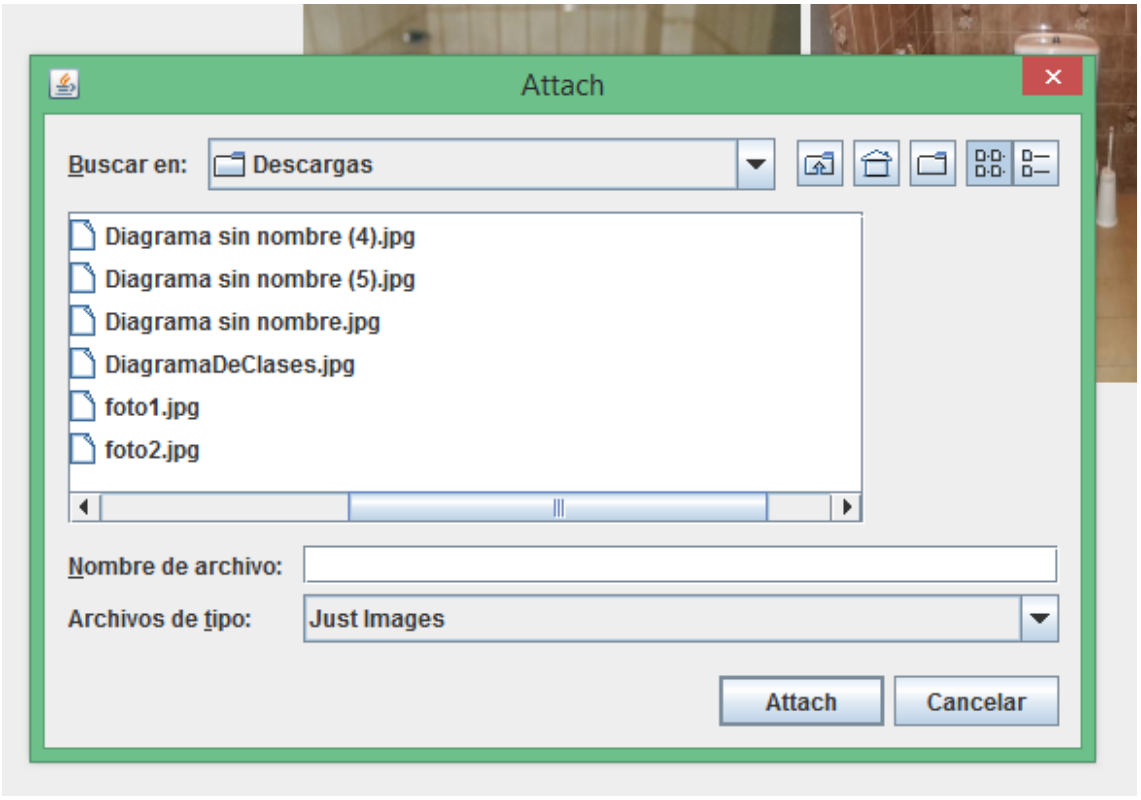

Figura 4.7 Dialogo de añadir fotos

La figura 4.6 muestra la ventana de gestión de imágenes del inmueble. Como indica el titulo de la figura, esta corresponde concretamente al inmueble con identificador número 5.

En él aparecen las diferentes imágenes pertenecientes al inmueble (si tiene). Tal y como se describió en los casos de uso, con el botón añadir podemos añadir nuevas fotos mediante una ventana de dialogo, que podemos observar en la figura 4.7. Si pulsamos el botón de eliminar, la foto seleccionada, es decir, la que tiene un recuadro rojo a su alrededor se elimina del anuncio, del sistema de ficheros y su referencia en la base de datos de la aplicación. El tercer y último botón permite descargar las fotos que tenga el inmueble en la página web propia de la inmobiliaria, descarga las fotos y las añade a la aplicación como si de una imagen añadida directamente al sistema se tratara.

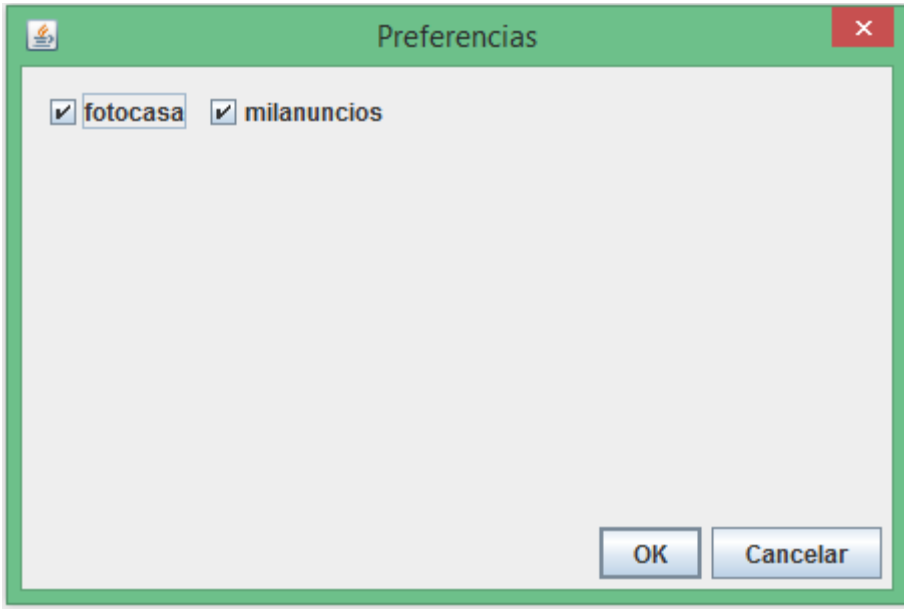

Figura 4.8 Ventana de preferencias

La última ventana que podemos encontrar en la aplicación es la ventana de preferencias, en ella podemos seleccionar con que paginas de anuncios queremos trabajar. Las páginas disponibles varían en función del número de páginas de anuncios que estén configuradas dentro de la base de datos. Una vez la página ha sido configurada, una nueva entrada aparece en la ventana de preferencias. El botón *OK* lee todas las casillas de verificación y en función de si están marcadas o desmarcadas añade o elimina las columnas especiales relacionadas con cada página de anuncios, es decir, la columna del identificador y la casilla de verificación para publicar o borrar el anuncio. El botón Cancelar cierra la aplicación sin guardar los cambios.

4.2 Capa de Lógica

La capa de lógica es el elemento central de la aplicación, en ella se encuentran todas las clases encargadas de realizar las operaciones de la aplicación, aparte se encarga de procesar los datos introducidos por el usuario y generar una respuesta y una salida a través de la capa de presentación.

La capa de lógica está dividida en tres partes. La primera se encarga, como se ha comentado en el párrafo anterior, de procesar la información introducida por el usuario a través de la interfaz de la aplicación. La segunda, es la encargada de realizar todas las operaciones relacionadas con la red, entre ellas se encuentra el envío de peticiones GET y POST para completar los formularios necesarias para iniciar sesión, dar de alta, borrar y modificar los anuncios. Además de realizar un *parse* a algunas páginas web para extraer valores ocultos o información dinámica. La tercera y última parte es la encargada de realizar todas las peticiones a la capa de persistencia para obtener toda la información persistente de la aplicación.

En el capítulo 5 describiremos con mucho más detalle las distintas clases que conforman la capa de lógica.

En la figura 4.9 podemos ver el diagrama de clases generado para la implementación de la aplicación.

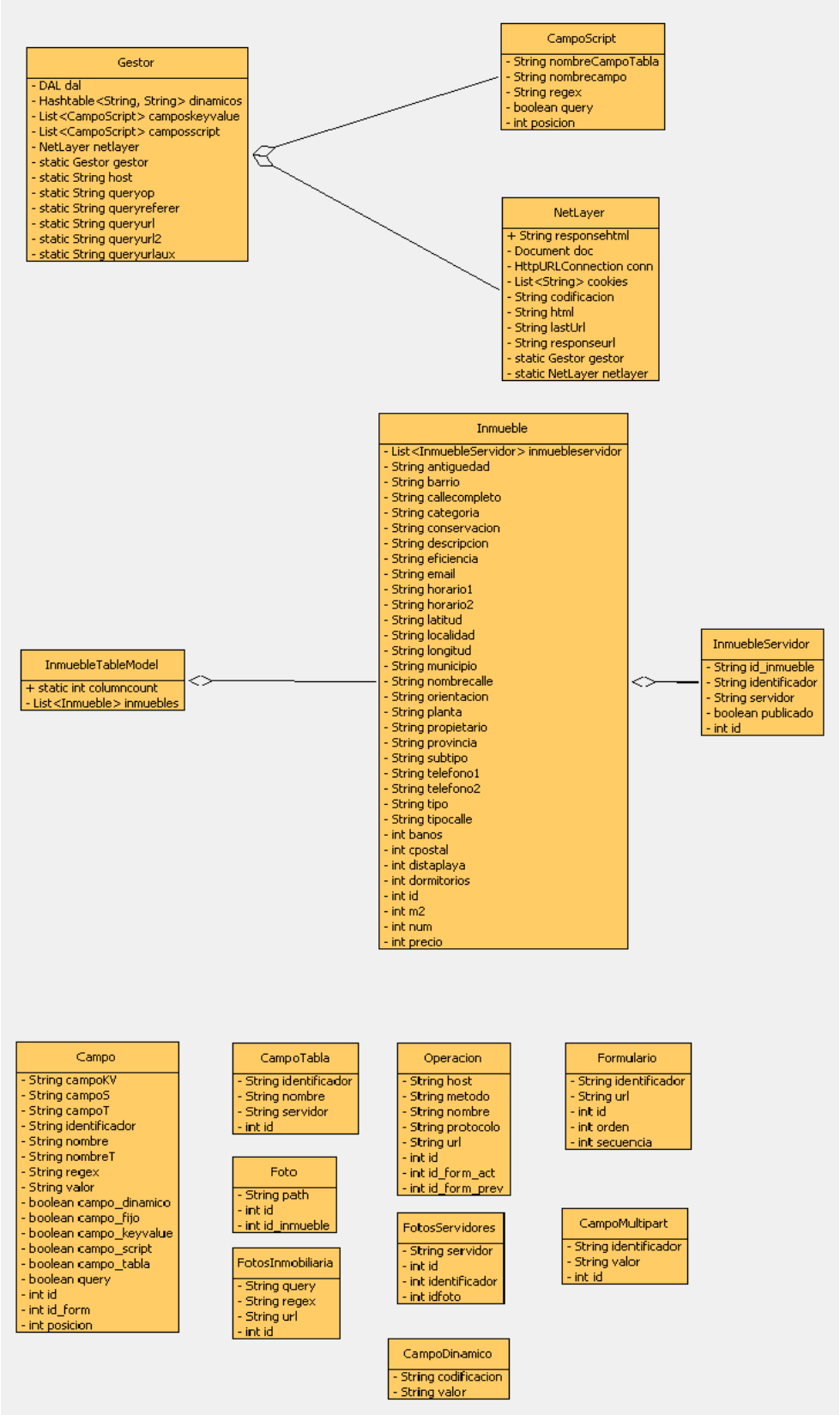

Figura 4.9 Diagrama de clases de la aplicación

4.3 Capa de presentación

La capa de persistencia es la capa que permite el acceso a los datos persistentes almacenados en una base de datos. Mediante las funciones disponibles en la librería MySQL para Java, podemos realizar todas las operaciones SQL que deseemos sobre la base de datos.

Como se describió en el capítulo 2, la base de datos de la aplicación es una base de datos alojada en un servidor web XAMPP. Toda la información para generar la interfaz de usuario, las preferencias y todos los datos requeridos para rellenar los diversos formularios requeridos para las operaciones de publicación, modificación o borrado de los anuncios.

Pese a que a lo largo de la memoria, se ha hablado de la base de datos instalada en XAMPP para almacenar toda la información persistente necesaria, la aplicación en realidad consume información de dos bases de datos diferentes. Esto se debe a que toda la información relacionada con los inmuebles, esta extraída directamente mediante consultas SELECT de la tabla de inmuebles almacenada en la base de datos de la inmobiliaria. De esta forma todas las modificaciones realizadas sobre los inmuebles de manera externa a la aplicación no afectan al correcto funcionamiento de esta. La segunda base de datos, la propia de la aplicación es la que si se ha descrito a lo largo de la memoria.

Como se ha mencionado en el párrafo anterior, de la tabla de la inmobiliaria se extrae la información sobre inmuebles tal como la dirección, el código postal, el numero de metros cuadrados, etc. Por tanto, de esta base de datos no se va a realizar un análisis detallado de las diferentes tablas y campos que pueda contener.

La implementación y detalles de la base de datos propia para la gestión de la aplicación, será descrita en el capítulo 5. En las figuras 4.10, 4.11, 4.12, 4.13 podemos observar el diagrama con la estructura de tablas de la base de datos, separado en varias figuras para poder analizarlo con más detalles.

La base de datos está formada por 22 tablas que se podrían dividir en 4 bloques, uno por cada figura de las mencionadas en el párrafo anterior.

El primer bloque (figura 4.9) está formado por tablas relacionadas entre sí encargadas de construir las *querys* necesarias para rellenar los formularios correspondientes a las diferentes operaciones que realiza la aplicación.

El segundo bloque está formado por tablas auxiliares encargadas de guardar preferencias o información de control de los anuncios, inmuebles y fotos necesarios para gestión.

El tercer bloque es el encargado de la gestión de las fotografías.

Finalmente, el cuarto bloque son una serie de tablas que almacenan equivalencias entre texto e identificadores numéricos.

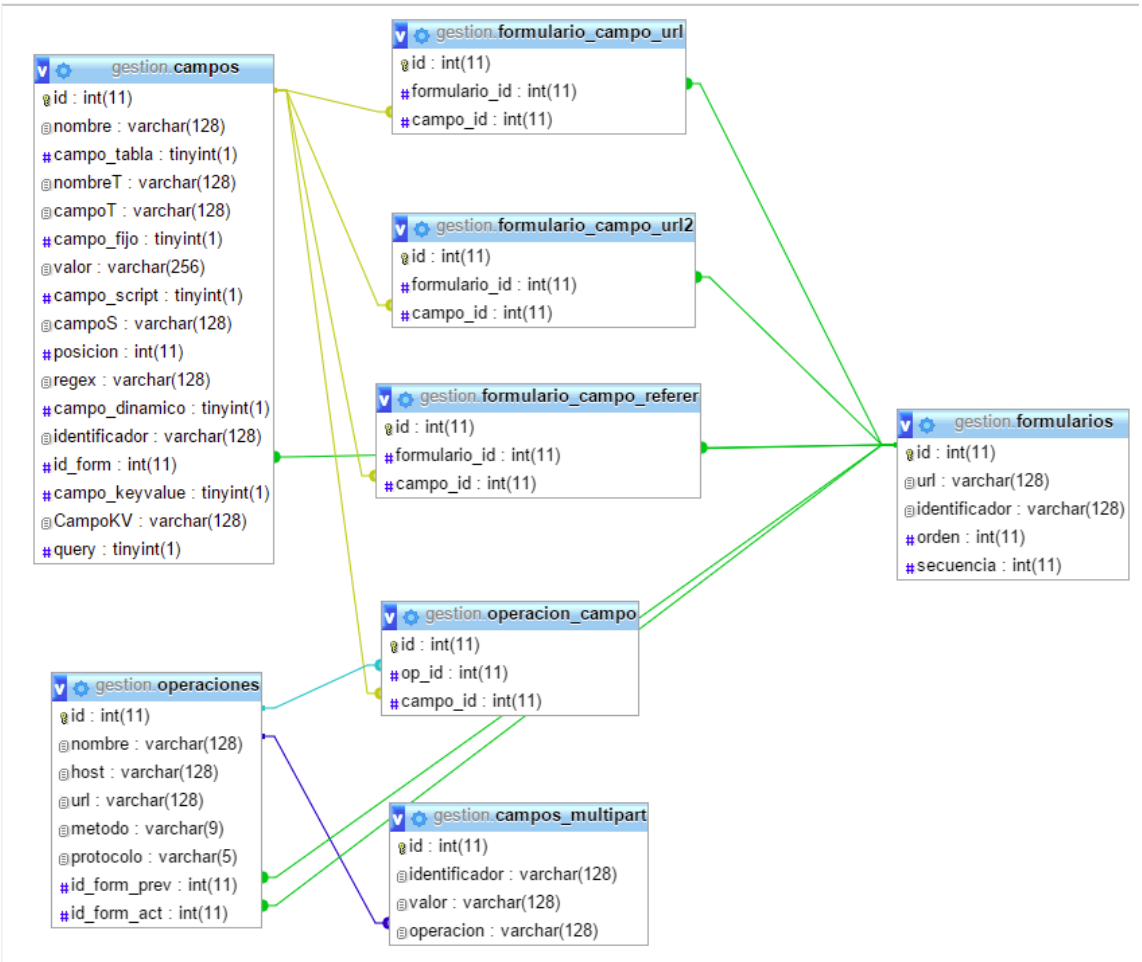

Figura 4.10 Diagrama de la base de datos 1

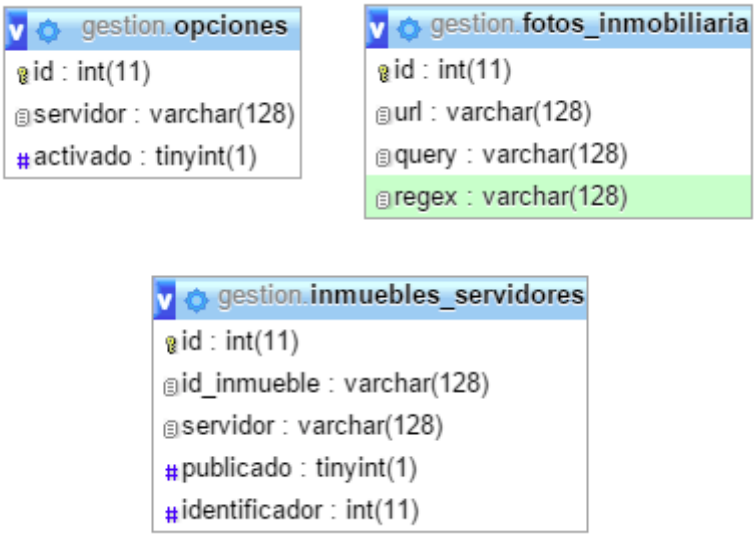

Figura 4.11 Diagrama de la base de datos 2

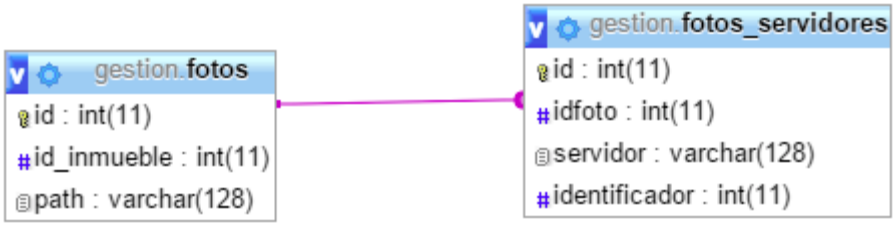

Figura 4.12 Diagrama de la base de datos 3

Figura 4.13 Diagrama de la base de datos 4

5. Implementación

En este capítulo se van a desarrollar de forma detallada en primer lugar las diferentes clases Java que conforman la capa de lógica de la aplicación y por último todas las tablas que componen la base de datos de gestión encargada de proporcionar los datos persistentes a la aplicación.

5.1 Clases de la aplicación

5.1.1 Campo

```
public class Campo

extends Object
```

Un objeto Campo representa un campo de un formulario.

Resumen de Campos	
private boolean	campo_dinamico
private boolean	campo_fijo
private boolean	campo_keyvalue
private boolean	campo_script
private boolean	campo_tabla
private String	campoKV
private String	campoS
private String	campoT
private int	id
private int	id_form
private String	identificador
private String	nombre
private String	nombreT
private int	nombre
private boolean	query
private String	regex
private String	valor

Resumen de Métodos	
String	getCampoKV() Devuelve el identificador de la variable en la que se encuentra almacenado el valor dentro de la URL.
String	getcampoS() Devuelve el identificador de la variable en la que se encuentra almacenado el javascript.
String	getcampoT() Devuelve el nombre del campo en el que se encuentra el valor del campo.
int	getForm_id() Devuelve el identificador del formulario en el que se encuentra el campo.
int	getId() Devuelve el identificador del campo.
String	getIdentificador() Devuelve el identificador del campo dinámico en el formulario.
String	getNombre() Devuelve el nombre el nombre del campo.
String	getNombreT() Devuelve el nombre de la tabla en la que se encuentra el valor del campo.
int	getPosicion() Devuelve el nombre en el que se encuentra el script que contiene el campo.
String	getRegex() Devuelve la expresión regular con la que localizar el valor del campo.
String	getValor() Devuelve el valor del campo fijo almacenado en la base de datos.
boolean	isCampo_dinamico() Indica si es un campo que se genera dinámicamente.
boolean	isCampo_fijo() Indica si el campo es un Campo con información estática.
boolean	isCampo_keyvalue() Indica si el campo proviene de un valor que forma parte de la url.
boolean	isCampo_script() Indica si el campo proviene de un valor generado por javascript.
boolean	isCampo_tabla() Indica si el Campo es un campo con información proveniente de un tabla.

boolean	**isQuery**() Indica si el campo debe ser incluido en la query.
void	**setCampo_dinamico**(int campo_dinamico) Establece si es un Campo que se genera dinámicamente.
void	**setCampo_fijo**(int campo_fijo) Establece si el campo es un Campo con información estática.
void	**setCampo_keyvalue**(int campo_keyvalue) Establece si el Campo proviene de un valor que forma parte de la url.
void	**setCampo_script**(int campo_script) Indica si el campo proviene de un valor generado por javascript.
void	**setCampo_tabla**(int campo_tabla) Establece si el Campo es un campo con información proveniente de un tabla.
void	**setCampoKV**(String campoKV) Establece el identificador de la variable en la que se encuentra almacenado el valor dentro de la URL.
void	**setCampoS**(String campoS) Establece el identificador de la variable en la que se encuentra almacenado el javascript.
void	**setCampoT**(String campoT) Establece el nombre del campo en el que se encuentra el valor del campo.
void	**setForm_id**(int form_id) Establece el identificador del formulario en el que se encuentra el campo.
void	**setId**(int id) Establece el identificador del campo.
void	**setIdentificador**(String identificador) Establece el identificador del campo dinámico en el formulario.
void	**setNombre**(String nombre) Establece el nombre el nombre del campo.
void	**setNombreT**(String nombreT) Establece el nombre de la tabla en la que se encuentra el valor del campo.
void	**setPosicion**(int nombre) Establece la nombre en el que se encuentra el script que contiene el campo.
void	**setQuery**(int query) Establece si el campo debe ser incluido en la query. 1 en caso afirmativo, 0 en caso contrario.

void	**setRegex**(String regex) Establece la expresión regular con la que localizar el valor del campo.
void	**setValor**(String valor) Establece el valor del campo fijo almacenado en la base de datos.

5.1.2 CampoDinamico

```
public class CampoDinamico

extends Object
```

Un objeto CampoDinamico representa un campo generado dinámicamente de un formulario.

Resumen de Campos	
private String	**codificacion**
private String	**valor**

Resumen de Métodos	
String	**getCodificacion**() Devuelve el tipo de codificación utilizada en el Campo.
String	**getValor**() Devuelve el valor del Campo dinámico.
void	**setCodificacion**(String codificación) Establece la codificación utilizada en el Campo.
void	**setValor**(String valor) Establece el valor del campo dinámico.

5.1.3 CampoMultipart

`public class` **`CampoMultipart`**

`extends Object`

Un objeto CampoMultipart representa un campo utilizado en las operaciones HTTP con ficheros multimedia.

Resumen de Campos	
private int	id
private String	identificador
private String	valor

Resumen de Métodos	
int	getId() Devuelve el identificador del campo en la base de datos.
String	getIdentificador() Devuelve el nombre con el que se identifica el campo en el servidor.
String	getValor() Devuelve el valor del Campo.
void	setId(int id) Establece el identificador del campo en la base de datos.
void	setIdentificador(String identificador) Establece el nombre con el que se identifica el campo en el servidor.
void	setValor(String valor) Establece el valor del campo.

5.1.4 CampoScript

```
public class CampoScript

extends Object
```

Un objeto CampoScript representa un campo generado por script de un formulario.

Resumen de Campos	
private String	nombrecampo
private String	nombreCampoTabla
private int	nombre
private boolean	query
private String	regex

Resumen de Métodos	
String	getNombrecampo() Devuelve el nombre del campo.
String	getNombreCampoTabla() Devuelve el identificador del campo en la tabla de la base de datos.
int	getPosicion() Devuelve el nombre del script en el que se localiza el campo.
String	getRegex() Devuelve la expresión regular con la que localizar el valor del campo.
boolean	isQuery() Indica si el Campo debe ser incluido en la query.
void	setNombrecampo(String nombrecampo) Establece el nombre del campo.
void	setNombreCampoTabla(String nombreCampoTabla) Establece el identificador del campo en la tabla de la base de datos.
void	setPosicion(int nombre) Establece la nombre del script en el que se localiza el campo.
void	setQuery(int query) Establece si el campo debe ser incluido en la query. 1 en caso afirmativo, 0 en caso contrario.

void	**setRegex**(String regex) Establece la expresión regular con la que localizar el valor del campo.

5.1.5 CampoTabla

```
public class CampoTabla
extends Object
```

Un objeto CampoTabla representa una fila de las tablas auxiliares.

Resumen de Campos	
private int	**id**
private String	**identificador**
private String	**nombre**
private String	**servidor**

Resumen de Métodos	
int	**getId**() Devuelve el identificador de la fila en la tabla.
String	**getIdentificador**() Devuelve el identificador del valor del campo.
String	**getNombre**() Devuelve el nombre descriptivo del campo.
String	**getServidor**() Devuelve la dirección del servidor al que pertenece el campo.
void	**setId**(int id) Establece el identificador de la fila en la tabla.
void	**setIdentificador**(String identificador) Establece el identificador del valor del campo.
void	**setNombre**(String nombre) Establece el nombre descriptivo del campo.
void	**setServidor**(String servidor) Establece la dirección del servidor al que pertenece el campo.

5.1.6 Formulario

```
public class Formulario

extends Object
```

Un objeto Formulario proporciona información sobre un formulario.

Resumen de Campos	
private int	id
private String	identificador
private int	orden
private int	secuencia
private String	url

Resumen de Métodos	
int	getId() Obtiene el identificador del formulario.
String	getIdentificador() Obtiene el identificador del formulario.
int	getOrden() Obtiene el orden del formulario en la web que lo contiene.
int	getSecuencia() Devuelve el orden en el que se accede a este formulario en una operación.
String	getUrl() Obtiene la URL en la que se encuentra el formulario.
void	setId(int id) Establece el identificador del formulario.
void	setIdentificador(String identificador) Establece el identificador del formulario.
void	setOrden(int orden) Establece el orden del formulario en la web que lo contiene.
void	setSecuencia(int secuencia) Establece el orden en el que se accede a este formulario en una operación.

void	**setUrl**(String url) Establece la URL en la que se encuentra el formulario.

5.1.7 Foto

```
public class Foto
```

```
extends Object
```

Un objeto Foto representa una foto perteneciente a un inmueble. En esta están incluidos todos los datos relevantes sobre la foto.

Resumen de Campos	
private int	**id**
private int	**id_inmueble**
private String	**path**

Resumen de Métodos	
int	**getId**() Devuelve el identificador de la foto en la base de datos.
int	**getId_inmueble**() Devuelve el identificador del inmueble al que pertenece la foto.
String	**getPath**() Devuelve la ruta parcial en la que se ubica la foto.
void	**setId**(int id) Establece el identificador de la foto en la base de datos.
void	**setId_inmueble**(int id_inmueble) Establece el identificador del inmueble al que pertenece la foto.
void	**setPath**(String path) Establece la ruta parcial en la que se ubica la foto.

5.1.8 FotosInmobiliaria

```
public class FotosInmobiliaria

extends Object
```

Un objeto FotosInmobiliaria representa una fila de la tabla FotosInmobiliaria. de la Base de Datos.

Resumen de Campos	
private int	id
private String	query
private String	regex
private String	url

Resumen de Métodos	
int	getId() Devuelve el identificador de la fila.
String	getQuery() Devuelve la query que acompaña a la URL.
String	getRegex() Devuelve la expresión utilizada para buscar las imágenes en la web de la inmobiliaria.
String	getUrl() Devuelve la url base de la inmobiliaria.
void	setId(int id) Establece el identificador de la fila.
void	setQuery(String query) Establece la query que acompaña a la URL.
void	setRegex(String regex) Establece la expresión utilizada para buscar las imágenes en la web de la inmobiliaria.
void	setUrl(String url) Devuelve la url base de la inmobiliaria.

5.1.9 FotosServidores

`public class` **`FotosServidores`**

`extends Object`

Un objeto FotosServidores representa una fila de la tabla FotosServidores de la Base de Datos.

Resumen de Campos	
private int	**id**
private int	**identificador**
private int	**idFoto**
private String	**servidor**

Resumen de Métodos	
int	**getId**() Devuelve el identificador de la fila.
int	**getIdentificador**() Devuelve el identificador de la foto en el servidor.
int	**getIdFoto**() Devuelve el identificador de la foto.
String	**getServidor**() Devuelve el servidor en el que esta publicada la foto.
void	**setId**(int id) Establece el identificador de la fila.
void	**setIdentificador**(int identificador) Establece el identificador de la foto en el servidor.
void	**setIdFoto**(int idfoto) Devuelve el identificador de la foto.
void	**setServidor**(String servidor) Establece el servidor en el que esta publicada la foto.

5.1.10 Gestor

```
public class Gestor

extends Object
```

La clase Gestor se encarga de gestionar todas las operaciones de acceso a la base de datos y la con••ion con el servidor.

Resumen de Campos	
private List<CampoScript>	**camposkeyvalue**
private List<CampoScript>	**camposscript**
private DAL	**dal**
private Hashtable<String>	**dinamicos**
private static Gestor	**Gestor**
private static String	**host**
private Netlayer	**netlayer**
private static String	**queryop**
private static String	**queryreferer**
private static String	**queryurl**
private static String	**queryurl2**
private static String	**queryurlaux**

Resumen de Métodos	
void	**actualizaIdentificadoresMilanuncios**(int id_inmueble) Disminuye en uno todos los identificadores de los servidores excepto el primero al borrar una foto del servidor Milanuncios.
void	**actualizarCampoInmueble**(int idpiso, String nombrecampo, String valor) Actualiza un campo de la tabla de inmuebles.
void	**actualizarFotoServidor**(int idfoto, String servidor, String Campo, String valor) Actualiza la información de la foto en el servidor.
void	**actualizarServidorPorInmueble**(int id_inmueble, String servidor, String nombrecampo, String valor) Actualiza los datos del inmueble en el servidor.

void	**borrarFoto**(String path) Elimina la foto del sistema de ficheros.
void	**borrarFotoDB**(int id_foto) Elimina la foto indicada de la base de datos.
void	**borrarFotoServidor**(int id_foto, String servidor) Elimina la información de la foto en los servidores de la base de datos.
List<CampoMultipart>	**buscarCamposMultipartPorOperacion**(String operacion) Devuelve una lista con los campos requeridos en la operación multipart.
FotosServidores	**buscarFotoServidor**(int idFoto, String servidor) Busca la información de la foto en el servidor indicado.
List<Foto>	**buscarFotosPorInmueble**(int id_inmueble) Busca el listado de fotos que pertenecen a un inmueble.
List<InmuebleServidor>	**buscarServidoresPorInmueble**(int id_inmueble) Devuelve una lista con los servidores en los que se encuentra publicado el inmueble.
String	**buscarValorCampoServidorInmueble**(int id_inmueble, String servidor, String nombrecampo) Devuelve el valor del Campo indicado.
String	**calculaRuta**(int idpiso) Calcula la ruta en la que ubicar el fichero.
void	**cambiarPreferencia**(String servidor, String preferencia) Modifica las preferencias almacenadas en la base de datos.
Hashtable<String,Boolean>	**cargarOpciones**() Carga las opciones de confirmación del programa.
void	**descargaFotosServidor**(int id_inmueble) Descarga las fotos del inmueble almacenados en la web.
String	**getValorCampo**(int id, String nombrecampo) Busca el valor de un Campo del inmueble proporcionado.
void	**insertarFotoServidor**(int idfoto, String servidor, String identificador) Inserta una fila en la tabla InmueblesServidores.

List<Inmueble>	**listarInmuebles**() Lista todos los inmuebles de la base de datos.
void	**realizarOperacion**(String nombreeop, Inmueble inmueble, Foto foto, int procedencia, String servidor) Realiza todo el proceso necesario para iniciar sesión, dar de alta, modificar y borrar los inmuebles.
void	**subIdFoto**(int id_foto, int id_inmueble, String path) Publica la foto indicada de la base de datos.
List<FotosInmobiliaria>	**tomarDatosFotosInmobiliaria**() Devuelve los datos necesarios para acceder a las fotos almacenadas en la web de la inmobiliaria.
int	**ultimoIdentificador**() Devuelve el identificador más alto de la lista de fotografías.

5.1.11 Inmueble

```
public class Inmueble

extends Object
```

Un objeto Inmueble representa un inmueble propiedad de la empresa. En este están incluidos todos los datos relevantes sobre el inmueble.

Resumen de Campos	
private String	**antiguedad**
private int	**banos**
private String	**barrio**
private String	**callecompleto**
private String	**categoria**
private String	**conservacion**
private int	**cpostal**
private String	**descripcion**
private int	**distaplaya**
private int	**dormitorios**
private String	**eficiencia**
private String	**email**
private String	**horario1**
private String	**horario2**
private int	**id**

private List<InmuebleServidor>	**inmuebleservidor**
private String	**latitud**
private String	**localidad**
private String	**longitud**
private int	**m2**
private String	**municipio**
private String	**nombrecalle**
private int	**num**
private String	**orientacion**
private String	**planta**
private int	**precio**
private String	**propietario**
private String	**provincia**
private String	**subtipo**
private String	**telefono1**
private String	**telefono2**
private String	**tipo**
private String	**tipocalle**

Resumen de Métodos	
String	**getAntiguedad**()
int	**getBanos**()
String	**getBarrio**()
String	**getCalleCompleto**()
String	**getCategoria**()
String	**getCodificacion**()
int	**getcPostal**()
String	**getDescripcion**()
int	**getDistaplaya**()
int	**getDormitorios**()
String	**getEficiencia**()
String	**getEmail**()
String	**getHorario1**()
String	**getHorario2**()
int	**getId**()
List<InmuebleServidor>	**getInmuebleservidor**()
String	**getLatitud**()

String	**getLocalidad**()
String	**getLongitud**()
int	**getM2**()
String	**getMunicipio**()
String	**getNombrecalle**()
int	**getNum**()
String	**getOrientacion**()
String	**getPlanta**()
int	**getPrecio**()
String	**getPropietario**()
String	**getProvincia**()
String	**getSubtipo**()
String	**getTelefono1**()
String	**getTelefono2**()
String	**getTipo**()
String	**getTipoCalle**()
void	**setAntiguedad**(String antiguedad)
void	**setBanos**(int banos)
void	**setBarrio**(String barrio)
void	**setCalleCompleto**(String callecompleto)
void	**setCategoria**(String categoria)
void	**setCodificacion**(String conservacion)
void	**setCpostal**(int cpostal)
void	**setDescripcion**(String descripcion)
void	**setDistaplaya**(int distaplaya)
void	**setDormitorios**(int dormitorios)
void	**setEficiencia**(String eficiencia)
void	**setEmail**(String email)
void	**setHorario1**(String horario1)
void	**setHorario2**(String horario2)
void	**setId**(int id)
void	**setInmuebleServidor**(List<InmuebleServidor> inmuebleservidor)
void	**setLatitud**(String latitud)
void	**setLocaidad**(String localidad)
void	**setLongitud**(String longitud)
void	**setM2**(int m2)
void	**setMunicipio**(String municipio)
void	**setNombrecalle**(String nombrecalle)
void	**setNum**(int num)

void	setOrientacion(String orientacion)
void	setPlanta(String planta)
void	setPrecio(int precio)
void	setPropietario(String propietario)
void	setProvincia(String provincia)
void	setSubtipo(String subtipo)
void	setTelefono1(String telefono1)
void	setTelefono2(String telefono2)
void	setTipo(String tipo)
void	setTipoCalle(String tipocalle)

5.1.12 InmuebleServidor

```
public class InmuebleServidor
```

```
extends Object
```

Un objeto InmuebleServidor representa una fila de la tabla Inmueble_Servidor de la Base de Datos.

Resumen de Campos	
private int	id
private String	id_inmueble
private String	identificador
private boolean	publicado
private String	servidor

Resumen de Métodos	
int	getId() Devuelve el identificador.
String	getId_inmueble() Devuelve el identificador del inmueble en la base de datos de la inmobiliaria.
String	getIdentificador() Devuelve el identificador del inmueble en el servidor.
String	getServidor() Devuelve el servidor al que representa la fila.

boolean	**isPublicado**() Indica si el inmueble esta publicado en el servidor.
void	**setId**(int id) Establece el identificador.
void	**setId_inmueble**(String id_inmueble) Establece el identificador del inmueble en la base de datos de la inmobiliaria.
void	**setIdentificador**(String identificador) Establece el identificador del inmueble en el servidor.
void	**setPublicado**(boolean publicado) Establece si el inmueble esta publicado en el servidor.
void	**setPublicado**(int publicado) Establece si el inmueble esta publicado en el servidor.
void	**setServidor**(String servidor) Establece el servidor al que representa la fila.

5.1.13 Netlayer

```
public class Netlayer

extends Object
```

La clase NetLayer se encarga de gestionar todas las operaciones de comunicación con los servidores.

Resumen de Campos	
private String	**codificacion**
private HttpURLConnection	**conn**
private List<String>	**cookies**
private org.jsoup.nodes.Doc••ent	**doc**
private static Gestor	**gestor**
private String	**html**
private String	**lastUrl**
private static Netlayer	**netlayer**
String	**responsehtml** String que almacena las respuestas html que devuelve el servidor al ejecutar una operación sobre ellas.

Resumen de Métodos	
CampoDinamico	**buscaValorCampoDinamico**(String urlform, String identificador, int numfoto) Este método busca el campo indicado en la web proporcionada como parámetro y obtiene el valor del Campo y su codificación.
String	**buscaValorCampoScript**(String nombre, int nombre, String regex) Este método busca el campo indicado en los scripts de la web proporcionada como parámetro y obtiene el valor del Campo y su codificación.
String	**buscaValorKeyValue**(String nombre)
void	**descargaFotosServidor**(int id_inmueble) Descarga las fotos del inmueble almacenados en la web.
String	**descargarWeb**(String url) Descarga en texto el código html de cierta página web.
void	**limpiarGlobales**() Reinicializa las variables globales de la capa de red.
void	**sendGet**(String host, String url, String referer) Este método envía una petición GET con query a una web con formulario.
void	**sendMultipart**(String rutaarchivo, String url, String nombreop, String queryop) Este método envía una petición POST con información multimedia al servidor.
void	**sendPost**(String host, String urloperacion, String referer, String query) Este método envía una petición POST con query a una web con formulario.

5.1.14 Operacion

```
public class Operacion

extends Object
```

Un objeto Operación proporciona información para ejecutar una operación HTTP / HTTPs sobre un formulario web.

Resumen de Campos	
private String	**host**
private int	**id**
private int	**id_form_act**

private int	id_form_prev
private String	metodo
private String	nombre
private String	protocolo
private String	url

Resumen de Métodos	
String	getPost() Devuelve el nombre del host en el que se ejecuta la operación.
int	getId() Obtiene el identificador de la operación.
int	getId_form_act() Devuelve el identificador del formulario perteneciente a la operación.
int	getId_form_prev() Devuelve el identificador del formulario predecesor a la operación.
String	getMetodo() Devuelve el método HTTP mediante el que se realiza la operación.
String	getNombre() Obtiene el nombre de la operación.
String	getProtocolo() Devuelve el protocolo por el que se realiza la operación web.
String	getUrl() Devuelve la URL a la que enviar la información de la operación.
void	setHost(String host) Establece el nombre del host en el que se ejecuta la operación.
void	setId(int id) Obtiene el identificador de la operación.
void	setId_form_act(int id_form_act) Establece el identificador del formulario perteneciente a la operación.
void	setId_form_prev(int id_form_prev) Establece el identificador del formulario predecesor a la operación.
void	setMetodo(String metodo) Establece el método HTTP mediante el que se realiza la operación.

void	**setNombre**(String nombre) Establece el nombre de la operación.
void	**setProtocolo**(String protocolo) Establece el protocolo por el que se realiza la operación web.
void	**setUrl**(String url) Establece la URL a la que enviar la información de la operación.

5.2 Tablas de la base de datos

5.2.1 Antiguedad

Esta tabla establece equivalencias entre las distintas antigüedades posibles y un identificador numérico.

- **Nombre**: Texto descriptivo del grado de antigüedad.
- **Identificador**: Valor numérico que representa el grado de antigüedad.
- **Servidor**: Indica a que página de anuncios pertenece la fila.

5.2.2 Campos

La tabla Campos es utilizada para representar los diferentes campos que podemos encontrar en un formulario web.

- **Nombre**: El nombre del campo.
- **Campo_tabla**: Booleano que indica si el valor del campo proviene de una tabla de la base de datos.
- **NombreT**: Nombre de la tabla en la que se encuentra el valor.
- **CampoT**: Nombre del campo de la tabla en el que se encuentra el valor.
- **Campo_fijo**: Booleano que indica si el valor del campo tiene un valor fijo.
- **Valor**: El valor fijo del campo.
- **Campo_script**: Booleano que indica si el valor del campo se encuentra en un script de una página web.
- **CampoS**: Identificador de la variable en la que se encuentra el valor del campo.
- **Posicion**: Posición en la que se encuentra el script dentro de la página web.
- **Regex**: Expresión regular necesaria para localizar la variable del script.
- **Campo_dinamico**: Booleano que indica si el valor del campo se genera dinámicamente en cada sesión.
- **Identificador**: El nombre identificativo del campo dentro del formulario de la página web.
- **Id_form**: El nombre identificativo del formulario en el que se encuentra el campo.
- **Campo_keyvalue**: Booleano que indica si el valor del campo se encuentra en la URL de respuesta.
- **CampoKV**: El identificador de la variable en la que se encuentra el valor del campo.

- **Query**: Booleano que indica si el campo del formulario tiene que ser incluido en la *query* de la operación.

5.2.3 Campos_Multipart

La tabla Campos_Multipart representa los campos necesarios para realizar una subida de contenido multimedia a un formulario web.

- **Identificador**: El nombre del campo.
- **Valor**: El valor del campo.
- **Operacion**: Nombre de la operación a la que pertenece el campo.

5.2.4 Categorias

La tabla Categorias establece una equivalencia entre el tipo de transacción inmobiliaria y un identificador numérico.

- **Nombre**: El nombre identificativo de la transacción inmobiliaria.
- **Identificador**: Valor numérico que representa la transacción inmobiliaria.
- **Servidor**: Indica a que página de anuncios pertenece la fila.

5.2.5 Conservacion

La tabla conservacion establece una equivalencia entre el estado de conservación y un identificador numérico.

- **Nombre**: El nombre identificativo del estado de conservación.
- **Identificador**: Valor numérico que representa el estado de conservación.
- **Servidor**: Indica a que página de anuncios pertenece la fila.

5.2.6 Eficiencia

La tabla eficiencia establece una equivalencia entre el tipo de transacción inmobiliaria y un identificador numérico.

- **Nombre**: El nombre identificativo de la transacción inmobiliaria.
- **Identificador**: Valor numérico que representa la transacción inmobiliaria.
- **Servidor**: Indica a que página de anuncios pertenece la fila.

5.2.7 Formularios

La tabla formularios almacena información sobre los diferentes formularios necesarios para realizar las diferentes operaciones que lleva a cabo la aplicación.

- **Url**: La dirección web en la que se encuentra el formulario.
- **Identificador**: El identificador del formulario.
- **Orden**: Establece la posición en la que se encuentra el formulario.
- **Secuencia**: Establece en qué orden se ejecuta el formulario dentro del conjunto de formularios involucrados en una operación.

5.2.8 Formulario_campo_referer

Esta tabla se encarga de relacionar que campos son necesarios para generar el *referer* necesario para completar un formulario vía HTTP.

- **Formulario_id**: Identificador del formulario.
- **Campo_id**: Identificador del campo.

5.2.9 Formulario_campo_url

Esta tabla se encarga de relacionar que campos son necesarios para completar un formulario.

- **Formulario_id**: Identificador del formulario.
- **Campo_id**: Identificador del campo.

5.2.10 Formulario_campo_url2

Esta tabla se encarga de relacionar que campos son necesarios para completar un formulario secundario.

- **Formulario_id**: Identificador del formulario.
- **Campo_id**: Identificador del campo.

5.2.11 Fotos

Esta tabla representa una imagen dentro del sistema de ficheros del ordenador que alberga la aplicación.

- **Id_inmueble**: El identificador del inmueble dentro de la base de datos de la inmobiliaria.
- **Path**: La ruta en la que se almacena la imagen en el sistema de ficheros dentro del ordenador que alberga la aplicación.

5.2.12 Fotos_inmobiliaria

Esta tabla almacena la información necesaria para obtener las imágenes pertenecientes a un inmueble dentro de la página web de la inmobiliaria.

- **Url**: La URL base de la inmobiliaria.
- **Query**: La *query* utilizada para acceder a los inmuebles.
- **Regex**: La expresión utilizada para encontrar las imágenes del inmueble dentro de la web.

5.2.13 Fotos_servidores

Esta tabla almacena información para gestionar las fotografías en los anuncios de las diferentes páginas web objetivo.

- **Idfoto**: El identificador de la foto en la tabla de fotos.
- **Servidor**: El nombre de la página de anuncios a la que pertenece la foto del anuncio.
- **Identificador**: El identificador de la foto dentro del anuncio.

5.2.14 Idencalles

La tabla idencalles establece una equivalencia entre el tipo de vía y un identificador numérico.

- **Nombre**: El nombre identificativo del tipo de vía.
- **Identificador**: Valor numérico que representa al tipo de vía.
- **Servidor**: Indica a que página de anuncios pertenece la fila.

5.2.15 Idenplanta

La tabla idenplanta establece una equivalencia entre la planta del piso y un identificador numérico.

- **Nombre**: El nombre identificativo de la planta.
- **Identificador**: Valor numérico que representa a la planta.
- **Servidor**: Indica a que página de anuncios pertenece la fila.

5.2.16 Inmuebles_Servidores

Esta tabla almacena la información necesaria para gestionar los diferentes anuncios de cada inmueble.

- **Id_inmueble**: El identificador del inmueble en la base de datos de la inmobiliaria.
- **Servidor**: Indica a que página web pertenece el anuncio.
- **Publicado**: Booleano que indica si al anuncio del inmueble esta publicado.
- **Identificador**: El identificador del anuncio.

5.2.17 Opciones

Indica que páginas web de anuncios se representan en la tabla de la aplicación.

- **Servidor**: El nombre de la página web de anuncios.
- **Activado**: Booleano que indica si la pagina debe mostrarse.

5.2.18 Operaciones

La tabla de operaciones almacena la información necesaria para realizar las diferentes operaciones que puede hacer la aplicación.

- **Nombre**: Texto descriptivo de la operación.
- **Host**: La URL del *host*.
- **Url**: Dirección web sobre la que realizar el método HTTP.
- **Metodo**: El método HTTP utilizado en la operación.
- **Protocolo**: El protocolo utilizado en la operación.
- **Id_form_prev**: Identificador con el formulario de la tabla formularios requerido para generar el *referer* de la operación.
- **Id_form_act**: Identificador con el formulario de la tabla formularios requerido para realizar la operación.

5.2.19 Operacion_Campo

Esta tabla relaciona que campos son necesarios para realizar una operación.

- **Op_id**: El identificador en la base de datos de la operación.
- **Campo_id**: El identificador en la base de datos del campo.

5.2.20 Orientacion

La tabla orientacion establece una equivalencia entre la orientación y un identificador numérico.

- **Nombre**: El nombre identificativo de la orientación.
- **Identificador**: Valor numérico que representa a la orientación.
- **Servidor**: Indica a que página de anuncios pertenece la fila.

5.2.21 Subtipos

La tabla subtipos establece una equivalencia entre el subtipo de inmueble y un identificador numérico.

- **Nombre**: El nombre identificativo del subtipo de inmueble.
- **Identificador**: Valor numérico que el subtipo de inmueble.
- **Servidor**: Indica a que página de anuncios pertenece la fila.

5.2.22 Tipos

La tabla tipos establece una equivalencia entre el tipo de inmueble y un identificador numérico.

- **Nombre**: El nombre identificativo del tipo de inmueble.
- **Identificador**: Valor numérico que el tipo de inmueble.
- **Servidor**: Indica a que página de anuncios pertenece la fila.

6. Funcionamiento de la aplicación

En este capítulo se describirá detalladamente el flujo de ejecución que sigue la aplicación a la hora de realizar las distintas operaciones que esta puede realizar. Es decir, analizando las acciones ejecutadas por el usuario en la capa de persistencia, las que realiza la capa de lógica de manera transparente al usuario y los diferentes accesos que realiza el usuario.

6.1 Descripción del funcionamiento

Nada más iniciar la aplicación, se muestra una tabla con celdas editables con los diferentes inmuebles que la inmobiliaria tiene en su base de datos.

Llegado a este punto el usuario puede editar las celdas para modificar los valores de sus campos, gestionar los anuncios del inmueble en las diferentes páginas de anuncios activadas o administrar las diferentes fotografías asociadas al inmueble.

El flujo de ejecución es idéntico para todas las operaciones, solamente varían los campos necesarios para realizar la operación dentro del servidor.

Por ello vamos a plantear un esquema de ejecución genérico para una mejor compresión al que nos referiremos como 'proceso general' a lo largo del capítulo:

1. El usuario selecciona una operación.
2. La capa de lógica solicita a la capa de persistencia todos los campos asociados a dicha operación, para ello se utiliza la tabla operacion_campo que relaciona los campos con la operación o operaciones a los que pertenece.
3. En función de donde se obtiene el valor del campo, se realizan unos procedimientos u otros.
    a. Si es un campo de valor fijo, se asigna el valor directamente.
    b. Si es un campo de valor proveniente de una base de datos, se accede a tabla que almacena el valor mediante la capa de persistencia y se obtiene.
    c. Si es un campo de valor generado por un script, se accede al script mediante las funciones de red ubicadas en la clase NetLayer, se realiza un *parse* de la pagina en la que se encuentra y se obtiene el valor.
    d. Si es un campo de valor generado dinámicamente en un formulario, se accede al formulario mediante las funciones de red ubicadas en la clase NetLayer, se realiza un *parse* de la pagina en la que se encuentra el campo y se obtiene el valor.
    e. Si es un campo de valor generado en la URL de respuesta a una operación, se accede a la URL y se obtiene el valor de esta.
4. Todos los campos se unen generando una *query*.
5. Una vez generada la *query*, se genera otra *query* equivalente para añadir el campo *referer* del método HTTP.

6. Una vez generadas ambas *query*, se construye el mensaje HTTP pertinente. Este es enviado al servidor mediante las operaciones POST o GET que ofrece la clase NetLayer para completar los formularios asociados a dicha operación.
7. Si es necesario, se analiza la respuesta para obtener campos como identificadores u otro tipo de datos de gestión.

Una vez planteado el esquema de ejecución de manera genérica, vamos a describir las diferentes acciones y procesos que se llevan a cabo para cada una de las funciones de la aplicación.

6.1.1 Alta de anuncio

El usuario marca la casilla de verificación para alguna de las páginas web de anuncios activas. Siguiendo el proceso general, se obtiene la *query* formada por diferentes campos de control y la información del inmueble procedente de la base de datos de la inmobiliaria. Con esta *query* se completa el formulario para dar de alta el anuncio. Una vez finalizado el proceso, se obtiene el identificador del anuncio de la respuesta y este se inserta o actualiza, en el caso de que exista, como una fila nueva en la tabla inmueble_servidores que relaciona los inmuebles con anuncios.

6.1.2 Eliminar anuncio

El usuario desmarca la casilla de verificación para alguna de las páginas web de anuncios activas. Siguiendo el proceso general, se obtiene la query que en este caso está formada por una serie de campos de control y el identificador del anuncio. Una vez el anuncio está eliminado, se actualiza el campo identificador de la tabla inmueble_servidores con un valor de 0.

6.1.3 Actualizar anuncios

Cuando el usuario modifica algún campo de los inmuebles listados en la tabla, el inmueble se añade a un listado con los inmuebles que han sido modificados. Si el usuario pulsa el botón de actualizar y la lista está vacía, se muestra un mensaje de advertencia que para actualizar los inmuebles primero debe haber algún campo modificado. En caso contrario, si la lista tiene algún inmueble, se construye un query idéntica a la utilizada para dar de alta a un anuncio, a excepción de que se añade un campo nuevo correspondiente al identificador del anuncio. Esto hace que el anuncio sea modificado con los valores que hayan cambiado desde la inserción del anuncio.

6.1.4 Actualizar todos los anuncios

El usuario pulsa el botón actualizar todos, la aplicación lista todos los inmuebles con anuncios publicados y actualiza dichos anuncios con la información que tenga almacenada en su base de datos en ese instante.

6.1.5 Subir, eliminar y descargar fotos

El usuario pulsa el botón subir foto de un inmueble. Si el inmueble tiene publicado algún anuncio, se una nueva pantalla encargada de gestionar las fotografías del inmueble. En caso contrario aparece un mensaje de alerta advirtiendo que debes publicar el anuncio antes de poder gestionar sus fotografías.

Una vez dentro de la ventana de gestión de las fotografías, podemos añadir fotos pulsando el botón añadir foto, que abrirá un dialogo para seleccionar una imagen local del ordenador que está ejecutando la aplicación. La fotografía será añadida a todos los anuncios del inmueble y se insertaran una o varias filas en la tabla fotos_servidores que almacena el identificador de la fotografía dentro del anuncio. Si existe al menos una foto, podremos seleccionarla desde la interfaz marcándola con un borde rojo. Con una imagen seleccionada podemos pulsar el botón eliminar que elimina la fotografía de todos los anuncios en los que se encuentre y elimina la fila correspondiente en la tabla fotos_servidores. Por último al pulsar el botón de descargar fotos de la inmobiliaria se descargan en local todas las fotos asociadas a un inmueble que se encuentren en la web. Cada una de estas fotos es procesada mediante el mismo sistema que las fotos añadidas manualmente a través de la aplicación.

6.1.6 Preferencias

Cuando el usuario abre la ventana de preferencias, la aplicación accede a la tabla opciones de la base de datos para leer que página de anuncios están disponibles. Para cada una de ella aparece una nueva casilla de verificación donde podemos marcar o desmarcar cada una de estas para determinar que páginas de anuncios queremos mostrar en la aplicación. Estos cambios también se registran en la tabla opciones modificando con 1 o 0 el campo activado.

6.2 Ejemplo: Inicio de sesión en Milanuncios

1. Se busca en la tabla operaciones la operación que tiene por nombre "login_ma".
2. Una vez identificada, se buscan todos los campos que se utilizan en la operación.
    a) SELECT c.* FROM `operacion_campo` oc JOIN `campos` c ON c.id = oc.campo_id WHERE `op_id` = 1
    b) En este caso el resultado de la consulta genera 4 resultados:
        i. comando
        ii. email
        iii. contra
        iv. rememberme
3. Ahora se analizan los campos resultantes comprobando que tipo de campos son, en este caso los cuatro son campos con valor fijo, por tanto se comprueba el campo valor de la tabla. Los resultados se van concatenando en lo que se denomina la *query* del mensaje.
    a) comando=login&email=viferpar.tfg%40gmail.com&contra=vpm5&rememberme=s
4. Construimos la *query* para el campo *referer* aplicando el mismo esquema, en este caso equivale a la *url* host, es decir, www.milanuncios.com.
5. Obtenemos la *url* a la que enviar la query, en algunos casos esta *url* lleva concatenada una tercera query que se genera como las otras dos. En este caso, http://www.milanuncios.com/cmd/.
6. Ahora comprobamos que protocolo se utiliza para completar el formulario, es decir, HTTP o HTTPS. En este caso, HTTP.

a) Una vez identificado, que tipo de operación. GET o POST. En este caso GET por lo que en vez de mandar la query como un elemento externo, se concatena a la url. En este caso, http://www.milanuncios.com/cmd/?comando=login&email=viferpar.tfg%40gmail.com&contra=vpm5&rememberme=s

7. La aplicación genera un mensaje HTTP GET con el host, el referer y la url final y realiza la operación.
8. Una vez completada, la web devuelve una página de respuesta que nos confirma que el inicio de sesión se ha completado con exito.

7. Conclusiones

En este último capítulo se plantearan las consideraciones obtenidas a partir del trabajo desarrollado para esto se realizará un repaso de los diferentes objetivos que se plantearon al inició de este proyecto y cuáles de ellos se han conseguido completar y con qué grado de acierto. Además, se explicaran las posibles ampliaciones que pueden llevarse a cabo en el proyecto con el objetivo de mejorar su funcionalidad o su aspecto visual.

7.1 Consideraciones finales

Haciendo un repaso de los objetivos que fueron planteados al inicio del proyecto y los resultados obtenidos, podemos concluir que la aplicación resuelve el problema presentado. La aplicación es capaz de visualizar de una manera esquemática e intuitiva los diferentes inmuebles pertenecientes a la inmobiliaria y observar cuáles de ellos han sido anunciados en las diferentes páginas web de anuncios. Además, permite publicar los nuevos anuncios y eliminar los existentes con marcar o desmarca una casilla, lo que permite a la inmobiliaria fluctuar los anuncios tanto como desee y de una forma muy sencilla.

El presente trabajo puede resultar útil en el ámbito inmobiliario, ya que permite a las inmobiliarias poder publicar una mayor cantidad de anuncios en el mismo tiempo necesitado para publicar un anuncio de forma manual.

7.2 Trabajo futuros

Como trabajo futuro a llevar a cabo a partir de este proyecto, podría realizarse una implementación totalmente nueva de la capa de presentación, es decir, un rediseño total de la interfaz. En este proyecto el objetivo principal era realizar una implementación eficiente que permitiese agilizar el proceso de gestión de los anuncios, por ello, se busco un diseño esquematizado y funcional de la interfaz que permitiese utilizar todas las funcionalidades de forma rápida. Sin embargo, una presentación más visual, detallada y usable, adaptada a los estándares de usabilidad y diseño actuales podría hacer la aplicación más atractiva y a la par usable.

Bibliografía

[1] HAYUN, AVI (2011) "Eclipse Window Builder vs Netbeans GUI Builder" en *ChaiPuter*, 23 de noviembre. <http://myblog.chaiware.org/2011/11/eclipse-window-builder-vs-netbeans-gui.html> [Consulta: 27 de abril de 2015]

[2] PRIETO SAEZ, N. et al. (2013) . *Empezar a programar usando java*. Valencia: Editorial UPV.

[3] PATRICIO SALINAS, C. (1996). *Actores y casos de uso*. <http://users.dcc.uchile.cl/~psalinas/uml/casosuso.html> [Consulta: 17 de mayo de 2015]

[4] KRUTCHEN, P. (1995). *The "4+1" View Model of Software Architecture*. Estados Unidos: IBM. <http://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf> [Consulta: 18 de mayo de 2015]

[5] CANALES MORA, R. (2003) "Documentación con Javadoc" en *Adictosaltrabajo*, 13 de julio. <http://www.adictosaltrabajo.com/tutoriales/javadoc/> [Consulta: 24 de febrero de 2015]

[6] W3Schools Organization. <http://www.w3schools.com/> [Consulta: 28 de enero de 2015]

[7] EMBnet (2010). *A quick guide to mysql. Tables and querys.* <http://www.embnet.org/sites/default/files/quickguides/guideMySQL.pdf> [Consulta: 3 de mayo de 2015]

[8] ORACLE. Creating a GUI With JFC/Swing. <http://docs.oracle.com/javase/tutorial/uiswing/> [Consulta: 5 de mayo de 2015]