Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación
Aplicaciones con Tecnología en Internet

Desarrollo de una aplicación web móvil multiplataforma para información de inmuebles basada en jQuery Mobile

Trabajo Especial de Grado
presentado ante la ilustre
Universidad Central de Venezuela
por el Bachiller
César Augusto León Anhuaman
C.I: 16.033.916

para optar al título de
Licenciado en Computación

Tutor
Prof. Sergio Rivas

Caracas, Julio / 2013.

Acta

Quienes suscriben miembros del Jurado, designado por el Consejo de Escuela de Computación, para examinar el Trabajo Especial de Grado presentado por el Bachiller César Augusto León Anhuaman, C.I. 16.033.916, con el título "Desarrollo de una aplicación web móvil multiplataforma para información de inmuebles basada en *jQuery Mobile*", a los fines de optar al título de Licenciado en Computación, dejan constancia lo siguiente:

Leído como fue, dicho trabajo por cada uno de los miembros del Jurado, se fijó el 29 de julio de 2013 a las 9:45 a.m., para que su autor lo defendiera en forma pública, se hizo en el Centro de Computación de la Facultad de Ciencias de la Universidad Central de Venezuela, mediante una presentación oral de su contenido. Finalizada la defensa pública del Trabajo Especial de Grado, el Jurado decidió aprobarlo con una nota de puntos, en fe de lo cuál se levanta la presente Acta, en Caracas al veintinueveavo día del mes de julio del año dos mil trece, dejándose también constancia de que actuó como Coordinador del Jurado el profesor Sergio Rivas.

Agradecimientos y Dedicatoria

Ha sido extenso el camino transitado pero también se ha contado con valioso apoyo a lo largo del mismo. Este trabajo de Grado se lo dedico a todas aquellas personas que compartieron experiencias conmigo durante la carrera y me apoyaron para poder seguir adelante. Hago especial mención a Maricruz Lista y Javier Mirabal quienes siempre estuvieron conmigo en cada etapa superada, para ellos un fuerte abrazo.

También dedico un profundo agradecimiento a la ilustre Universidad Central de Venezuela la cuál me dio la oportunidad de formarme como profesional y a los profesores por su honorable trabajo.

Oración

Dios dame la fuerza y el valor para no decaer ante los obstáculos de la vida, para no claudicar en mis esfuerzos y poder ser cada día un hombre mejor.

Cesar León.

Resumen

El presente Trabajo Especial de Grado aborda el tema de la búsqueda de información acerca de los inmuebles cargados en el sistema web Inmobilia.com. Para el caso de está investigación se desea permitir el acceso a la información de los inmuebles de manera ubicua, utilizando diferentes dispositivos móviles (celulares inteligentes, tabletas, iPods y cualquier otro dispositivo móvil que pueda utilizar un navegador web).

En el marco conceptual se investigó acerca las tecnologías móviles, mejores prácticas, tipos de dispositivos y una gran cantidad de aspectos importantes para tener una base sólida al momento de desarrollar la aplicación web móvil.

En el marco aplicativo se desarrolló la aplicación web móvil denominada *Inmobiliamobile* la cuál está basada en la tecnología *jQuery Mobile*. La aplicación se encuentra separada de Inmobilia.com desde donde se comparte la información de los inmuebles. Al ser una aplicación web para móviles se considera posible utilizar la misma en cualquier dispositivo que cuente con un navegador web. La aplicación se adapta automáticamente a la resolución de la pantalla y maneja eventos propios de los dispositivos móviles (*touch*, cambios de orientación, desplazamientos de la página, entre otras).

Palabras claves

Programación Extrema, tecnología, Internet, PHP, dispositivos móviles, web.

Contactos

- **César León:** cesaraugustoleon@gmail.com
- **Sergio Rivas:** sergiorivas@gmail.com

Índice general

Índice de figuras

Introducción

El constante desarrollo de la tecnología permite facilitar en varios sentidos las actividades que se realizan cotidianamente, al punto de permitir desarrollar las mismas de diferentes maneras. Tal es el caso de el acceso a la información, el cuál en la actualidad se puede realizar en algunos casos sin importar en donde nos encontremos ya que la mayoría de la gente posee algún medio para acceder a Internet.

La empresa Inmobilia.com posee una aplicación web destinada a la publicación de anuncios de inmuebles en Internet. Los administradores de esta empresa desean aprovechar las ventajas de la tecnología para poder ofrecer a sus usuario acceso ubicuo a la información de los inmuebles, por lo que se plantea la elaboración de una nueva aplicación web para móviles que permita el acceso de manera ubicua a sus usuarios.

El presente documento describe el proceso mediante el cuál se llevó a cabo el desarrollo de dicha aplicación, el método de desarrollo de software y las tecnologías utilizadas para este fin. El documento se encuentra estructurado de la siguiente manera:

- Parte I - Propuesta: En esta parte del documento se describe el contexto del problema a resolver y la solución propuesta para el mismo.
- Parte II - Marco metodológico: Comprende los conceptos teóricos básicos de las herramientas a utilizar para en el desarrollo de la aplicación. Se describe el método de desarrollo de software y las tecnologías utilizadas.
- Parte III - Marco aplicativo: En esta sección se describe detalladamente como se llevó a cabo el desarrollo de la aplicación web móvil, siguiendo los lineamientos propuestos por el método XP.
- Parte IV - Conclusiones y recomendaciones: El apartado final del documento incluye los últimos comentarios sobre las funcionalidades desarrolladas, así como las consideraciones y recomendaciones a tener en cuenta para el buen uso de la aplicación desarrollada.

Parte I

Propuesta

Capítulo 1

La empresa Inmobilia

En este capítulo se describe el contexto de la problemática que se desea solucionar con el presente Trabajo Especial de Grado por lo que se inicia describiendo la empresa Inmobilia:

La empresa Inmobilia, es conocida a nivel internacional como una empresa que provee a sus clientes (denominados Franquicias), varios medios de publicación para sus inmuebles y servicios. Entre los medios ofrecidos se encuentran la publicación gratuita de inmuebles a través de Internet, la publicación en revista digital y revista física de inmuebles y servicios. Esta empresa es propietaria y administradora de la aplicación web Inmobilia.com.

Inmobilia.com es una aplicación web destinada a la publicación de inmuebles, los cuáles pueden ser del tipo apartamentos, casas, oficinas, otros. La función principal de la aplicación es presentar a los usuarios toda la información del inmueble, incluyendo la información de su persona de contacto para que el mismo pueda realizar la negociación correspondiente.

1.1. El buscador de inmuebles

Para consultar la información de los inmuebles se utiliza el buscador principal. Para realizar una búsqueda los usuarios deben seleccionar los criterios de búsqueda que corresponda (estado, ciudad, tipo de inmueble, entre otras) y hacer clic en el botón *buscar*. A continuación se observa el buscador de Inmobilia.com y sus componentes:

Figura 1.1: Buscador de Inmobilia.com y sus componentes

A continuación se explica brevemente cada uno de los criterios de búsqueda:

- Estado: Se refiere a la entidad territorial o estado geográfico del país seleccionado por el usuario, ejemplo: Aragua, Anzoátegui, otros. Sólo se mostrarán entre los estados para seleccionar aquellos en los que haya existencia del inmueble a buscar.
- Ciudad: Se refiere a una entidad territorial del estado seleccionado por el usuario, ejemplo: Caracas, Guarenas, otras. Sólo se mostrarán entre las ciudades para seleccionar aquellas en los que haya existencia de inmuebles.
- Tipo de inmueble: Pueden ser apartamentos, casas, oficinas, entre otras. Sólo se mostrarán entre las opciones de los tipos de inmueble aquellos en los que haya existencia según el estado y ciudad seleccionado por el usuario.
- Código: Cada inmueble posee un código único que lo identifica. Si un usuario conoce el código del producto que desea buscar puede encontrar el mismo ingresando el código en el campo con el mismo nombre.
- Rango de precios: Se puede establecer el rango de precios por el que se desea buscar. Este criterio consta de dos campos, uno para el valor del precio mínimo y otro para el valor del precio máximo a buscar, colocando estos valores se limitará la búsqueda de inmuebles a sólo los que tengan el precio entre el rango mencionado.
- Metros de construcción: Permite limitar la búsqueda a sólo los inmuebles que cumplan con un rango de metros de construcción determinado, por ejemplo: 1-50 mt2, 51-100 mt2.
- Habitaciones: Estipula el número de habitaciones por el cuál se realizará la búsqueda.
- Baños: Estipula el número de baños por el cuál se realizará la búsqueda.

1.2. Información detallada del inmueble

A continuación se observa la sección para el detalle del inmueble:

Figura 1.2: Sección para el detalle del inmueble

En esta sección se presenta al usuario toda la información del inmueble: código, precio, ubicación, características, imágenes, videos ubicación con google maps, entre otras. Además se muestra la información de la persona de contacto, con esta información el usuario puede comunicarse con el vendedor para realizar la negociación de compra o alquiler, según sea el caso.

1.3. Descripción del problema

El continuo desarrollo de la tecnología de computadores ha hecho posible vivir en algunos aspectos, mejor de lo que se esperaba en el pasado. Desde acceder a masivas cantidades de información en Internet, a simplemente pasar un tiempo de entretenimiento en alguna actividad *online* con amigos. La tecnología informática continúa mejorando la calidad de vida tanto a nivel laboral como personal, y está alterando las características del manejo de la información, considerado como el mayor recurso de cualquier empresa, buscando agilizar y dar más confiabilidad a los resultados. Es por esto que se ha masificado el uso de los sistemas de información que contribuyen hoy en día con la evolución de la economía.

Esta situación de desarrollo ha impulsado nuevas maneras de realizar negocios, de presentar información y de desarrollar de software. Esto hace que las aplicaciones web deban estar en constante actualización para mantener su competitividad en el negocio al cuál se dedican.

Inmobilia es una empresa que se dedica a la publicación de inmuebles de todo tipo. Actualmente tiene 29 franquicias en 16 países en América y Europa. La organización no realiza transacciones de compra y venta de inmuebles, simplemente se desempeña como una empresa publicitaria que tiene sus propios medios de promoción y publicidad: revista Inmobilia y la aplicación web www.inmobilia.com.

La aplicación web denominada Inmobilia.com, está diseñada para ser utilizada a través de un computador personal. Los usuarios al momento de buscar la información de algún inmueble, deben ingresar a la aplicación a través de algún dispositivo no portable para poder navegar de manera cómoda y eficaz, en caso de que los usuarios no cuenten con alguno de estos dispositivos en el momento que necesite realizar la búsqueda, intentarán ingresar a la aplicación web utilizando algún dispositivo móvil (por ejemplo celulares inteligentes), como la aplicación web no se encuentra adaptada para la navegación a través de dispositivos móviles, se encuentra con una **reducida capacidad de acceso a la información de los inmuebles** en el momento y lugar en que lo necesita.

Esta limitación en el acceso a la información se debe a la falta de una aplicación que permita realizar esta actividad de manera ubicua. De continuar esta situación, la empresa corre el riesgo de reducir el número de usuarios activos en la aplicación Inmobilia.com por no ofrecer comodidades al usuario que sí podrían llegar a ofrecer otros sitios web que se dedican a negocios similares.

Por las razones antes expuestas se impone corregir esta situación mediante la implementación de una aplicación web móvil que suministre datos actualizados y oportunos sobre la existencia de inmuebles. Esta aplicación debe permitir a los usuarios realizar búsquedas de cualquiera de los inmuebles cargados en la misma. También se desea que la aplicación móvil permita la opción de compartir el contenido visualizado en las redes sociales *Facebook* y *Twitter* como una manera ampliar la cantidad de usuarios al que pudiera llegar la información, y finalmente mejorar la relación de los usuarios con la aplicación web.

1.4. Objetivo general

Implementar una aplicación web de Inmobilia.com para móviles que permita realizar la búsqueda de inmuebles.

1.5. Objetivos específicos

- Adaptar y aplicar el proceso de desarrollo de software Programación Extrema en la implementación de la aplicación web móvil de Inmobilia.com.
- Diseñar las interfaces gráficas de la aplicación que permita una visualización uniforme entre los dispositivos móviles
- Desarrollar una aplicación móvil que permita a los usuarios realizar búsquedas de inmuebles y además que esté integrada con las redes sociales *Facebook* y *Twitter*.
- Integrar la aplicación web móvil a desarrollar con la base de datos de la aplicación web ya existente de Inmobilia.com.
- Probar con distintos dispositivos móviles la aplicación web implementada.

Parte II

Marco conceptual

Capítulo 2

Herramientas tecnológicas

En este capítulo se describe detalladamente cada una de las herramientas tecnológicas utilizadas para desarrollar la aplicación web móvil de Inmobilia.com.

2.1. Modelo Cliente/Servidor

Cliente/Servidor [Hemmendinger, 1998], es un modelo distribuido de computación en el cuál aplicaciones desde un cliente solicitan servicios a un proceso servidor. Los clientes y servidores generalmente se encuentran en diferentes computadoras interconectadas por una red.

Una aplicación cliente es un proceso o programa que envía mensajes a un servidor por medio de la red. Esos mensajes solicitan al servidor la ejecución de una tarea específica, como buscar el registro de un cliente en una base de datos o devolver una porción de un archivo del disco duro del mismo.

El proceso o programa servidor espera por las solicitudes de los clientes, las cuáles son recibidas por medio de la red. El servidor recibe estas solicitudes y ejecuta acciones como consultas a base de datos o lectura de archivos. El proceso servidor generalmente se ejecuta sobre computadores con grandes capacidades de recursos.

Un ejemplo de un sistema Cliente/Servidor es una aplicación bancaria que permita el acceso a un usuario a la información de su cuenta, la cuál se encuentra en un servidor de base de datos. Todo el acceso ocurre por medio de una computadora personal. El cliente solicita información al usuario sobre la operación que desea realizar, valida los datos ingresados, transmite los datos al servidor de base de datos y muestra la respuesta del mismo al usuario.

2.2. Tecnologías Cliente/Servidor

La variedad de herramientas para el desarrollo web es muy extensa. Siguiendo el modelo de aplicaciones web de tres capas se pueden clasificar las herramientas tecnológicas en dos grandes grupos, estos son: las herramientas del lado del cliente y las herramientas del lado del servidor.

La principal diferencia entre estos grupos es el objetivo al cuál están destinados. Por parte del cliente se desea mostrar de manera organizada las información recibida del servidor, para lograr esto, el cliente se basa en tecnologías como: HTML, Javascript y Hojas de estilos. El servidor tiene como objetivo atender las peticiones realizadas por el cliente, realizando cualquier procesamiento que requiera el mismo y generando vistas con información requerida de base de datos las cuáles serán enviadas al cliente.

Se describe a continuación las herramientas a utilizar para este desarrollo web.

2.2.1. Tecnologías del lado del cliente

Las tecnologías del lado del cliente se refieren a las herramientas de software que utiliza el dispositivo internamente para poder trabajar con las aplicaciones web. A continuación se describen las mismas:

- HTML y HTML5

HTML es un lenguaje para escribir aplicaciones web [W3Schools, 2013, W3Schools, 2013]. Sus siglas significan Lenguaje de Marcado de Hyper Texto (Hyper Text Markup Language). Este lenguaje es un conjunto de etiquetas de marcado, donde las mismas describen el contenido del documento HTML. Los documentos HTML contienen etiquetas y texto plano.

Las **etiquetas HTML** son palabras claves (nombre de la etiqueta), encerradas como <html>. Usualmente las etiquetas se utilizan en pares como y , donde la primera etiqueta es el inicio de un elemento y la segunda es el final del mismo.

HTML5 es el nuevo estándar para HTML [W3Schools, 2013]. La versión anterior de HTML salió en el año 1999, desde ese entonces la web ha cambiado bastante. HTML5 está aún en desarrollo, sin embargo, los navegadores principales soportan varios de los nuevos elementos y APIs.

Algunas de las reglas establecidas para HTML5 son:

- Las nuevas funcionalidades deben basarse en HTML, CSS, DOM y Javascript.
- Reducir la necesidad de utilizar *plugins* externos (como *flash*).
- Mejorar el manejo de los errores.
- Mas marcado para reemplazar *scripts*.

- HTML5 debe funcionar correctamente independientemente del dispositivo que lo interprete.
- El proceso de desarrollo debe ser visible para el público.

Para definir un documento HTML5 basta con especificar el tipo del documento como <!DOCTYPE html>.

- CSS (Cascade Style Sheets)

CSS [W3C, 2013], es el lenguaje para describir la presentación gráfica de una aplicación web, incluye los colores, plantillas y fuentes. Permite a los desarrolladores adaptar la presentación para diferentes tipos de dispositivos, para pantallas grandes y pequeñas e incluso para impresoras. CSS es independiente de HTML y puede ser utilizado con XML. La separación entre HTML y CSS facilita el mantenimiento de las aplicaciones web, facilita compartir las hojas de estilos con otras aplicaciones y permite crear hojas de estilos para diferentes ambientes.

- JS (Javascript)

Javascript [W3Schools, 2013], es un lenguaje de programación, que puede ser insertado en páginas HTML para manipular objetos de la misma y mejorar la interacción del usuario con la aplicación. Javascript puede ser ejecutado por todos los navegadores modernos.

- AJAX (Asynchronous JavaScript and XML)

AJAX [W3Schools, 2013], es un grupo de técnicas de desarrollo web utilizadas en el lado del cliente (HTML, CSS, JS), que permite el intercambio de data con el servidor de manera asíncrona, es decir, en *background*, sin afectar el comportamiento de la página existente. Utilizando AJAX se pueden actualizar partes de la página sin tener que recargar la página completa.

- JQuery

jQuery [The jQuery Foundation, 2013], es una librería para manejo de Javascript rápida y concisa que simplifica la interpretación de los documentos HTML, la manipulación de eventos HTML, animaciones e interacciones AJAX para aumentar la velocidad del desarrollo web. jQuery está diseñada para cambiar la manera como se escribe el código Javascript.

- JQuery Mobile

Es un *framework* de desarrollo para aplicaciones web móviles [The jQuery Foundation, 2013, The jQuery Foundation, 2013], ideales para celulares inteligentes y tabletas, basado en

jQuery y soporta HTML5. Permite desarrollar aplicaciones web que pueden ser soportadas por todas las plataformas populares de los celulares inteligentes, (iPhone OS, Blackberry, Android y Windows Phone).

JQuery Mobile ha sido pensado para soportar una gran variedad de plataformas móviles. Este *framework* incluye un sistema de navegación con AJAX, el cuál provee animaciones en la transición de las páginas e incluye también un gran número de elementos gráficos reutilizables: páginas, ventanas de diálogo, barras de herramientas, listas de objetos, botones con íconos, elementos de formularios, acordeones entre otros.

Arquitectura de una aplicación con JQuery Mobile

JQuery Mobile provee una estructura simple para desarrollar aplicaciones utilizando HTML5 como se puede ver a continuación:

```
<!DOCTYPE html>
<html>
<head>
        <title>My Page</title>
        <meta name="viewport" content="width=device-width, init
        <link rel="stylesheet" href="http://code.jquery.com/mob
        <script src="http://code.jquery.com/jquery-1.8.2.min.js
        <script src="http://code.jquery.com/mobile/1.2.0/jquery
</head>
<body>

<div data-role="page">

        <div data-role="header">
                <h1>My Title</h1>
        </div><!-- /header -->

        <div data-role="content">
                <p>Hello world</p>
        </div><!-- /content -->

</div><!-- /page -->

</body>
</html>
```

Figura 2.1: Estructura básica de una aplicación con *JQuery Mobile*

En esta estructura se observa la declaración de un documento HTML5 básico, compuesto por un "*header*" y un "*body*". En el *header* se encuentra el elemento *meta-viewport*, el mismo define las dimensiones a utilizar para mostrar la aplicación en el dispositivo. En

este caso se configura la página para que se muestre con un ancho de 100 % el ancho de la pantalla del dispositivo. Luego se realizan llamadas a los archivos de *JQuery Mobile*, los mismos constan de un archivo Javascript y una hoja de estilos.

En el *body* se encuentra un elemento HTML clasificado con la etiqueta *data-role="page"*, el mismo será la página a mostrar en el dispositivo y por medio de estos elementos *JQuery Mobile* navega de una página a otra.

Elementos de interfaz gráfica con JQuery Mobile

JQuery Mobile provee una serie de elementos de interfaz *touch* compuesto con navegación basada en AJAX para soportar la animación en las transiciones de las páginas. A continuación se describen algunos de los elementos de interfaz de *JQuery Mobile*:

- **Listado de objetos:** Se trata de un elemento que representa un listado de objetos del sistema, por ejemplo un listado de inmuebles. El cuál se puede tocar fácilmente en la pantalla para realizar alguna acción con el mismo. El mismo es similar a un listado de botones. En la siguiente figura se puede observar el elemento gráfico para el listado de objetos:

Figura 2.2: Elemento gráfico de *JQuery Mobile* (listado de objetos)

- **Acordeones:** Los acordeones son estructuras gráficas que permiten agrupar información según se desee. Este elemento tiene la posibilidad de mostrar y ocultar la información que contiene para permitir el ahorro de espacio vertical en la pantalla del dispositivo. En la siguiente figura se observa un acordeón básico:

Figura 2.3: Elemento gráfico de *JQuery Mobile* (acordeones)

- **Botones:** *JQuery Mobile* provee una serie de botones con múltiples opciones de configuración, el objetivo de este elemento es representar fácilmente que al tocar el botón se realiza una acción, por ejemplo cambiar a otra página o iniciar sesión en el sistema. A continuación se presenta la imagen de algunos de los estilos de botones presentes en *JQuery Mobile*:

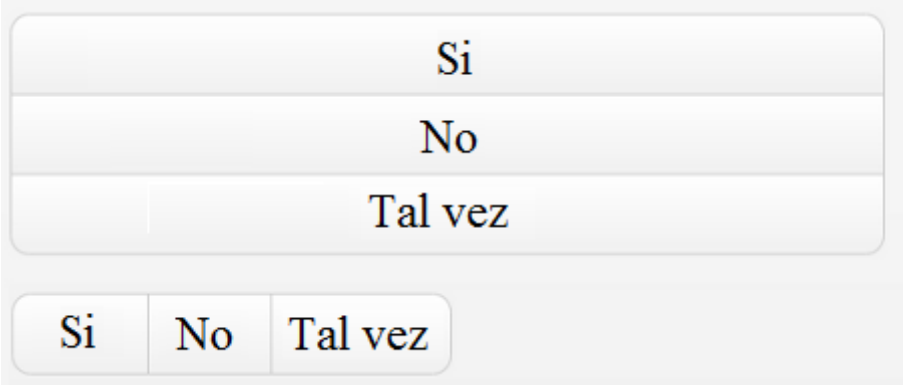


Figura 2.4: Elemento gráfico de *JQuery Mobile* (botones)

- **Objetos de formulario:** *JQuery Mobile* ha diseñado una serie de elementos de formularios con el objetivo de facilitar el ingreso de información en los formularios utilizando dispositivos móviles, buscando aprovechar las bondades de la manipulación con "*touch*". A continuación se muestra la imagen de algunos de los elementos de formulario en *JQuery Mobile*:

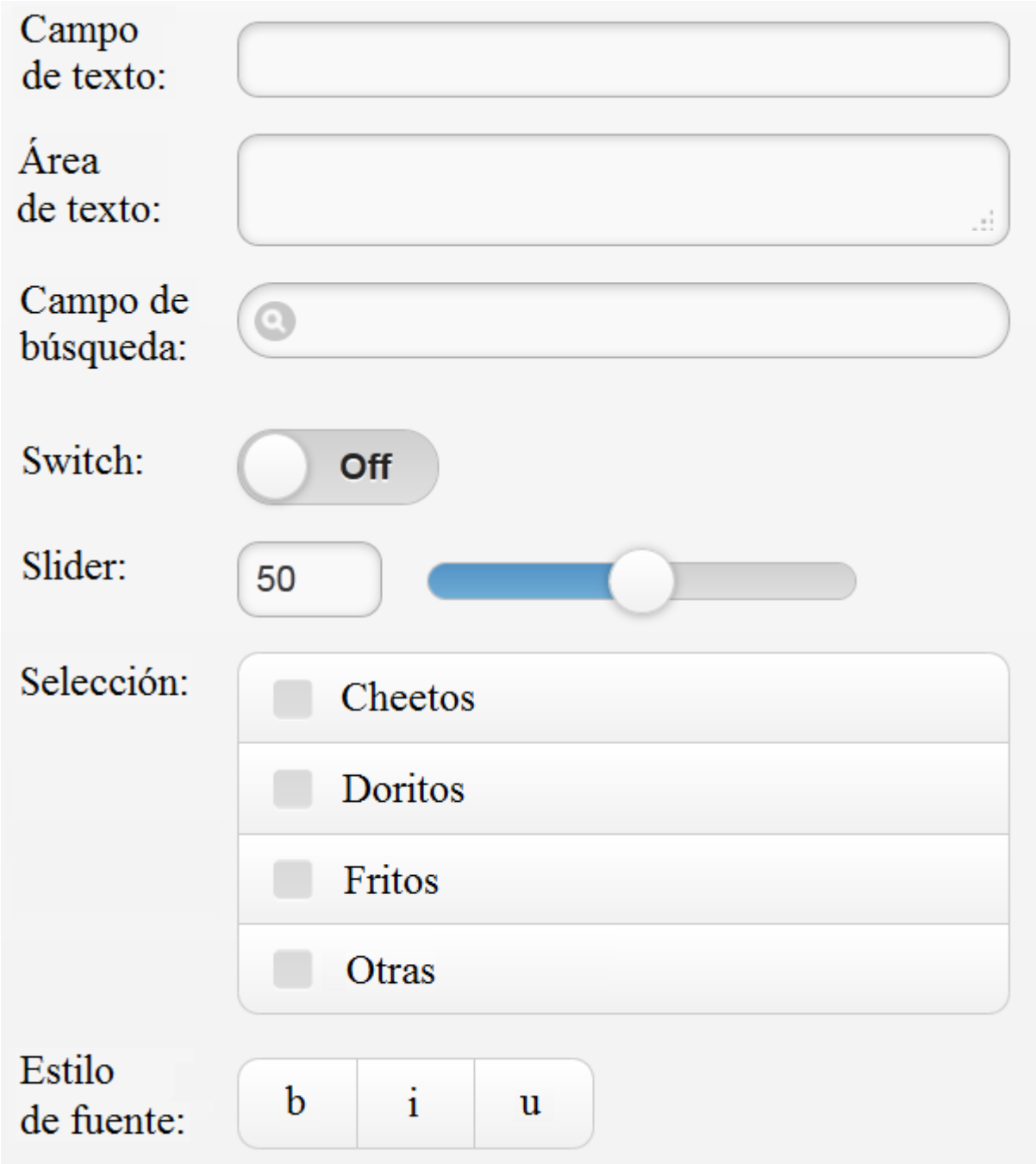


Figura 2.5: Elemento gráfico de *JQuery Mobile* (objetos de formulario)

Ejemplos de Aplicaciones desarrolladas con JQuery Mobile

JQuery Mobile ha sido utilizado por varias empresas para desarrollar su aplicación web móvil, entre las mismas se encuentran:

- La Universidad de Stanford: Aplicación web móvil destinada a proveer información a sus usuarios acerca de las últimas noticias y eventos de la universidad. Se puede acceder a la misma a través de la URL: **http://m.stanford.edu/**

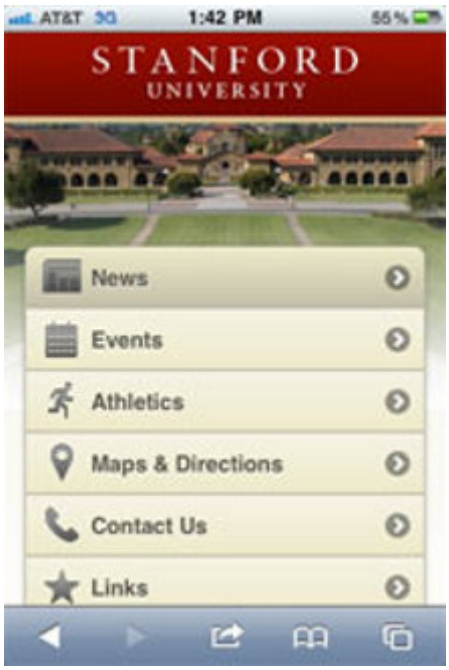


Figura 2.6: Aplicación de ejemplo: Universidad de Stanford

- Academia Khan: Academia sin fines de lucro cuyo objetivo es proveer educación de calidad a cualquier persona en cualquier sitio. Su aplicación web móvil permite a sus usuarios acceder a vídeos, lecciones e información de la academia Khan. Se puede ingresar a la misma a través de la URL: **http://khanapp.com/**

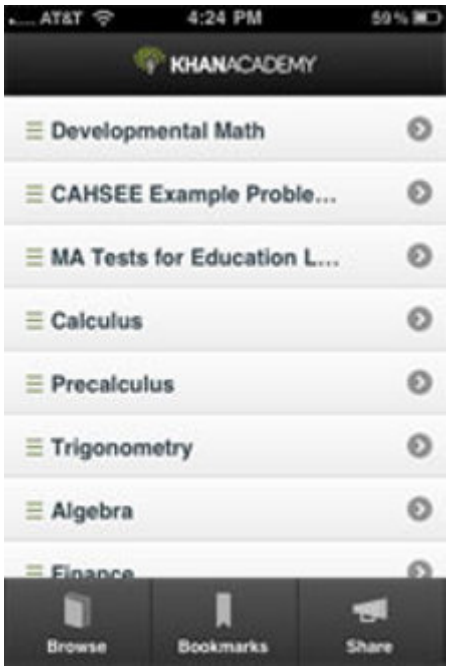


Figura 2.7: Aplicación de ejemplo: Academia Khan

- American Century: Empresa que ayuda a sus clientes a alcanzar metas financieras ofreciendo distintas opciones de inversión. Se puede acceder a la misma a través de la URL: **https://www.americancentury.com/mobile/**

Figura 2.8: Aplicación de ejemplo:American Century

2.2.2. Tecnologías del lado del servidor

Las tecnologías del lado del servidor se refieren a las herramientas de software que utiliza el servidor de aplicaciones para poder procesar las peticiones HTTP de los clientes y dar respuesta a los mismos. A continuación se describen las tecnologías del lado del servidor:

- Apache

El término **servidor web**, se puede referir tanto al hardware (la computadora que trabaja como servidor), como al software (la aplicación del servidor). La función principal de un servidor web es entregar las páginas web solicitadas por los clientes (navegadores web), utilizando el protocolo HTTP. Esto quiere decir que entrega documentos HTML y cualquier contenido adicional como: imágenes, hojas de estilos y *scripts*.

El proyecto **Apache** [The Apache Software Foundation, 2012], es un desarrollo colaborativo de software enfocado en implementar el código fuente de un servidor HTTP (servidor web), que sea robusto, seguro, eficiente, comercial, útil, extensible y gratuito.

El proyecto pertenece a Apache Software Foundation y es manejado por un grupo de voluntarios alrededor del mundo, utilizando Internet y la web para comunicarse, planear y desarrollar el servidor y su documentación. Cientos de usuarios contribuyen con ideas, código y documentación del proyecto. Apache ha sido el servidor web más popular en Internet desde abril del 1996.

- PHP

Es un lenguaje de propósito general [The PHP Group, 2013, The PHP Group, 2013], ampliamente utilizado que está especialmente adecuado para el desarrollo web y puede ser embebido en código HTML.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
  <head>
    <title>Example</title>
  </head>
  <body>
    <?php  echo "Esto es un script!";  ?>
  </body>
</html>
```

Figura 2.9: Ejemplo: Código PHP embebido en HTML

Las páginas PHP contienen HTML con código embebido el cuál cumple alguna tarea. El código PHP está encerrado en instrucciones especiales de procesamiento *<?php* y *?>* que permiten entrar y salir del modo en PHP.

Lo que distingue PHP de otros lenguajes (por ejemplo Javascript del lado del cliente), es que el código PHP es ejecutado en el lado del servidor, generando HTML el cuál es enviado al cliente. El cliente recibe el resultado de ejecutar los *scripts* PHP, pero no tendrá manera de saber cuál código fue ejecutado.

- Symfony

Symfony [SensioLabs, 2012], es un *framework* de PHP para desarrollo de aplicaciones web. El mismo está basado en el modelo MVC.

Su filosofía: "No tener que reinventar la rueda". Esto se refiere a la posibilidad de reutilizar código que haya sido probado con la intención de aumentar la velocidad de desarrollo, permitiendo a los desarrolladores concentrar su tiempo en las reglas de negocio.

Symfony está compuesto básicamente por:

- Una serie de herramientas prefabricadas para ser integradas al proyecto rápidamente como componentes de software. Esto permite al desarrollador tener que escribir menos cantidad de código y reducir el riesgo de errores.
- Una metodología para aplicaciones. La estructura de Symfony permite a los desarrolladores trabajar eficientemente y efectivamente en los aspectos más complejos de una tarea y el uso de las mejores prácticas de desarrollo de software garantiza la estabilidad, mantenibilidad e incrementabilidad de las aplicaciones que se desarrollen.

- MySQL

Un **Sistema Manejador de Bases de Datos (SMBD)** [Silva, 2006], es un conjunto de programas, procedimientos, lenguajes, etc. que suministra, tanto a los usuarios no informáticos como a los analistas, programadores o al administrador, los medios necesarios para describir, recuperar y manipular los datos almacenados en la base de datos, manteniendo su integridad, confidencialidad y seguridad

MySQL [Oracle Corporation, 2013], es un **Sistema Manejador de Bases de Datos** relacional, multihilo y multiusuario.

Actualmente MySQL es el sistema de gestión de bases de datos más popular del mundo con más de 100 millones de instalaciones alrededor del mundo, con aproximadamente 40 mil descargas del software. MySQL es ampliamente utilizado en aplicaciones web, en plataformas (Linux/Windows-Apache-MySQLPHP/ Perl/Python), y por herramientas de seguimiento de errores. Su popularidad en las aplicaciones web está muy ligada a PHP, que a menudo aparece en combinación con MySQL, sin embargo existen varias APIs que permiten, a aplicaciones escritas en diversos lenguajes de programación, acceder a las bases de datos MySQL, como son C, C++, C#, Pascal, Perl, Python, Ruby, Tcl, entre otras.

MySQL es muy rápida en la lectura cuando utiliza el motor no transaccional MyISAM, pero puede provocar problemas de integridad en entornos de alta concu-rrencia en la modificación. En aplicaciones web hay baja concurrencia en la modificación de datos y en cambio el entorno es intensivo en lectura de datos, lo que hace a MySQL ideal para este tipo de aplicaciones.

2.3. Mejores prácticas para el desarrollo de aplicaciones web móviles

Las tecnologías web se han vuelto lo suficientemente poderosas como para ser utilizadas para desarrollar aplicaciones complejas; esto ha sucedido por muchos años en el desarrollo de aplicaciones para *desktop* y *laptops*, pero se ha ido incrementando para dispositivos móviles también.

Las mejores prácticas tienen como objetivo permitir desarrollar aplicaciones web móviles ricas y dinámicas. La W3C reúne las prácticas más relevantes de la ingeniería del software, promoviendo aquellas prácticas que permiten al usuario una mejor experiencia y alertando acerca de las que no son apropiadas.

A continuación se mencionan las mejores prácticas para el desarrollo de aplicaciones web móviles estipuladas por la W3C [W3C, 2013]:

- Datos de la aplicación
    - Utilizar *cookies* escasamente: Utilizar *cookies* es bueno para almacenar pequeñas cantidades de información (por ejemplo: datos de personalización de la

aplicación para el usuario), pero para cantidades grandes de información puede afectar el desempeño de la aplicación. También puede que las *cookies* estén desactivadas en la configuración del usuario, por lo que la aplicación no podría basar su funcionamiento en la información guardada en las mismas.

- Utilizar las tecnologías apropiadas para el almacenamiento de datos locales en el lado del cliente: Algunos dispositivos proveen mecanismos para almacenamiento de información, el cuál puede almacenar más datos de los que se deberían en una *cookie*, mejorando el desempeño de la aplicación.
- Replicar los datos locales en el servidor si es necesario: En caso de que se utilicen tecnologías de almacenamiento del lado del cliente, los datos que necesiten ser compartidos entre dispositivos (por ejemplo: preferencias y estado de la aplicación configurado en un dispositivo en particular), deben ser replicados en el servidor.

- Seguridad y privacidad

  - No ejecutar datos de un JSON desconocido: Utilizar JSON para transferir datos al cliente es una técnica poderosa ya que se puede ejecutar más rápido que transferir un archivo grande de información. Sin embargo, la ejecución de un JSON no confiable podría tener un gran riesgo de seguridad ya que en los dispositivos móviles la información del usuario podría estar expuesta (ejemplo: ubicación del usuario, datos de contacto, entre otras).

- Conciencia y control por parte del usuario

  - Informar al usuario antes de acceder a la información personal o la del dispositivo: Se debe informar al usuario cuando la aplicación necesite acceder a la información personal y la del dispositivo. Las notificaciones deben proveer suficiente información para que el usuario juzgue según su criterio si desea permitir el acceso a los datos.
  - Premitir autenticación automática: Si la aplicación requiere autenticar al usuario (utilizando nombre de usuario y contraseña), se debe proveer una opción para autenticar al mismo de manera automática la próxima vez que necesite utilizar la aplicación, esto se debe a que un campo de datos en un dispositivo móvil es más complicado de llenar que uno en una computadora.

- Uso moderado de los recursos del dispositivo

  - Utilizar compresión en las transferencias: Comprimir el contenido para enviarlo eficientemente.
  - Minimizar el tamaño de la aplicación y sus datos: Aplicaciones pequeñas se descargan y ejecutan más rápido que las aplicaciones grandes.
  - Evitar redirecciones: El tiempo que se demora una redirección HTTP es mucho mayor en redes móviles por lo que se deben mantener las redirecciones al mínimo.

- Optimizar las peticiones de red: Establecer las conexiones de red necesarias para completar una petición HTTP puede tomar significativamente más tiempo en una red móvil que en una red cableada. Aunque normalmente el ancho de banda de una red móvil es más restringido es preferible realizar menos peticiones y más grandes.
- Minimizar el uso de recursos externos: Minimizar el uso de recursos como: hojas de estilos, *scripts*, imágenes, otras ya que estas generan peticiones HTTP que son particularmente costosas en una red móvil.
- Agregar imágenes fijas en un único recurso compuesto (Sprites): Las aplicaciones web usualmente utiliza un número de imágenes fijas, íconos, botones y otras. Si se combinan estas imágenes en una sola se reduce la cantidad de peticiones HTTP.
- Guardar recursos en caché: Recursos que cambian ocasionalmente (por ejemplo: el *avatar* del usuario), puede ser guardado en caché identificándolo con una URI que incluye un *Hash* del contenido del recurso. Ejemplo: <img src=" http://www.example.com/ userimages/ joeblogs-67f90da089da "> De esta manera el navegador no tendrá que validar su caché sino que el contenido se actualizará naturalmente.
- Guardar datos de las peticiones AJAX en caché: Si es posible, los datos que sean accedidos utilizando peticiones AJAX podrían guardarse en caché.
- No enviar información de las *cookies* si no es necesaria: Algunos recursos no necesitan la información de las *cookies*, entonces se puede mejorar el desempeño de la aplicación al no enviarlas.
- Mantener un tamaño del DOM razonable: El tamaño del Modelo de Documento de Objetos (DOM), debe se limitado en los dispositivos móviles. Las páginas largas y complejas pueden exceder este límite y causar errores inesperados.

▪ La experiencia del usuario

- Optimizar el tiempo de inicio de la aplicación: La experiencia del usuario es altamente influenciada por el tiempo de inicio de la aplicación. Las tecnologías "*offline*" de las aplicaciones web (ejemplo HTML5), colocan las aplicaciones web a la par con las aplicaciones nativas en cuanto a velocidad de inicio se refiere y a su habilidad de ser utilizada incluso cuando la cobertura de la red es intermitente.
- Minimizar la latencia percibida: Reducir la latencia percibida es un factor importante al proveer usabilidad general de la aplicación web.
- Diseñar múltiples métodos de interacción: Los métodos de interacción varían según cada dispositivo. Hay tres métodos principales de interacción que deben ser considerados al diseñar las interfaces gráficas, estos son, basados en "focus", basados en el puntero y los basados en "*touch*".
- Mantener el "focus" al actualizar páginas dinámicas: Utilizar la funcionalidad "focus" de Javascript sólo cuando sea necesario para el uso de la aplicación, esto para no confundir al usuario y permitirle tener control sobre las interacciones.

- Utilizar ID´s fragmentados para manejar las vistas de la aplicación: Las aplicaciones web pueden cambiar una vista sin necesidad de recargar completamente la página, ocultando y mostrando secciones de contenido. Cada una de estas secciones debe tener un identificador único para permitir este comportamiento.
- Hacer que los números telefónicos sean "Click-to-Call": Los esquemas estandarizados de URI han definido algunas funcionalidades comunes de los dispositivos, por ejemplo: realizar llamadas telefónicas, enviar mensajes de texto y manejar la agenda telefónica. Estos esquemas pueden permitir utilizar fácilmente estas funciones desde una aplicación web.
- Asegurar la fluidez de los textos: En las pantallas pequeñas es importante que los párrafos de texto se distribuyan vertical y horizontalmente para que no requieran desplazamiento horizontal para poder leerlo.
- Utilizar el elemento "Meta Viewport" para identificar el tamaño de la pantalla deseado: Este elemento le dice al dispositivo la escala en la que debe desplegar la página. Un ejemplo típico del elemento *Viewport* sería: <meta name="viewport" content="width=device-width, initial-scale=1.0"/>. Esta propiedad le informa al navegador que debe desplegar la página a 100 % de ancho de la pantalla. Es apropiado en páginas específicamente diseñadas para un tamaño de pantalla.

- Manejo de variaciones en la entrega de contenidos

  - Preferiblemente utilizar detección del lado del servidor cuando sea posible: Utilizar la evidencia disponible en el servidor para determinar las propiedades para la entrega de contenido, y así adaptar las respuestas al cliente antes de la transferencia, mejorando la experiencia del usuario y evitando la transferencia de datos innecesarios e incompatibles.
  - Utilizar detección del lado del cliente cuando sea necesario: Cuando no es posible determinar ciertas propiedades de la entrega de contenido desde el servidor, esta información puede estar disponible en el cliente. Una vez obtenida en el cliente, esta información puede utilizarse directamente para adaptar la presentación, o puede ser usada para solicitar al servidor contenido adaptado.
  - Utilizar clasificación de dispositivos para simplificar la adaptación del contenido: Ya que existe una gran variedad de dispositivos, se deben clasificar los dispositivos e implementar una variante de la aplicación para cada uno. Esto permite explotar las capacidades del dispositivo.
  - Ofrecer al usuario la opción de para elegir la interfaz: Cuando existan múltiples versiones de una aplicación (a soportar varias clasificaciones de dispositivos), siempre se debe ofrecer al usuario la oportunidad de cambiar la versión de la aplicación.

En este capítulo se han descrito detalladamente las herramientas tecnológicas a utilizar como base conceptual para guiar el desarrollo de la aplicación web móvil.

Capítulo 3

Programación Extrema (XP)

Al momento de realizar un sistema o software se tiende a investigar diferentes métodos, técnicas y herramientas para entornos de desarrollo, sin embargo existe un gran número de autores, teorías, propuestas y demás, a tomar en consideración.

Para un equipo de desarrollo de software es importante la velocidad de avance en el desarrollo, pero también es importante ser ágil en la respuesta a cambios imprevistos, como por ejemplo: cambios en los requerimientos, falta de presupuesto o recursos, entre otras. Por esta necesidad surgen los denominados métodos ágiles.

3.1. Métodos ágiles

Los métodos ágiles ágiles [Beck, 2011], (como por ejemplo XP, SCRUM, DSDM, Crystal, etc...), se basan en la adaptabilidad de cualquier cambio como medio para aumentar las posibilidades de éxito de un proyecto.

De forma que un método ágil es aquel que tiene como principios:

- Los individuos y sus interacciones son más importantes que los procesos y las herramientas.
- El software que funciona es más importante que la documentación exhaustiva.
- La colaboración con el cliente en lugar de la negociación de contratos.
- La respuesta delante del cambio en lugar de seguir un plan cerrado.

3.2. Definición de Programación Extrema (XP)

La Programación Extrema [Beck, 1999], es un método de desarrollo ligero (o ágil) basado en una serie de valores y de prácticas de buenas maneras que persigue el objetivo de aumentar la productividad a la hora de desarrollar software.

Este modelo de programación se basa en una serie de métodos de desarrollo de software en la que se da prioridad a los trabajos que dan un resultado directo y que reducen la burocracia que hay alrededor de la programación.

3.3. Objetivos

Los objetivos de XP son los siguientes:

- La satisfacción del cliente: Este método trata de dar al cliente el software que él necesita y cuando lo necesita. Por tanto, se debe responder muy rápido a las necesidades del mismo, incluso cuando los cambios sean al final de ciclo de la programación.
- Potenciar al máximo el trabajo en grupo. Tanto los jefes de proyecto, los clientes y desarrolladores, son parte del equipo y están involucrados en el desarrollo del software.

3.4. Problemas de desarrollo de software

Los problemas más comunes en el desarrollo de software son:

- Retrasos en la planificación: Llegada la fecha de entregar el software éste no está disponible.
- El proyecto se cancela: Después de cierto tiempo desarrollando el proyecto, se decide cancelar el mismo.
- Sistemas deteriorados: El software se ha creado pero después de un par de años el costo de su mantenimiento es tan complicado que definitivamente se abandona su producción.
- Tasa de defectos: El software se pone en producción pero los defectos son tantos que nadie lo usa.
- Requisitos mal comprendidos: El software no resuelve los requisitos planificados inicialmente.
- Cambios de negocio: El problema que resolvía el software ha cambiado y el software no se ha adaptado.
- Falsa riqueza: El software hace muchas cosas técnicamente muy interesantes, pero no resuelven el problema del cliente.

- Cambios de personal: Después de unos años de trabajo los programadores abandonan el proyecto.

XP plantea estrategias para manejar los problemas antes mencionados, logrando así ser un método muy productivo y además, producir software de calidad. A continuación se describe como XP maneja los problemas del proceso de desarrollo:

- Retrasos en la planificación: XP propone realizar ciclos de entrega cortos, un mes cuando mucho, de esta manera el impacto de los retrasos será limitado. Entre cada ciclo de entrega, XP utiliza de una a dos semanas para recibir un observaciones detalladas del progreso de los requerimientos solicitados por el cliente. Entre una iteración, XP planifica tareas de uno a tres días de duración, para que el equipo pueda resolver problemas incluso durante una iteración. Finalmente, XP propone que se implementen primero las funcionalidades con más prioridad, para que cualquier funcionalidad que no sea completada para la fecha de entrega sea de bajo valor.

- El proyecto se cancela: XP solicita al cliente que defina cuál sería la mínima entrega que satisfaga el objetivo de su negocio, así menos cosas podrán salir mal antes de subir el proyecto al ambiente de producción.

- Sistemas deteriorados: XP crea y mantiene un grupo de pruebas, las cuáles son ejecutadas luego de cada cambio (varias veces al día), para asegurar la calidad del producto.

- Tasa de defectos: XP prueba el producto desde dos perspectivas, como programador diseñando pruebas para cada función y como cliente diseñando pruebas por cada funcionalidad.

- Requisitos mal comprendidos: XP propone que el cliente sea parte integral del equipo de desarrollo. De esta manera la especificación del proyecto es refinada continuamente durante el desarrollo.

- Cambios de negocio: XP disminuye los ciclos de entrega, de esta manera hay menos cambios durante el desarrollo de cada una.

- Falsa riqueza: XP insiste en que sólo las tareas con mayor prioridad son las que se deben realizar.

- Cambios de personal: XP solicita a los programadores el compromiso de aceptar la responsabilidad de estimar y completar su propio trabajo y obtener información acerca de los tiempos invertidos en el desarrollo para que las estimaciones sean más precisas, de esta manera hay menos chance de que un programador se frustre por solicitarle hacer algo imposible. Finalmente, XP incorpora un modelo explícito para el cambio de personal. Los nuevos miembros del equipo son motivados a aceptar cada vez más responsabilidades y son asistidos a lo largo del camino por los demás integrantes del equipo.

3.5. Las cuatro variables

XP define cuatro variables para proyectos de software: costo, tiempo, calidad y ámbito.

Beck propone que sólo tres puedan ser establecidas por las fuerzas externas (jefes de proyecto y clientes), mientras que el ámbito debe ser establecido por los programadores en función de las otras tres.

Poniendo un episodio diario de desarrollo:

El jefe de proyecto: "Se necesitan estos requisitos realizados para el día 1 de mes próximo, con lo que contamos con el equipo actual. La calidad es lo primero".

Todos saben qué es lo primero que se pierde en estos casos: "la calidad", ¿Por qué? Porque nadie es capaz de trabajar bien cuando se le somete a mucha presión.

XP propone que se equilibren todas las partes implicadas en el proyecto hasta que el valor que alcancen las cuatro variables sea el correcto para todas las partes: "Si quieres más calidad en menos tiempo tendrás que aumentar el equipo e incrementar el costo".

Además con el agravante de que estas cuatro variables no guardan una relación tan directa como en principio pueda parecer. El incremento del número de programadores no repercutirá de manera lineal en el tiempo de desarrollo del proyecto, siendo de todos conocidos el dicho: "nueve mujeres no pueden tener un hijo en un mes".

Con la calidad suele suceder un fenómeno extraño: frecuentemente un proyecto en el que se trate de aumentar la calidad conduce a que el proyecto pueda realizarse en menos tiempo, siempre con unos márgenes obviamente. Es verdad que cuando un equipo de desarrollo se acostumbra a realizar pruebas intensivas, se siguen estándares de codificación, poco a poco se comenzará a andar más rápido y más seguro, por tanto más preparados para futuros cambios, sin estrés y así sucesivamente.

Frente a esto existe la tentación de entregar el trabajo más rápido, por tanto probar menos, codificar más rápido y peor, sin hacer planteamientos maduros, esto repercutirá en la confianza de los clientes, al entregarle trabajos con fallos. Esta es una apuesta a muy corto plazo y suele ser una invitación al desastre, conduce a la desmoralización del equipo y con ello a la larga a la ralentización del proyecto y la pérdida de tiempo que se habría conseguido en un principio.

La cuarta variable, el ámbito del proyecto, suele ser conveniente que sea establecida por el equipo de desarrollo. Es una variable muy importante que va a decir hasta donde va a llegar el software, que problemas va a resolver y cuáles se van a dejar para siguientes versiones.

3.6. Los cuatro valores

En el ciclo de vida del desarrollo de un proyecto de software los cambios van a aparecer, cambiarán los requisitos, las reglas de negocio, el personal, la tecnología, todo va a cambiar. Por tanto el problema no es el cambio en sí, ya que este va a suceder sino la incapacidad de enfrentar estos cambios.

Como en cualquier otra actividad humana se necesitan valores para desarrollar el trabajo y conseguir los planteamientos iniciales.

Estos cuatro valores son:

- Comunicación
- Sencillez
- Retroalimentación
- Valentía

Comunicación: Es común que alguna vez ocurra un problema en el equipo de desarrollo por falta de comunicación, por no comentar un cambio crítico en el diseño, por no preguntar lo que se piensa al cliente. La mala comunicación no surge por casualidad y hay circunstancias que conducen a la ruptura de la comunicación. XP ayuda mediante sus prácticas a fomentar la comunicación.

Sencillez: La sencillez no siempre es fácil. Usualmente los programadores agregan funcionalidades u otros detalles al sistema en desarrollo sin que estos se encuentren en los requerimiento iniciales, los agregan, por el pensamiento de que esta funcionalidad será utilizada a futuro. XP en sus valores define que es mejor hacer una cosa sencilla hoy, que hacer una cosa complicada y no utilizarla después.

Retroalimentación: "No me preguntes a mí, pregúntale al sistema", es la primera clave de la retroalimentación, por medio de pruebas funcionales el software mantendrá informado del grado de fiabilidad del sistema, esta información realmente no tiene precio. Los clientes y las personas que escriben pruebas tienen una retroalimentación real de su sistema.

Valentía: Asumir retos, ser valientes antes los problemas y afrontarlos. Desarrollar software se asemeja a escalar una montaña, cuando se hace una cima hay que volver a bajar para hacer otra cima y así constantemente, planteando hacer sistemas cada vez más sencillos y fiables.

3.7. Actividades básicas del desarrollo de software

A continuación se presentan las actividades básicas para desarrollar buen software propuestas por el método XP.

Codificación: Es el proceso en donde se implementan y se plasman las ideas a través del código. En una programación en XP el código expresa la interpretación del problema, así se puede utilizar el código para comunicar, para compartir las ideas y por tanto para aprender y mejorar.

Pruebas: Las pruebas dan la oportunidad de saber si lo que se implementó es lo que en realidad se pensaba que se había implementado. Las pruebas indican que el trabajo funciona, cuando no se pueda pensar en ninguna prueba que pudiese originar un fallo en el sistema entonces este está completo.

Al diseñar pruebas, se debe de pensar en todas las posibles pruebas para el código. Programar y probar es más rápido que sólo programar. Habrá menos errores, se tendrá que volver menos veces sobre el código, se perderá menos tiempo escuchado como los clientes dicen que no funciona. Se puede ganar media hora de productividad sin hacer pruebas, pero se perderá mucho tiempo en la depuración.

Las pruebas deben de ser sensatas y valientes. No se deben hacer pruebas sencillas que no testen a fondo el sistema, sino es más probable que aparezcan errores más adelante.

Escuchar: Los programadores no lo conocen todo y sobre todo muchas cosas que las personas de negocios piensan que son interesantes. Si ellos pudieran programarse su propio software ¿para que querrían un programador?.

Si van a hacer pruebas hay que preguntar al cliente si lo obtenido es lo deseado y hay que preguntar a quien necesita la información (usuario final). Hay que entender cuáles son los problemas del negocio del cliente, tener una escucha activa explicando lo que es fácil y difícil de obtener y la retroalimentación entre ambos ayudan a todos a entender los problemas.

Se debe buscar la manera de estructurar la comunicación de tal manera que las cosas importantes para el desarrollo sean comunicadas de manera efectiva, cuando se necesiten y con el nivel de detalle necesario. Igualmente se debe evitar la comunicación de cosas que no ayuden al desarrollo, eso se logra cuando lo importante para el desarrollo ha sido realmente entendido.

Diseño: El diseño crea una estructura que organiza la lógica del sistema, un buen diseño permite que el sistema crezca con cambios en un sólo lugar. Los diseños deben ser sencillos, si alguna parte del sistema es de desarrollo complejo, hay que dividirla en varias. Si hay fallos en el diseño o malos diseños, estos deben de ser corregidos cuanto antes.

3.8. Fundamentos del método XP

A continuación se describen las prácticas propuestas por XP, para el desarrollo de software:

3.8.1. Planificación

XP plantea la planificación como un permanente diálogo entre las partes, la empresarial (deseable) y la técnica (posible). Las personas del negocio necesitan determinar:

- Ámbito: ¿Qué es lo que el software debe resolver para que este genere valor?
- Prioridad: ¿Qué debe ser hecho en primer lugar?
- Composición de versiones: ¿Cuánto es necesario hacer para saber si el negocio va mejor con software que sin el?. En cuanto el software aporte algo al negocio se debe tener listas las primeras versiones.
- Fechas de versiones: Se debe definir cuáles son las fechas en la presencia del software o parte del mismo pudiese marcar la diferencia.
- Estimaciones: ¿Cuánto tiempo lleva implementar una característica?
- Consecuencias: Informar sobre las consecuencias de la toma de decisiones por parte del negocio. Por ejemplo el cambiar las bases de datos a Oracle. Procesos: ¿Cómo se organiza el trabajo y el equipo?
- Programación detallada: Dentro de una versión ¿qué problemas se resolverán primero?

3.8.2. Pequeñas versiones

Cada versión debe de ser tan pequeña como fuera posible, conteniendo los requisitos de negocios más importantes, las versiones tiene que tener sentido como un todo, no se puede implementar media característica y lanzar la versión.

Es mejor planificar para 1 mes o 2 que para seis meses y un año, las compañías que entregan software muy voluminoso no son capaces de hacerlo con mucha frecuencia.

3.9. Diseño

A continuación se describen algunos principios que hay que tener en cuenta a la hora de realizar el diseño del software.

3.9.1. Metáfora

Una metáfora es una historia acerca de cómo funciona el sistema. Por ejemplo, metáforas sencillas: “Programa de gestión de compras, ventas, con gestión de cartera y almacén”. Las metáforas ayudan a cualquier persona a entender el objetivo del programa.

3.9.2. Diseño sencillo

El diseño adecuado para el software es aquel que:

- Funciona con todas las pruebas.
- No tiene lógica duplicada.
- Manifiesta cada intención importante para los programadores.
- Tiene el menor número de clases y métodos.

Contrariamente a lo que se pensaba el "Implementa para hoy, diseña para mañana", no es del todo correcto si se piensa que el futuro es incierto.

3.10. Desarrollo

A continuación se describen los principios para la fase de desarrollo del software.

3.10.1. Recodificación

Cuando se implementan nuevas características en los programas, se plantea la manera de hacerlo lo más simple posible. Después de implementar esta característica, el programador se pregunta como hacer el programa más simple sin perder funcionalidad, este proceso se le denomina recodificar o refactorizar (refactoring). No se debe recodificar ante especulaciones sino cuándo el sistema lo requiera.

3.10.2. Programación por parejas

Todo el código de producción lo escriben dos personas frente a la computadora, con un sólo ratón y un sólo teclado. Cada miembro de la pareja juega su papel: uno codifica en la computadora y piensa la mejor manera de hacerlo, el otro piensa más estratégicamente, ¿va a funcionar? ¿puede haber pruebas donde no funcione? ¿hay forma de simplificar el sistema global para que el problema desaparezca?.

3.10.3. Propiedad colectiva

Cualquiera que crea que puede aportar valor al código en cualquier parte puede hacerlo, ningún miembro del equipo es propietario del código. Si se hace el código propietario y se necesita su autor para que lo cambie entonces se estará alejándo cada vez más de la comprensión del problema, si se necesita un cambio sobre una parte del código se hace y punto. XP propone una propiedad colectiva sobre el código nadie conoce cada parte igual de bien pero todos conocen algo sobre cada parte, esto preparará para la sustitución no traumática de cada miembro del equipo.

3.10.4. Integración continúa

El código se debe integrar como mínimo una vez al día y realizar las pruebas sobre la totalidad del sistema. Una pareja de programadores se encargará de integrar todo el código en una máquina y realizar todas las pruebas hasta que estas funcionen al 100 %.

3.10.5. 40 Horas semanales

Para estar fresco y motivado cada día. Esto requiere que se trabajen 40 horas a la semana, muchos programadores no pueden estar más de 35 horas concentrados a la semana, otros pueden llegar hasta 45 pero ninguno puede llegar a 60 horas durante varias semanas y aún seguir fresco, creativo y confiado. Las horas extras son síntoma de serios problemas en el proyecto.

3.10.6. Cliente In-situ

Un cliente real debe sentarse con el equipo de programadores, estar disponible para responder a sus preguntas, resolver discusiones y fijar las prioridades. Lo difícil es que el cliente ceda una persona que conozca el negocio para que se integre en el equipo normalmente estos elementos son muy valiosos, pero se le debe hacer ver que será mejor para su negocio tener un software pronto en funcionamiento.

3.10.7. Estándares de codificación

Si los programadores van a estar tocando partes distintas del sistema, intercambiando compañeros, haciendo *refactoring*, se debe establecer un estándar de codificación aceptado e implantado por todo el equipo.

3.11. Pruebas

Todas las características del programa deben probarse, los programadores escriben pruebas para verificar el correcto funcionamiento del programa, los clientes realizan pruebas funcionales. El resultado es un programa más seguro que conforme pasa el tiempo es capaz de aceptar nuevos cambios.

Entre las formas de pruebas más comunes se encuentran las puebas unitarias y de aceptacion.

Pruebas unitarias: Las pruebas unitarias [Sun, 2000], están dirigidas a probar las funcionalidades aisladamente. Por ejemplo una funcionalidad que envíe correos electrónicos debe ser capaz de probarse aisladamente chequeando que sea capaz de enviar correos y asegurándose de probar todas las posibilidades de fallo y éxito. En un modelo basado en pruebas, se deben definir casos de prueba para cada funcionalidad de forma aislada, una prueba no debe depender de otras funcionalidades.

Estas pruebas aseguran la calidad del código entregado. Es la mejor forma de detectar errores tempranamente en el desarrollo. No obstante, esto no asegura detectar todos los errores, por lo tanto las prueba de integración y aceptación siguen siendo necesarias.

Pruebas de aceptación (PA): Las pruebas de aceptación [Linkedin, 2011], tienen como propósito demostrar al cliente el cumplimiento de un requisito del software, realizando ejecuciones de las funcionalidades en distintos escenario y parametros. Las características de una PA son:

- Describe un escenario (secuencia de pasos) de ejecución o uso del sistema desde la perspectiva del cliente.
- Puede estar asociada a un requisito funcional o requisito no funcional.
- Un requisito tiene una o más PAs asociadas.
- Las PAs cubren desde escenarios típicos/frecuentes hasta los más excepcionales.
- Una PA puede tener infinitas instanciaciones (ejecuciones con valores concretos). El diseño de las instanciaciones y su aplicación es trabajo del evaluador.

En este capítulo se explicó detalladamente el método de desarrollo de software a utilizar en la realización de la aplicación web móvil. Para este caso se trata del método de desa-rrollo Programación Extrema, el mismo adaptará a las condiciones específicas de este desarrollo.

Parte III

Marco aplicativo

Capítulo 4

Adaptación del proceso de desarrollo XP

A continuación se detallan las prácticas y principios que se toman del proceso de desarrollo XP para la creación de la aplicación web móvil.

Se ha tomado este método de desarrollo debido a que se desea que el desarrollo sea flexible, el diseño simple y abierto al cambio, donde el objetivo principal sea la realización de la aplicación y no producir una extensa documentación que necesite muchos artefactos. XP especifica un conjunto de prácticas y actividades que se utilizan a lo largo del desarrollo de la aplicación.

4.1. Metáfora

Según los planteamientos de XP se define la siguiente metáfora para explicar el objetivo central de la aplicación: "Aplicación web móvil para la búsqueda y visualización de inmuebles de Inmobilia.com". En la siguiente figura se muestra gráficamente la metáfora mencionada.

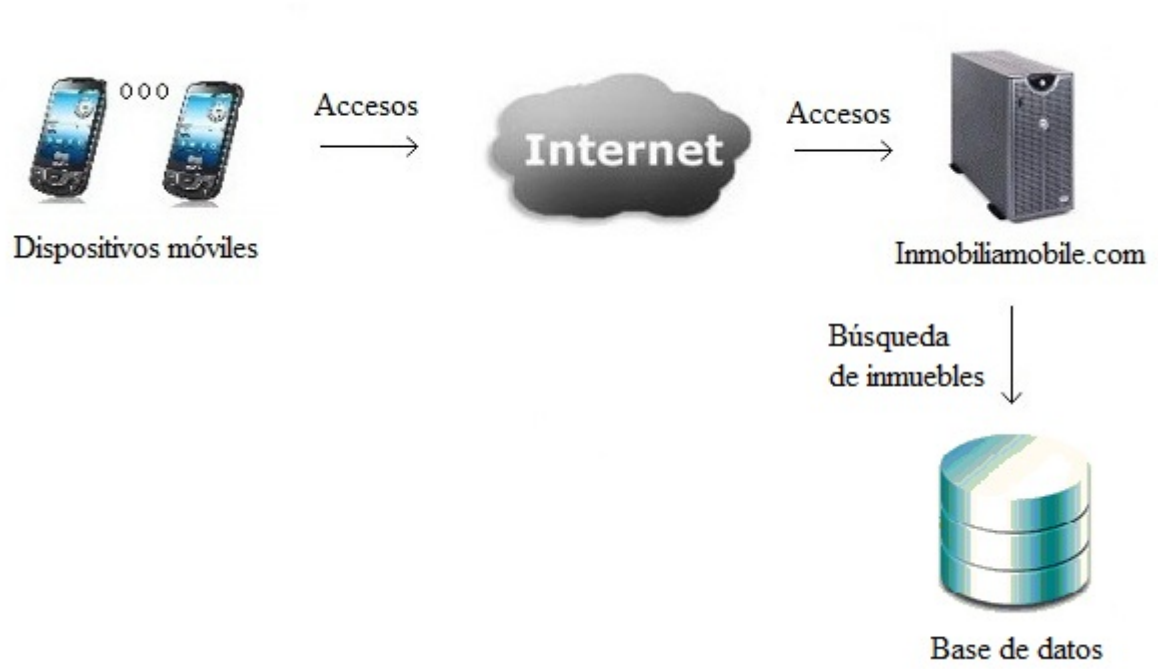


Figura 4.1: Representación gráfica de la metáfora

4.2. Iteraciones

Siguiendo el proceso de desarrollo de XP se utilizan iteraciones. En cada una de ellas se planifican un conjunto de actividades y tareas que deben desarrollarse en un período de tiempo determinado. Las iteraciones son relativamente cortas, es por esto que su duración se estima en una semana aproximadamente.

4.3. Actividades

En cada iteración se aplican las cuatro actividades que propone XP, estas no necesariamente se tienen que realizar en un orden específico y se puede intercambiar de una a otra cuando se considere necesario. Se trata de las actividades: diseño, planificación, codificación y pruebas.

- Planificación: En esta actividad se determina el alcance que se le da a cada iteración, definiendo qué objetivos y metas deben ser realizadas. Se utilizan tablas de planificación, donde se describen los requerimientos del sistema. Éstas sirven de base

para las pruebas funcionales y ayudan a proporcionar una estimación del tiempo necesario para el desarrollo.

- Diseño: Siguiendo el proceso de desarrollo de XP, el diseño debe ser simple y sencillo. Se define una metáfora del sistema que muestra una visión global de lo que se quiere desarrollar, la cuál es: "Aplicación web móvil para búsqueda de inmuebles". Adicionalmente, se pueden usar diagramas simples para entender cada funcionalidad a implementar. En esta actividad se utiliza la refactorización para cambiar el diseño o código del sistema por otro que se adapte mejor a la solución.

- Codificación: Para esta actividad se codifica en el lenguaje de programación PHP. Se mantiene una versión centralizada del código del proyecto, con la que se realiza frecuentemente la integración del código para poder realizar las pruebas con la última versión del mismo. Se establecen estándares de codificación, manteniendo la consistencia y legibilidad del código.

- Pruebas: Para cada una de las iteraciones se realizan códigos de pruebas donde se puede constatar si la aplicación realiza la actividad esperada o no. Durante una iteración cada requerimiento estipulado en la planificación es sometido a pruebas funcionales. Se especifican escenarios, verificando así cuando un requerimiento ha sido implementado de manera correcta. Los requerimientos no se consideran completados hasta que no hayan pasado satisfactoriamente las pruebas funcionales. Cabe destacar que a lo largo del presente trabajo, es posible que algunas iteraciones no implementen las cuatro actividades antes descritas, debido a que no son requeridas.

4.4. Descripción detallada de las iteraciones

El desarrollo de la aplicación web móvil para Inmobilia.com comprende varias iteraciones donde se avanza hasta lograr el producto final. Estas iteraciones se pueden agrupar de la siguiente manera:

- Iteración 0 y 1: Iteraciones iniciales para levantamiento de requerimientos e información acerca de las características del negocio a modelar. En este grupo se crea y configura el proyecto Symfony.

- Iteración 2 y 3: Se implementan las páginas principales de la aplicación (el *home* y el listado de inmueble). Las cuáles permiten el acceso a las funcionalidades principales de la aplicación.

- Iteración 4: Se crea la sección para mostrar las características del inmueble.

- Iteración 5, 6 y 7: Desarrollo del buscador de inmuebles.

- Iteración 8: Recodificación para la sección de listado de inmuebles para soportar páginación.

- Iteración 9 y 10: Recodificación del detalle del inmueble para agregar funcionalidades planificadas (redes sociales, *slider* de imágenes y mapas).

- Iteración 11: Implementación de funcionalidades necesarias para manejar la sesión del usuario.
- Iteración 12 y 13: Recodificación en el detalle del inmueble para agregar funcionalidades (marcar como favorito y comentar un inmueble).
- Iteración 14: Se realizan las pruebas de aceptación por parte de los usuarios. Se realizan correcciones en la aplicación según las observaciones recibidas de los usuarios.

En esta sección del documento se describe detalladamente cada iteración realizada durante el desarrollo del software.

4.4.1. Iteración 0: Identificación del modelo

En la iteración 0 se define e identifica el modelo que compone el repositorio digital de inmuebles. Se trata de una base de datos en MySQL la cuál maneja más de 60 tablas, donde cada una es utilizada por varias funcionalidades. El objetivo de esta iteración es identificar un subconjunto del modelo con sólo aquellas tablas que sean necesarias para soportar las funcionalidades de la aplicación web móvil a desarrollar.

- Planificación

Nro.	Fecha Inicio	Descripción
1.0	13/08/12	Estudiar las tablas del sistema.
2.0	17/08/12	Identificar las tablas a usar de Inmobilia.

- Diseño

A continuación se explica el modelo de datos que conforma la parte de persistencia del sistema. Inicialmente se mostrará con una idea general de las tablas asociadas para luego explicar con más detalle cuáles son las tablas que participan en el sistema.

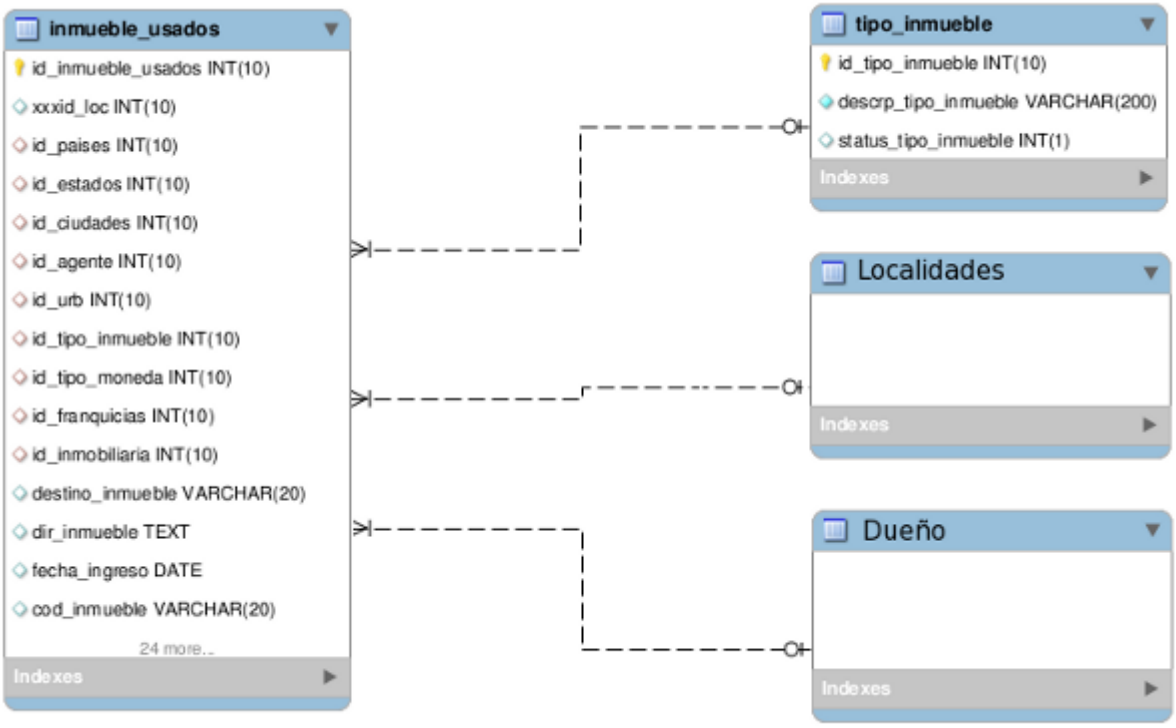


Figura 4.2: Diagrama general de la estructura del modelo

En la imagen se puede apreciar de manera general y simplificada el modelo de datos que maneja la aplicación web móvil de Inmobilia. Los "inmuebles" tienen asociado valores de localidad (claves foráneas de localidades), un tipo de inmueble (claves foráneas del tipo de inmueble) y tiene asociado un dueño. Cabe destacar que la tabla localidades y la tabla dueño es una agrupación de tablas que se explican a continuación.

La tabla principal del sistema es "inmueble_usados", en la misma se encuentra almacenada toda la información de los inmuebles que se buscará y consultará en la aplicación web móvil de Inmobilia. Cada inmueble tiene asociada información de otras tablas a través de relaciones de clave foránea como el tipo de inmueble (almacenado en la tabla "tipo_inmueble"), que especifica si el inmueble es un apartamento, casa, oficina u otras.

Los inmuebles también tienen asociada la información de la localidad en la que se encuentran, que se refiere al estado, ciudad y urbanización donde se encuentra físicamente el inmueble, se asume que el único país donde se realiza la búsqueda será Venezuela. También se tiene la información del dueño del inmueble que es la persona que carga el inmueble al sistema, puede ser del tipo inmobiliaria, agente y particular, cada uno de estos tipos de dueños tiene asociada una tabla con su mismo nombre.

En la siguiente figura se muestra el diagrama completo del modelo de datos a utilizar en la aplicación.

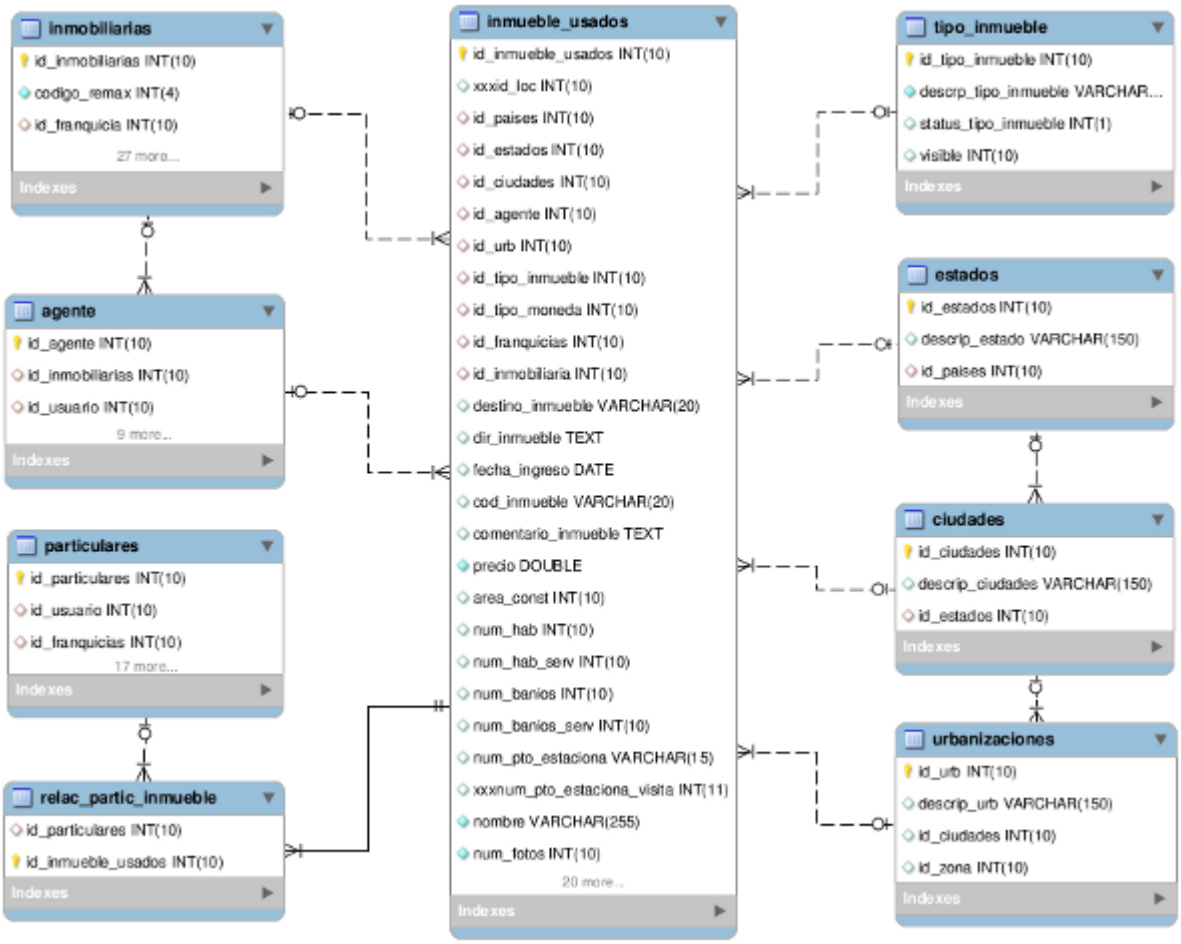


Figura 4.3: Diagrama completo de la estructura del modelo de datos

4.4.2. Iteración 1: Estructura base del proyecto

El objetivo de esta iteración es implementar las porciones de código necesarias para generar una estructura sobre la que funcionara la aplicación completa. Esta estructura inicial comprende las llamadas a los archivos CSS y JS necesarios así como la estructura que utilizará cada sección de la aplicación móvil. La estructura base del proyecto se conoce como el *layout* (plantilla), de la aplicación y sobre esta se carga el contenido de cada sección.

- Planificación

Nro.	Fecha Inicio	Descripción
3.0	20/08/12	Definir la estructura del *layout* HTML.
4.0	22/08/12	Implementar el código necesario para soportar la estructura definida (incluye: Código PHP, CSS, JS y HTML).

- Diseño

A continuación se muestra la imagen de lo que se desea implementar como *layout* para esta iteración, el diseño de la interfaz gráfica para este caso debe ser simple por lo que se utiliza sólo estilos de CSS y una única imagen de fondo para generarlo.

Figura 4.4: Diseño del *layout* de la aplicación

En la imagen se observa la estructura principal de la aplicación web móvil, primero se coloca un encabezado (*header*), el cuál identifica e informa al usuario sobre la aplicación en la que se encuentra.

En segundo lugar se tiene un área de contenido, en esta área se cargará cualquier contenido que pertenezca a las secciones de la página.

Por último se encuentra un pie de página (*footer*), se trata de un menú de navegación con las opciones más relevantes de la aplicación. Este menú se mantiene fijo en la parte inferior de la pantalla sin importar si el usuario se desplaza hacia arriba o hacia abajo en la sección que esté visualizando.

- Codificación

La implementación del *layout* se puede ver en el siguiente código:

layout.php - Editor

layout.php

```
html  body  div
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <?php include_http_metas() ?>
5      <?php include_metas() ?>
6      <?php include_title() ?>
7      <link rel="shortcut icon" href="/favicon.ico" />
8      <?php include_stylesheets() ?>
9      <?php include_javascripts() ?>
10   </head>
11   <body style="background-image: url('/images/back_<?php echo '1'; ?>.png'); background-re
12     <div data-role="page" id="foo" style="background: none;">
13       <div id="header" data-role="header">
14         <h1>Inmobilia.com<br/>
15           <span style="font-size: 10px; text-align: center;">Su portal inmobiliario</span>
16         </h1>
17       </div>
18       <hr style="height: 5px; margin: 0; background-color: lightgray;" />
19       <!-- /header -->
20
21       <div data-role="content">
22         <div class="main-container">
23           <?php echo $sf_content ?>
24         </div>
25       </div><!-- /content -->
26
27       <div data-id="myfooter" data-role="footer" data-position="fixed">
28         <div data-role="navbar">
29           <ul>
30             <li><?php echo link_to('Volver', '@homepage', array('data-rel' => 'back', 'dat
31             <li><?php echo link_to('Inicio', '@homepage', array('data-role' => 'button', '
32             <li><?php echo link_to('Buscar', '@main?action=buscar', array('data-role' => '
33           </ul>
34         </div><!-- /navbar -->
35       </div><!-- /footer -->
36     </div><!-- /page -->
37   </body>
38 </html>
```

36 | 1 INS

Figura 4.5: Código del *layout* de la aplicación

Este código genera la estructura principal de la aplicación explicada anteriormente en el diseño.

En la línea 13 hasta la 19 se encuentra el código necesario para el encabezado. En la línea 21 hasta 25 el código para el contenido de la sección, el cuál es cargado en esta área

utilizando la variable de symfony "*$Sf_ content*". Finalmente en las líneas 27 hasta la 35 se encuentra el código necesario para cargar el pie de página de la aplicación con el código necesario para que los botones que allí se encuentran funcionen correctamente.

- Pruebas

Lo esperado para esta iteración es poder visualizar la plantilla principal en el navegador la cuál será la base para el resto del sistema por lo que las pruebas para la interfaz se hicieron cargando la página con distintos navegadores y un emulador de celular, observando y comprobando si la página cargada sigue lo definido en el diseño.

Se obtuvieron resultados según lo esperado, sin fallas gráficas entre los navegadores web.

4.4.3. Iteración 2: Página de inicio

El objetivo de esta iteración es implementar la sección inicial de la aplicación, esta sección será la vista principal que se le mostrará al usuario al ingresar en la aplicación, por lo cuál debe mostrar botones de acceso a todas las funcionalidades de la aplicación. Las mismas se resumen en acceso a funcionalidades de muestra de inmuebles relevantes y acceso al buscador de inmuebles.

- Planificación

Nro.	Fecha Inicio	Descripción
5.0	27/08/12	Definir las funcionalidades para el menú de esta sección.
6.0	27/08/12	Diseñar la interfaz gráfica para la sección.
7.0	28/08/12	Implementar el código necesario para soportar las funcionalidades definidas (incluye: Código PHP, CSS, JS y HTML).

- Diseño

Como se trata de una aplicación de búsqueda de inmuebles, se debe permitir al usuario llegar a la información de los mismos de manera rápida, por lo que se desea incluir en la página de inicio funcionalidades de acceso a inmuebles relevantes. Se trata del acceso rápido a los inmuebles agregados recientemente y los inmuebles más visitados.

A continuación se muestra la imagen de la interfaz gráfica diseñada para la sección de inicio.

Figura 4.6: Diseño de la página de inicio

- Codificación

La implementación de la página de inicio se puede ver en la siguiente imagen:

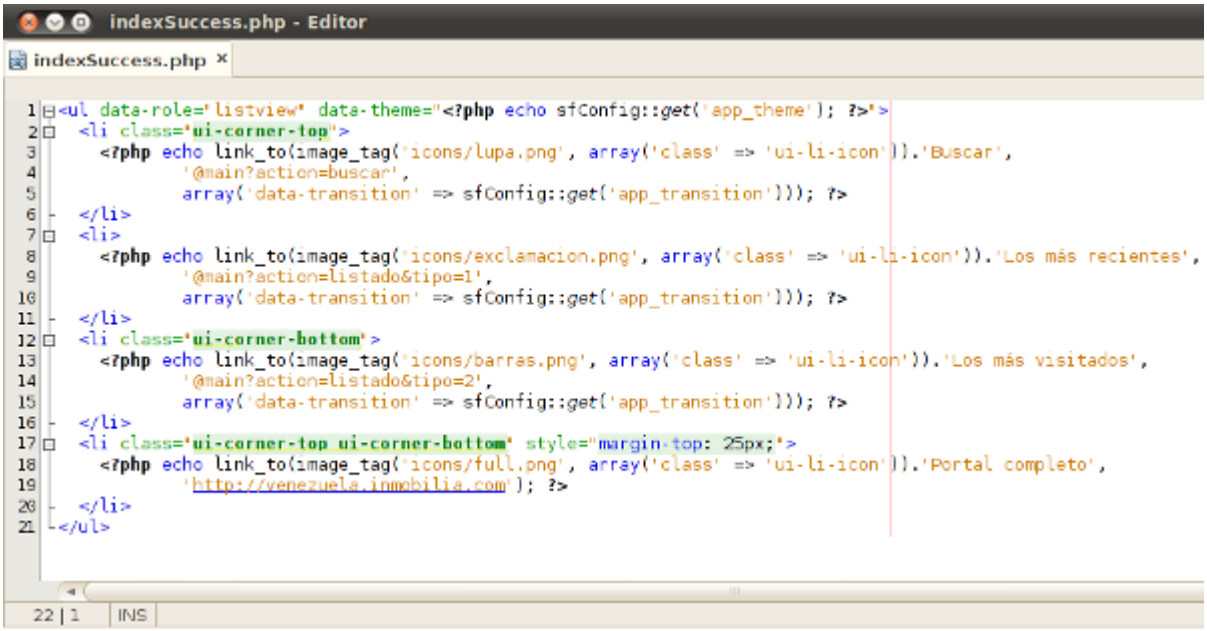


Figura 4.7: Código de las opciones del menú del *home*

Para el caso de la página de inicio sólo se requirió código en la capa de la vista. En la imagen se observan las opciones de navegación del menú de la página de inicio, en la línea 2 se encuentra la opción para ir al buscador, en la línea 7 la opción para ver los inmuebles más recientes y en la línea 17 la opción para los inmuebles más visitados. Cada una de las opciones contiene el código necesario para ser interpretado por *JQuery Mobile* y convertirlo en un botón similar a una aplicación de celular nativa.

Este trozo de código se combina con el código del *layout* de la iteración anterior para generar la vista diseñada de la página de inicio.

- Pruebas

Las pruebas de aceptación para la interfaz se hicieron cargando la página con distintos emuladores de celulares y un celular marca Blackberry, observando y comprobando si la página cargada sigue lo definido en el diseño.

Se obtuvieron los resultados esperados, sin fallas gráficas entre los navegadores web ni en las URL´s de los botones.

4.4.4. Iteración 3: Listado de inmuebles

En esta iteración se desea implementar la sección de listado de inmuebles. Este es el listado principal de la aplicación y el mismo será reutilizado por varias funcionalidades de la aplicación como: inmuebles recientes, inmuebles más visitados y los resultados de búsqueda de los inmuebles. Esto es posible dado que cada funcionalidad consulta la misma tabla de inmuebles en base de datos.

- Planificación

Nro.	Fecha Inicio	Descripción
8.0	29/08/12	Diseño de la interfaz gráfica del listado.
9.0	30/08/12	Implementación de la funcionalidad de inmuebles recientes (incluye: código PHP, CSS, JS y HTML).
10.0	31/08/12	Implementación de la funcionalidad de inmuebles más visitados (incluye: código PHP, CSS, JS y HTML).

- Diseño

La sección desarrollada en esta iteración presenta un listado de inmuebles al usuario según los parámetros seleccionados por el mismo. En este caso se soportan las funcionalidades de "Inmuebles recientes" e "Inmuebles más visitados" utilizando un mismo listado y sólo cambiando los inmuebles a mostrar según lo indique el parámetro "tipo". El parámetro mencionado indica el tipo de procesamiento que se debe hacer para seleccionar los inmuebles del listado, si el parámetro tipo es igual a "1", entonces se listan los inmuebles más recientes y si el tipo es igual a "2" se listan los inmuebles más visitados.

- Codificación

Dado que esta iteración comprende el desarrollo de dos funcionalidades, se muestra la codificación de las porciones de código que tienen en común y también se muestran las porciones de código particulares de cada funcionalidad. A continuación se muestra el código de la **capa controlador**:

actions.class.php - Editor

actions.class.php

```
22
23    public function executeListado(sfWebRequest $request)
24    {
25      $this->setLayout('layout');
26
27      switch($request->getParameter('tipo')){
28        case 1:  //Los mas recientes
29          $this->titulo = 'Inmuebles más recientes (10)';
30          $this->items = InmuebleUsadosPeer::getInmueblesMasRecientes();
31          break;
32
33        case 2:  //Los mas visitados
34          $this->titulo = 'Inmuebles más visitados (10)';
35          $this->items = InmuebleUsadosPeer::getInmueblesMasVisitados();
36          break;
37      }
38    }
39
```

Figura 4.8: Listado de inmuebles (código del controlador)

En la imagen se aprecia el código necesario (según la estructura de symfony), para recibir y procesar la petición HTTP del cliente. En la línea 27 inicia un *switch* el cuál según el parámetro recibido del cliente selecciona el tipo de procesamiento a realizar. Para el caso de los "Inmuebles recientes", el procesamiento se realiza en la línea 30, donde se realizará la llamada al siguiente código en la **capa del modelo**:

InmuebleUsadosPeer.php - Editor

InmuebleUsadosPeer.php

```
41    //Devuelve los inmuebles mas recientes
42    public static function getInmueblesMasRecientes($limit = 10)
43    {
44      $c = new Criteria();
45      $c->add(self::ID_PAISES, 1, Criteria::EQUAL);
46      $c->add(self::ELIMINADO, 0, Criteria::EQUAL);
47      $c->add(self::STATUS,'Activo',Criteria::EQUAL);
48      $c->setLimit($limit);
49      $c->addDescendingOrderByColumn('fecha_ult_mod');
50
51      return self::doSelect($c);
52    }
53
```

Figura 4.9: Inmuebles más recientes (código del modelo)

En la figura se observa el código implementado para generar la consulta SQL correspondiente utilizando la clase *criteria*. Básicamente en esta porción de código se definen los criterios de la consulta SQL a realizar, el ordenamiento que tendrá el resultado, cantidad de registros a retornar y se realiza la búsqueda en base de datos devolviendo los inmuebles que hayan sido agregados al sistema recientemente. Para el caso de los inmuebles más visitados se utiliza un código similar al mostrado en la imagen.

Finalmente el resultado de la consulta es recibido por el controlador y suministrado a la vista, donde se encarga de imprimir cada resultado como se ve en el código en la **capa de la vista**:

listadoSuccess.php - Editor

listadoSuccess.php

```
15      <?php include_partial('partials/pager', array('items' => $items)); ?>
16    <?php endif; ?>
17  <?php endif; ?>
18
19  <ul data-role="listview" style="margin: 0;" <?php if($sf_request->getParameter('tipo') ==
20    <?php foreach($items as $item): ?>
21      <li id="li_<?php echo $item->getIdInmuebleUsados(); ?>">
22        <a href="<?php echo url_for('@main?action=detalle&id_inmueble='.$item->getIdInmuebl
23          <div style="height: 60px; line-height: 17px;">
24            <?php $source = inmobilia::getRutaImagenPrincipalInmueble($item); ?>
25            <?php echo image_tag($source, array('class' => 'ui-li-thumb', 'style' => 'margi
26            <span class="texto_azul" style="padding-left: 85px;"><?php echo $item->getNombr
27            <i class="texto_rojo" style="padding-left: 85px; font-size: 11px;">Código: (<?p
28            <i style="font-size: 11px; padding-left: 85px;">Urbanizacion: <?php echo $item-
29            <br/><i style="font-size: 11px; padding-left: 85px;">En <?php echo $item->getDe
30          </div>
31
32          <div class="list_icons">
33            <table id="" style="font-size: 10px;">
34              <tr>
35                <td class="list_precio"> </td>
36                <td style="border-right: 1px solid #D3D3D3"><?php echo $item->getPrecio();
37
38                <td class="list_habs"> </td>
39                <td style="border-right: 1px solid #D3D3D3"><?php echo $item->getNumHab();
40
41                <td class="list_bano"> </td>
42                <td style="border-right: 1px solid #D3D3D3"><?php echo $item->getNumBanios(
43
44                <td class="list_metros"> </td>
45                <td style="border-right: 1px solid #D3D3D3"><?php echo $item->getAreaConst(
46
47                <td class="list_estac"> </td>
48                <td><?php echo $item->getNumPtoEstaciona(); ?></td>
49              </tr>
50            </table>
51          </div>
52        </a>
```

Figura 4.10: Listado de inmuebles (código de la vista)

Cada uno de los registros es un inmueble (almacenados en la variable *$items*, línea 6). En el listado se muestra la información más relevante de cada inmueble, como el título del inmueble, su ubicación y número de habitaciones, baños, entre otros.

Haciendo uso de los elementos de *JQuery Mobile*, se coloca cada inmueble como un elemento "*Linked List*". Este tipo de elemento permite al usuario seleccionar un inmueble tocando el mismo con el dedo en la pantalla de su dispositivo móvil *touch* y así poder acceder a la información detallada del inmueble (desarrollado en una próxima iteración).

- Pruebas

Se realizaron pruebas de carga y funcionalidad donde se verifica directamente en base de datos cuáles serían los inmuebles que se esperan obtener como resultado y comparándolo por el resultado obtenido en cada una de las funcionalidades desarrolladas en esta iteración. Se comprobó que los inmuebles obtenidos eran los que se esperaban.

También se validó con un celular BlackBerry que se realice correctamente la carga del listado con los elementos de jQuery, se obtuvieron los resultados esperados y se corrigieron algunas fallas gráficas y en el formato del texto.

4.4.5. Iteración 4: Detalle del inmueble

En esta iteración se implementa la sección para la muestra de toda la información de un inmueble clasificada en: información de contacto, ubicación y características generales.

- Planificación

Nro.	Fecha Inicio	Descripción
11.0	3/09/12	Diseño de la interfaz gráfica del detalle.
12.0	4/09/12	Integración gráfica del diseño y sus elementos de jQuery.
13.0	4/09/12	Implementación para mostrar la información de contacto del inmueble.
14.0	5/09/12	Implementación para mostrar la información de la ubicación del inmueble.
15.0	6/09/12	Implementación para mostrar la información de las características del inmueble.

- Diseño

Cada inmueble puede poseer gran cantidad de información que ocupe mucho espacio en la pantalla de un dispositivo móvil. Por esta razón, el diseño de la interfaz gráfica se basa en los elementos proporcionados por *JQuery Mobile*. El elemento clave para estructurar la información es el denominado "*Accordion*", el mismo permite agrupar información que sea de la misma índole, por ejemplo: agrupar la información de la ubicación del inmueble, país, estado, ciudad, otras. Además también permite mostrar y ocultar cada grupo de información para reducir el espacio vertical que se requiere para mostrar la misma.

A nivel funcional, para poder mostrar la información de un inmueble de la manera más simple posible, en esta sección se manipula únicamente un objeto inmueble el cuál se obtendrá a partir de un valor de *id* del inmueble proporcionado por la URL solicitada. Utilizando este objeto se obtiene el resto de información relacionada, ubicación, contactos, otras.

- Codificación

Para poder acceder al detalle de un inmueble se debe proveer un parámetro que es el *id* del inmueble. Este *id* es único y permite ubicar el inmueble en la base de datos para obtener su información. A continuación se muestra la codificación necesaria para mostrar la información del detalle del inmueble:

actions.class.php - Editor

actions.class.php ×

```
44  public function executeDetalle(sfWebRequest $request)
45  {
46    $this->setLayout('layout');
47    if($request->isMethod('post')){
48      $this->buscarPorCodigo($request);
49    }else{
50      if($request->getParameter('id_inmueble') && is_numeric($request->getParameter('id_inmueble'))
51      {
52        $this->inmueble = InmuebleUsadosPeer::retrieveByPK($request->getParameter('id_inmueble'));
53
54        if(!$this->inmueble){
55          $this->inmueble = null;
56        }
57      }else{
58        $this->redirect('@main?action=buscar');
59      }
60    }
61  }
62
```

Figura 4.11: Código del detalle del inmueble (capa controlador)

En la figura se encuentra el procesamiento inicial que se realiza al querer ingresar al detalle de cualquier inmueble, en la línea 50, se obtiene el parámetro *id* del inmueble para realizar la búsqueda del objeto en base de datos utilizando *Propel*, se coloca el resultado en la variable *$this->inmueble* (línea 52), la misma será utilizada por la vista para mostrar la información correspondiente.

detalleSuccess.php - Editor

detalleSuccess.php

```
1  <?php if($inmueble): ?>
2    <div class='detalle_content'>
3      <div class='title_content' style='font-weight: bold;'>
4        <h2 style='padding: 0;'>
5          <span class='texto_azul'><?php echo $inmueble->getNombre(); ?></span>
6        </h2>
7        <br/><i class='texto_rojo' style='font-size: 11px;'>Código: (<?php echo $inmueble->getIdInmuebleUsados
8        <br/><i style='font-size: 11px;'>Urbanizacion: <?php echo $inmueble->getUrbanizaciones()->getDescripUr
9        <br/><i style='font-size: 11px;'>En <?php echo $inmueble->getDestinoInmueble(); ?></i>
10     </div>
11
12     <div class='social_content'>
13       <?php include_partial('social'); ?>
14     </div>
15
16     <div style='padding-top: 10px;'>
17       <?php include_partial('imagenes_inmueble', array('inmueble' => $inmueble)); ?>
18     </div>
19   </div>
20
21   <div class='detalle_acordiones'>
22     <?php include_partial('acordiones', array('inmueble' => $inmueble)); ?>
23   </div>
24
25   <div class='detalle_content' style='margin-top: 20px;'>
26     <?php include_partial('social'); ?>
27   </div>
28 <?php else: ?>
29   <div class='detalle_content'>
30     <div class='title_content' style='font-weight: bold;'>
31       <h2 style='line-height: 0; padding: 0;'>
32         <span class='texto_azul'>Inmueble no encontrado</span>
33       </h2>
34     </div>
35   </div>
36 <?php endif; ?>
37
```

Figura 4.12: Código del detalle del inmueble (capa vista)

En la imagen se aprecia cómo se presenta la información del inmueble obtenido en la imagen anterior. Haciendo uso de la variable *$inmueble*, se obtiene cada característica del inmueble. Importante por mencionar en la imagen es cada agrupación de datos, en la línea 22, se encuentra el código necesario para imprimir la información del contacto del inmueble, para poder determinar quién es el usuario propietario del inmueble se verifican los valores de *id_ agente*, *id_ inmobiliarias* y *id_ particular* según sea el caso, obteniendo los datos desde otro objeto distinto al inmueble. En la siguiente imagen se muestra la codificación necesaria para determinar el dueño del inmueble:

InmuebleUsadosPeer.php - Editor

InmuebleUsadosPeer.php

```
52    //Devuelve los datos del duenio del inmueble
53    public static function getDuenoInfo($idInmueble)
54    {
55      $datos = null;
56      $inmueble = self::retrieveByPK($idInmueble);
57
58      if($inmueble->getIdAgente()){
59        $datos = AgentePeer::getAgenteInfo($inmueble->getIdAgente());
60      }elseif($inmueble->getIdInmobiliaria()){
61        $datos = InmobiliariasPeer::getInmobiliariaInfo($inmueble->getIdInmobiliaria());
62      }else{
63        $datos = ParticularesPeer::getParticularInfo($inmueble->getIdInmuebleUsados());
64      }
65
66      return $datos;
67    }
68
```

Figura 4.13: Código para determinar el dueño del inmueble

En la figura se muestra como se realiza la selección del objeto que es dueño del inmueble, en la línea 58 se realizan validaciones para seleccionar el objeto correcto y devolverlo al controlador para que pueda mostrarse su información. Los objetos que pueden ser dueño de un inmueble se clasifican en agente, inmobiliarias y particulares. Cada uno de estos tipos de objetos posee distintos métodos para acceder a su información.

- Pruebas

Se determinó un conjunto de inmuebles válidos y se probó ingresar a la sección de detalle del inmueble pasando como parámetro los *ids* de los inmuebles del conjunto seleccionado. Se validó que la información que aparece en la sección sea la misma que se encuentra al buscar ese *id* directamente en base de datos. También se validó que se seleccione correctamente el objeto que es dueño del inmueble y que se muestre la información correcta del mismo.

4.4.6. Iteración 5: Implementación de buscador por código

En esta iteración se inicia la implementación de la sección para el buscador de inmuebles. Comprende actividades de generación de la sección diseño e integración de la interfaz gráfica y codificación de la funcionalidad de búsqueda.

- Planificación

Nro.	Fecha Inicio	Descripción
16.0	10/09/12	Diseño de la interfaz gráfica del buscador por código.
17.0	12/09/12	Implementación de manejo de la solicitud de búsqueda por código.
18.0	13/09/12	Integración gráfica, de los elementos del buscador.
19.0	14/09/12	Validación y muestra de errores en el formulario.

- Diseño

Para el caso de la búsqueda de inmuebles por código, el usuario debe colocar un código de inmueble que permita determinar de manera concluyente el objeto que desea visualizar. Una vez que se ingrese un código válido, entonces se debe proceder a mostrar la información del mismo (utilizando la sección de "Detalle del inmueble"). En caso contrario debe mostrarse un mensaje de error.

En el siguiente diagrama se describe el comportamiento mencionado:

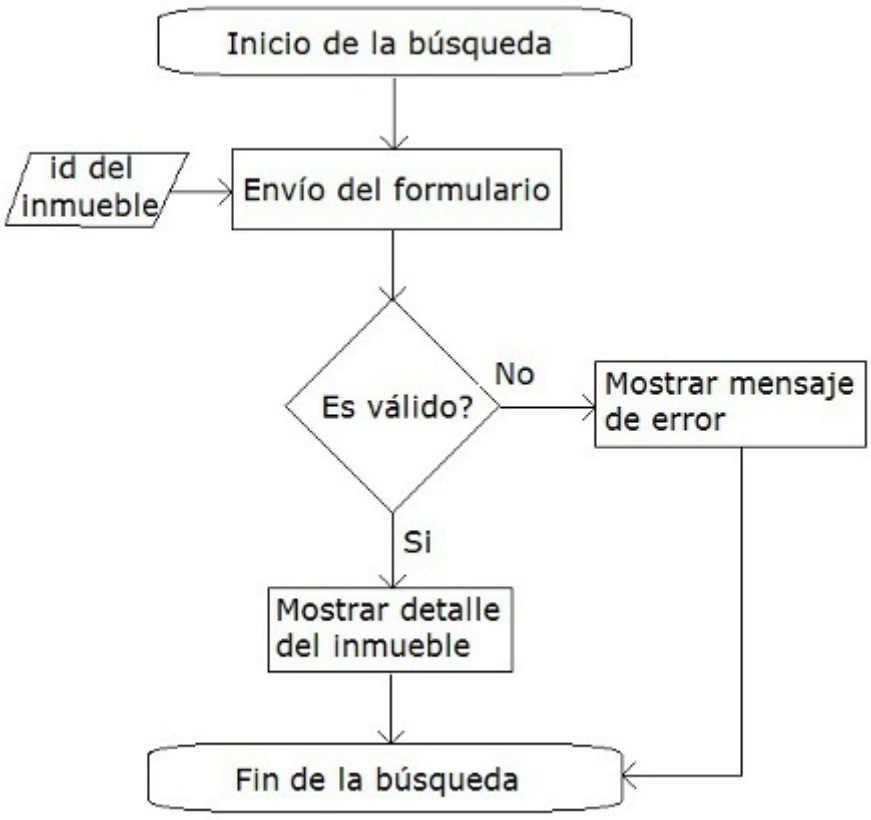


Figura 4.14: Diagrama de flujo del buscador por código

- Codificación

Uno de los pasos principales para implementar el buscador por código, es la elaboración de la clase del formulario. Symfony en la estructura que tiene definida, indica que para mejorar el manejo de los formularios, los mismos deben manejar toda su lógica en una clase aparte. A continuación se muestra la imagen de la clase del formulario para el buscador por código.

BuscadorPorCodigoForm.class.php - Editor

BuscadorPorCodigoForm.class.php ×

```
1  <?php
2
3  /**
4   * Localidades form.
5   *
6   * @package    inmobiliamobile
7   * @subpackage form
8   * @author     Cesar Leon
9   */
10 class BuscadorPorCodigoForm extends sfForm
11 {
12   public function configure()
13   {
14     //Widget
15     $this->widgetSchema['codigo'] = new sfWidgetFormInputText(array('type'
16
17     //Validores
18     $this->validatorSchema['codigo'] = new sfValidatorNumber(
19             array('required' => true, 'min' => 1),
20             array('required' => 'Código requerido',
21                   'invalid' => 'Solo números',
22                   'min' => 'Solo números positivos'));
23
24     //Labels
25     $this->widgetSchema->setLabels(array(
26       'codigo' => 'Codigo del Inmueble <span class="texto_rojo">*</span>'
27     ));
28
29     $this->widgetSchema->setNameFormat('buscador[%s]');
30   }
31 }
32
```

Figura 4.15: Código de la clase del formulario (búsqueda por código)

En la imagen se aprecia el código necesario para crear el formulario a utilizar por el buscador por código. En esta clase se define el tipo de campo a mostrar (campos de texto, numéricos, otras), como se puede ver en la línea 15 donde se declara un campo de tipo texto. También se definen las validaciones que se le debe realizar a los campos, en las líneas 18 a 22 se define que el campo es requerido y que sólo debe tener números positivos.

Con el formulario completado ya se puede instanciar el mismo para ser utilizado, en la

siguiente imagen se observa el código necesario para cargar y procesar el formulario del buscador por código:

actions.class.php - Editor

actions.class.php

```
  public function executeBuscar(sfWebRequest $request)
88 {
89   $this->setLayout('layout');
90   $this->acordion = 2;
91
92   $this->form = new BuscadorPorCodigoForm();
93   $this->formAvanzado = new BuscadorAvanzadoForm();
94 }
95
96 protected function buscarPorCodigo(sfWebRequest $request)
97 {
98   $this->acordion = 1;
99   $this->form = new BuscadorPorCodigoForm();
100
101  $this->form->bind($request->getParameter($this->form->getName()),
102                    $request->getFiles($this->form->getName())
103                   );
104
105  if(!$this->form->isValid()){
106    $this->setTemplate('buscar');
107    $this->formAvanzado = new BuscadorAvanzadoForm();
108  }else{
109    //Se busca el inmueble
110    $this->inmueble = InmuebleUsadosPeer::retrieveByPK($this->form->getValue('
111
112    if(!$this->inmueble){
113      $this->inmueble = null;
114    }
115  }
116 }
117
```

Figura 4.16: Código de la carga y procesamiento del buscador por código

En la imagen se observa al principio (líneas 88 a 94), las instrucciones necesarias para instanciar el formulario del buscador y cargar la sección del buscador.

También en las líneas 96 a 113 se encuentran las instrucciones necesarias para el procesamiento de la búsqueda por código. En la línea 102 se valida los campos del formulario a través de la instrucción *$this->form->isValid()*. Esta instrucción es proporcionada por la estructura de las clases de formulario de symfony como un método sencillo y rápido para ejecutar las validaciones correspondientes a un formulario en el lado del servidor. Como se describió en el diseño, si el formulario es válido se procede a buscar el *id* del inmueble en base de datos (línea 107), en caso contrario se muestra el mensaje de error correspondiente.

- Pruebas

Para la probar la búsqueda por código se contemplan 3 casos:

1. El caso donde exista un inmueble con el código introducido.
2. El caso donde no exista un inmueble con el código introducido.
3. El caso donde el código sea inválido.

Cada uno de estos casos es probado por separado:

- Caso 1: Se introdujeron varios códigos de inmuebles válidos y se comprobó que se cargará la sección del detalle del inmueble mostrando la información correcta del inmueble.
- Caso 2: Se introdujeron varios códigos de inmuebles que no existen o que estén desactivados y se obtuvo un mensaje de error como era esperado.
- Caso 3: Para el caso de un código inválido se debe mostrar un mensaje de error al usuario indicándole el error en el mismo, luego de realizar la prueba con varios códigos inválidos se obtuvo que este comportamiento se cumplió como se esperaba.

4.4.7. Iteración 6: Implementación de búsqueda por localidad.

En esta iteración se inicia la implementación de la sección para el buscador por localidad de los inmuebles. Comprende actividades de generación de la sección diseño e integración de la interfaz gráfica y codificación de la funcionalidad de búsqueda por localidad.

- Planificación

Nro.	Fecha Inicio	Descripción
20.0	15/09/12	Diseño de la interfaz gráfica del buscador por localidad.
21.0	16/09/12	Evaluación de los criterios relevantes de búsqueda de inmuebles.
22.0	16/09/12	Implementación de la clase del formulario correspondiente.
23.0	17/09/12	Integración gráfica de los elementos del buscador por localidad.
24.0	18/09/12	Implementación de funcionalidad de recarga de “selects” anidados para las localidades.
25.0	19/09/12	Implementación de acción para el proceso de búsqueda.
26.0	20/09/12	Validación y muestra de errores en el formulario.

- Diseño

Para realizar la búsqueda por localidad, el usuario debe especificar cada parámetro de localidad (estado, ciudad y urbanización), esto es una premisa obligatoria según las reglas de negocio que cumple Inmobilia.com. Al seleccionar cada parámetro de la localidad se recargan los valores de la sublocalidad correspondiente, por ejemplo: al seleccionar un estado se recargan los valores de la lista de ciudades y al seleccionar una ciudad se recargan los valores de la lista de urbanizaciones, esto con el objetivo de realizar búsquedas solamente en las localidades que tengan inmuebles.

En la siguiente figura se observa el diseño de la interfaz gráfica para el buscador por localidad:

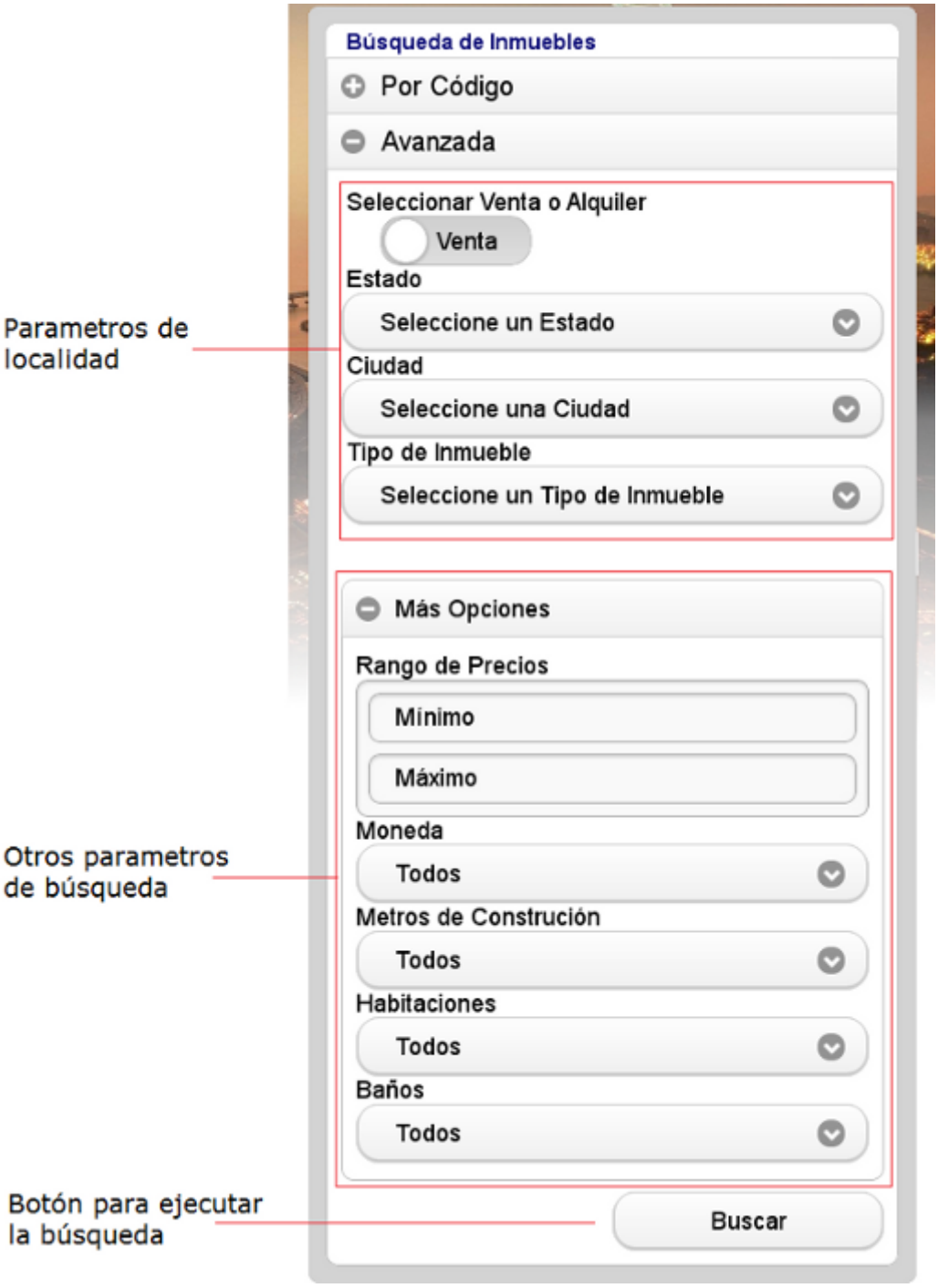


Figura 4.17: Diseño de la interfaz gráfica del buscador por localidad

El área para "otros parámetros de búsqueda", se refiere a los parámetros adicionales del buscador, los cuáles serán desarrollados en una próxima iteración. Al colocar los parámetros de localidad y realizar una búsqueda, se debe mostrar un listado de resultados (reutilizando el listado implementado en una iteración anterior), en caso contrario se le debe mostrar al usuario los mensajes de error correspondientes. En la siguiente figura se describe el comportamiento que debe tener el buscador por localidad:

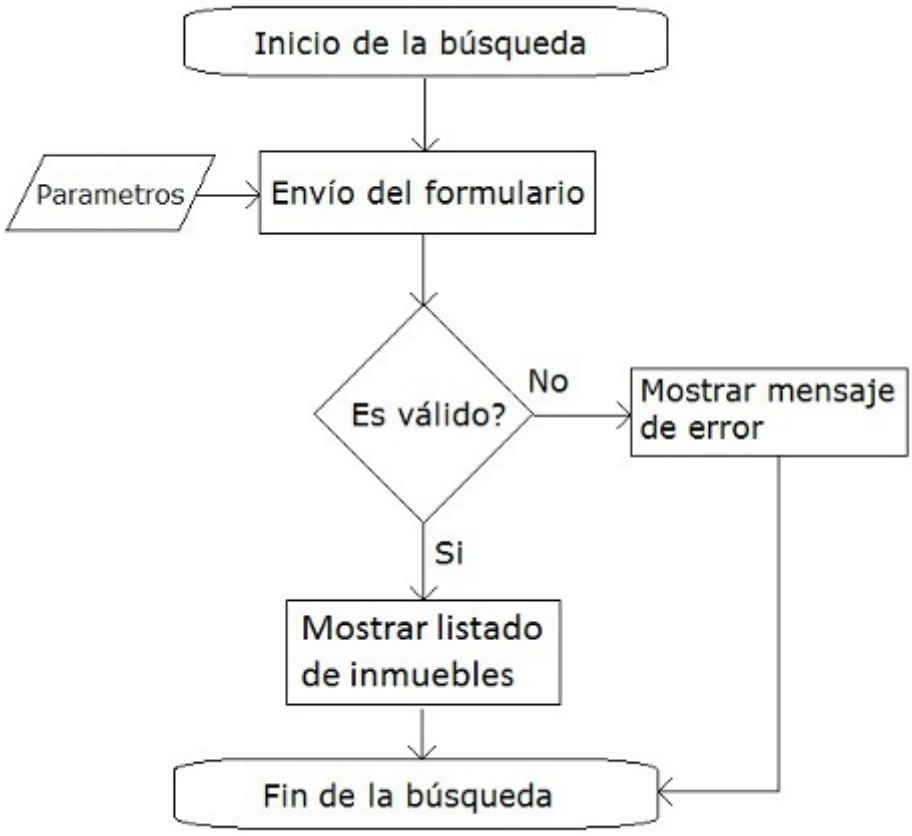


Figura 4.18: Diagrama de flujo del buscador por localidad

- Codificación

La primera parte de la codificación para esta iteración se encuentra en la clase del formulario necesaria según la estructura de Symfony. Ver imagen a continuación:

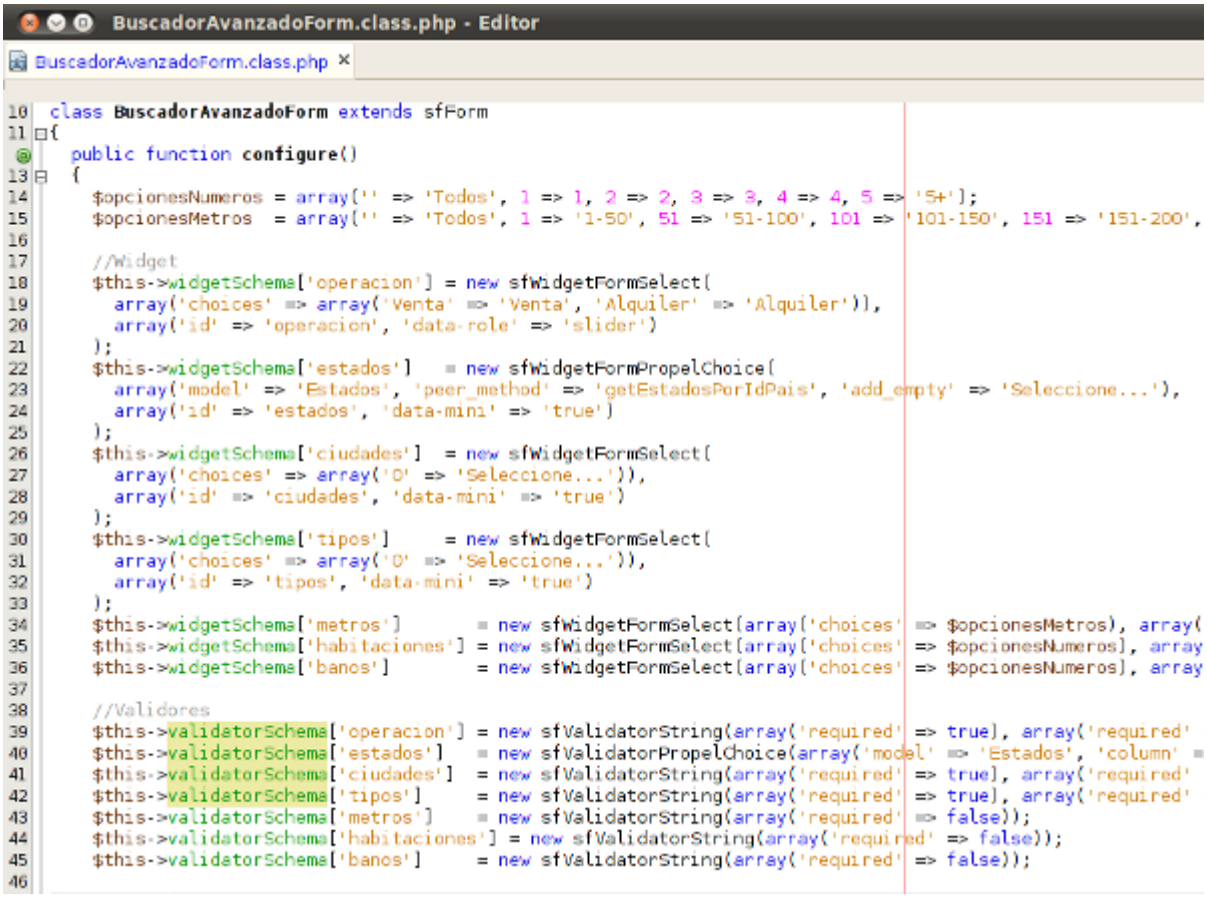


Figura 4.19: Código de la clase del formulario (búsqueda por localidad)

En la imagen se puede ver la definición de los parámetros de localidad, entre las líneas 17 a 33, indicando que se trata de un campo de tipo lista desplegable (*sfFormChoice*). Entre las líneas 38 a 45 se encuentran las validaciones a realizar para los campo mencionados anteriormente, indicando que los campos son requeridos.

A continuación se muestra el código que se encarga de procesar la solicitud de búsqueda por localidad:

actions.class.php - Editor

actions.class.php

```
115   protected function busquedaAvanzada(sfWebRequest $request)
116   {
117     $this->acordion = 2;
118     $this->formAvanzado = new BuscadorAvanzadoForm();
119
120     $this->formAvanzado->bind($request->getParameter($this->formAvanzado->getName()
121
122     //Verificacion para soporte de boton volver
123     if($request->isMethod('post')){
124       $this->getUser()->setAttribute('formAvanzado', $this->formAvanzado);
125     }elseif($this->getUser()->getAttribute('formAvanzado')){
126       $this->formAvanzado = $this->getUser()->getAttribute('formAvanzado');
127     }else{ $this->redirect('@main?action=buscar'); }
128
129     if(!$this->formAvanzado->isValid()){
130       $this->setTemplate('buscar');
131       $this->form = new BuscadorPorCodigoForm();
132     }else{
133       //Se definen los valores de la pagina del listado
134       if($request->getParameter('page')){
135         $page = $request->getParameter('page');
136       }else{ $page = 1; }
137
138       //Se buscan los inmuebles
139       $c = InmuebleUsadosPeer::getInmueblesBuscardorAvanzado($this->formAvanzado);
140       $pager = new sfPropelPager('InmuebleUsados', 10);
141       $pager->setCriteria($c);
142       $pager->setPage($page);
143       $pager->init();
144
145       $this->items = $pager;
146
147       //Se define el titulo de la pagina
148       if($this->formAvanzado->getValue('operacion') == 'Venta'){
149         $this->titulo = 'Inmuebles en venta ('.$this->items->getNbResults().')';
150       }else{
151         $this->titulo = 'Inmuebles en alquiler ('.$this->items->getNbResults().')';
152       }
153     }
```

Figura 4.20: Código de la carga y procesamiento del buscador por localidad

Para esta funcionalidad la misma función que se encarga de mostrar la sección del buscador también permite procesar la solicitud de búsqueda. Desde las líneas 115 a 120 se realizan las instrucciones necesarias para cargar la sección del buscador y desde las líneas 123 a 153 se encuentra el procesamiento del formulario de búsqueda. En la línea 129 se comprueba que el formulario tenga la validación correcta, para este caso se trata sólo de comprobar que los campos no estén vacíos.

En la línea 143 se realiza la consulta a base de datos de los inmuebles que cumplan con los parámetros seleccionados. A continuación el código del modelo:

InmuebleUsadosPeer.php - Editor

InmuebleUsadosPeer.php

```
94   public static function getInmueblesBuscardorAvanzado($form)
95   {
96     $operacion = $form->getValue('operacion');
97     $estado = $form->getValue('estados');
98     $ciudad = $form->getValue('ciudades');
99     $tipo = $form->getValue('tipos');
100
101    //Parametros adicionales
102    $metros = $form->getValue('metros');
103    $habitaciones = $form->getValue('habitaciones');
104    $banos = $form->getValue('banos');
105
106    $c = new Criteria();
107    $c = self::caracteristicasPorDefecto($c);
108    $c->addDescendingOrderByColumn('fecha_ult_mod');
109    $c->add(self::DESTINO_INMUEBLE, $operacion, Criteria::EQUAL);
110    $c->add(self::ID_ESTADOS, $estado, Criteria::EQUAL);
111    $c->add(self::ID_CIUDADES, $ciudad, Criteria::EQUAL);
112    $c->add(self::ID_TIPO_INMUEBLE, $tipo, Criteria::EQUAL);
113
114    //Manejo del parametro metros
115    if($metros){
116      switch($metros){
117        case 201:
118          $c->add(self::AREA_CONST, $metros, Criteria::GREATER_EQUAL);
119          break;
120        default:
121          $metrosMax = $metros + 49; //Se asigna el valor del limite superior
122          $c->add(self::AREA_CONST, $metros, Criteria::GREATER_EQUAL);
123          $c->addAnd(self::AREA_CONST, $metrosMax, Criteria::LESS_EQUAL);
124          break;
125      }
126    }
127
128    //Manejo del parametro habitaciones
129    if($habitaciones){
130      if($habitaciones < 5){
131        $c->add(self::NUM_HAB, $habitaciones, Criteria::EQUAL);
132      }else{
```

Figura 4.21: Código de la búsqueda de inmuebles del buscador por localidad

En esta porción de código se realiza la búsqueda de todos los inmuebles que correspondan con los criterios de búsqueda seleccionados. En la línea 109 se asignan las restricciones por defecto de cualquier inmueble para poder mostrarse, tal es el caso del estatus del inmueble y que se encuentre en el país Venezuela. En las líneas 109 a 112 se asignan a la consulta los parámetros de localidad seleccionados por el usuario.

Finalmente toda la información de los inmuebles obtenida desde base de datos es recibida por la vista y utilizada para mostrar la información de los mismos utilizando el mismo código del listado de inmuebles (mostrado en una iteración anterior).

- Pruebas

Se realizaron pruebas con el buscador por localidad comparando el resultado obtenido contra el resultado de ejecutar la consulta directamente en base de datos. En este proceso se encontraron discrepancias entre los resultados obtenidos, por lo que se realizaron correcciones y pruebas repetidas veces hasta que ambos resultados estuvieron correctos.

También se realizaron pruebas en las recargas de las listas desplegables anidadas, validando contra la base de datos que las sublocalidades cargadas eran las que se esperaban cargar. Se obtuvieron los resultados esperados en estas pruebas, pudiendo comprobar que las localidades cargadas son correctas.

4.4.8. Iteración 7: Implementación de parámetros adicionales para el buscador por localidad (recodificación)

En esta iteración se plantea implementar parámetros adicionales en el buscador por localidad realizado en la iteración anterior. Estos parámetros son los mismos que se encuentran en el sistema Inmobilia.com. Se trata de los parámetros de número de habitaciones, baños, metros cuadrados y rango de precios.

- Planificación

Nro.	Fecha Inicio	Descripción
27.0	21/09/12	Integración gráfica de los parámetros diseñados.
28.0	24/09/12	Implementación del parámetro número de habitaciones.
29.0	25/09/12	Implementación del parámetro número de baños.
30.0	26/09/12	Implementación del parámetro metros cuadrados.
31.0	27/09/12	Implementación del parámetro rango de precios.

- Diseño

Para realizar de manera sencilla la implementación de los nuevos parámetros, cada uno de los mismos agregara una condición adicional al *query* de búsqueda de los inmuebles, apoyándose en la búsqueda que ya se encuentra implementada y en funcionamiento. Conociendo esto se logra realizar la recodificación de pocas partes del código.

- Codificación

Siguiendo la estructura de Symfony se agregan los nuevos campos de parámetros a la clase del formulario del buscador por localidad al igual que sus validadores correspondientes (clase del formulario implementada en la iteración anterior).

Para manipular los nuevos parámetros no hace falta modificar el código presente en el controlador y la vista, sin embargo, para el caso del modelo se debe agregar cada parámetro

al *query* de búsqueda, sólo en caso de que el usuario haya seleccionado alguno de ellos, a continuación la recodificación del código del modelo:

InmuebleUsadosPeer.php - Editor

InmuebleUsadosPeer.php

```
110     $c->add(self::ID_ESTADOS, $estado, Criteria::EQUAL);
111     $c->add(self::ID_CIUDADES, $ciudad, Criteria::EQUAL);
112     $c->add(self::ID_TIPO_INMUEBLE, $tipo, Criteria::EQUAL);
113
114     //Manejo del parametro metros
115     if($metros){
116       switch($metros){
117         case 201:
118           $c->add(self::AREA_CONST, $metros, Criteria::GREATER_EQUAL);
119           break;
120         default:
121           $metrosMax = $metros + 49; //Se asigna el valor del limite superior
122           $c->add(self::AREA_CONST, $metros, Criteria::GREATER_EQUAL);
123           $c->addAnd(self::AREA_CONST, $metrosMax, Criteria::LESS_EQUAL);
124           break;
125       }
126     }
127
128     //Manejo del parametro habitaciones
129     if($habitaciones){
130       if($habitaciones < 5){
131         $c->add(self::NUM_HAB, $habitaciones, Criteria::EQUAL);
132       }else{
133         $c->add(self::NUM_HAB, $habitaciones, Criteria::GREATER_EQUAL);
134       }
135     }
136
137     //Manejo del parametro banos
138     if($banos){
139       if($banos < 5){
140         $c->add(self::NUM_BANIOS, $banos, Criteria::EQUAL);
141       }else{
142         $c->add(self::NUM_BANIOS, $banos, Criteria::GREATER_EQUAL);
143       }
144     }
145
146     return $c;
147   }
148 } // InmuebleUsadosPeer
```

Figura 4.22: Código de los parámetros adicionales del buscador por localidad

Como se puede observar en la imagen, cada parámetro, si existe, es agregado al *query* de búsqueda por localidad utilizando la clase *Propel* proporcionada por Symfony, en las líneas 115 a 126 se encuentra el parámetro de metraje, manejo del número de habitaciones en las líneas 129 a 135 y número de baños en las líneas 138 a 144.

- Pruebas

Para probar la implementación realizada se plantean los siguientes casos:

Caso 1: Sólo un parámetro es seleccionado. En este caso se prueba cada parámetro por separado, la idea es validar que al seleccionar un valor de parámetro el resultado obtenido cumpla con lo seleccionado. Por ejemplo: si el usuario selecciona que los inmuebles a buscar sólo tengan 2 habitaciones, se valida que todos los resultados obtenidos tengan sólo 2 habitaciones, ni más, ni menos. Los resultados obtenidos al probar cada parámetro por separado son correctos, cada resultado de búsqueda muestra sólo las ocurrencias que cumplen con los parámetros seleccionados por el usuario.

Caso 2: Se seleccionan varios parámetros en la misma búsqueda. La prueba en este caso se realiza seleccionando cualquier combinación de 2 o más parámetros de búsqueda a la vez. Para poder probar que la búsqueda funciona correctamente se deben probar búsquedas como: inmuebles que tengan 2 habitaciones, 2 baños y entre 51-100 metros cuadrados de construcción.

Se realizó la prueba probando múltiples combinaciones de parámetros y en todos los casos validando que cada inmueble cumpla con todos los parámetros seleccionados. Se obtuvo que para todas las pruebas realizadas los inmuebles obtenidos cumplieron con los parámetros, como era de esperarse.

4.4.9. Iteración 8: Paginador para el listado de resultados (recodificación)

Al realizar la búsqueda por localidad, se obtiene un listado con una gran cantidad de inmuebles a mostrar. Esto sobrecarga las capacidades de los dispositivos móviles por lo que se plantea realizar una separación de los resultados por página, donde cada página mostrará sólo 10 resultados.

- Planificación

Nro.	Fecha Inicio	Descripción
32.0	21/09/12	Diseño de la interfaz gráfica de los controles para realizar la paginación.
33.0	24/09/12	Integración gráfica del paginador.
34.0	25/09/12	Implementación del manejo de paginación.

- Diseño

Para implementar esta característica se debe reutilizar el mismo *query* de búsqueda utilizada en el buscador de inmuebles por localidad, con la diferencia de que se le agrega un límite y un *offset* a la misma, para obtener sólo 10 resultados por cada vez que se realice la consulta a base de datos.

- Codificación

Symfony provee un objeto que permite realizar la paginación de los resultados de un *query*, este objeto se llama "*pager*", el mismo en conjunto con *Propel* permite cumplir la necesidad planteada en esta iteración. A continuación, la imagen del código necesario para el manejo del paginador:

actions.class.php - Editor

actions.class.php

```
135   //Se definen los valores de la pagina del listado
136   if($request->getParameter('page')){
137     $page = $request->getParameter('page');
138   }else{
139     $page = 1;
140   }
141
142   //Se buscan los inmuebles
143   $c = InmuebleUsadosPeer::getInmueblesBuscardorAvanzado($this->formAvanzado
144   $pager = new sfPropelPager('InmuebleUsados', 10);
145   $pager->setCriteria($c);
146   $pager->setPage($page);
147   $pager->init();
148
149   $this->items = $pager;
150
```

Figura 4.23: Código del paginador para el listado de inmuebles

En la imagen se observa la instanciación del paginador en la línea 144, y en la línea 147 se ejecuta el *query* de búsqueda por medio del objeto paginador. Esto devuelve un arreglo de sólo 10 objetos que serán mostrados al usuario.

En la siguiente imagen se encuentra el código de la vista necesario para mostrar el paginador:

_pager.php - Editor

_pager.php

```
1  <div class="ui-body ui-body-c" style="border-left: none; border-right: none; paddin
2    <div data-role="controlgroup" data-type="horizontal" data-theme="<?php echo sfCon
3      <?php //Link para la primera pagina ?>
4      <?php if ($items->getFirstPage() != $items->getPage()) :?>
5        <?php echo link_to('Página 1', '@main?action=listado&tipo=3&page='.$items->ge
6      <?php endif; ?>
7
8      <?php //Links para las paginas ?>
9      <?php $links = $items->getLinks(); ?>
10     <?php foreach ($links as $page): ?>
11       <?php ($page == $items->getPage()) ? $theme = sfConfig::get('app_theme_button
12       <?php echo link_to($page, '@main?action=listado&tipo=3&page='.$page, array('d
13     <?php endforeach; ?>
14
15     <?php //Link para la ultima pagina ?>
16     <?php if ($items->getLastPage() != $items->getPage()) :?>
17       <?php echo link_to('Página '.$items->getLastPage(), '@main?action=listado&tip
18     <?php endif; ?>
19   </div>
20   <i><?php echo $items->getNbResults().' resultados' ?> - <?php echo 'Página '.$ite
21 </div>
22
```

Figura 4.24: Código del paginador (capa vista)

El objetivo de este código es mostrar botones de navegación al usuario que le permitan decidir la página de contenido que desea visualizar. Entre las líneas 5 y 17 se imprimen todos los botones de las páginas a mostrar. En este caso se le muestra al usuario un menú que le permite navegar hacia la primera y última página del listado, así como también las páginas siguientes a la que tenga seleccionada actualmente.

- Pruebas

Se realizó la comparación de varios resultados de búsqueda contra el resultado obtenido de aplicar el *query* SQL directamente en base de datos. Se comprobó que para cada página que se seleccione en la aplicación se presentan los inmuebles que se esperaba obtener desde la base de datos.

4.4.10. Iteración 9: Funcionalidades de redes sociales y enviar a un amigo para el detalle del inmueble

En el detalle del inmueble se debe agregar funcionalidades para compartir el mismo en redes sociales (*Facebook* y *Twitter*), también se agregan funcionalidades para enviar el inmueble vía correo electrónico a un amigo.

- Planificación

Nro.	Fecha Inicio	Descripción
35.0	26/09/12	Integración gráfica de los elementos para compartir el inmueble.
36.0	27/09/12	Implementación funcional de los links para redes sociales.
37.0	28/09/12	Implementación de funcionalidad de "Enviar a un amigo".

- Diseño

Redes sociales: para poder utilizar correctamente la funcionalidad de compartir en redes sociales se colocan enlaces con las características necesarias para que sea reconocido por la red en la que se desea publicar el contenido. Para que funcione más rápido la funcionalidad es necesario que el usuario tenga una sesión abierta de su red social, en caso contrario se solicita al usuario que inicie sesión.

Enviar a un amigo: el envió del inmueble se realiza a través del correo electrónico, proporcionando la dirección del correo destino y cualquier información adicional que se desee comunicar a la persona destino. Toda la información es manejada por un formulario HTML.

- Codificación

A continuación se muestra el código para la funcionalidad de compartir en **redes sociales**:

_social.php - Editor

_social.php

```
1  <table class="center" width="100%">
2    <tr>
3      <td>
4        <div style="margin: auto; width: 65px;">
5          <a data-rel="dialog" data-transition="pop" href="<?php echo url_for('@main?
6            <div class="social_icon_div">
7              <?php echo image_tag('icons/mail.png', array('alt' => '')) ?>
8
9              <br/>Enviar
10           </div>
11         </a>
12       </div>
13     </td>
14     <td>
15       <div style="margin: auto; width: 65px;">
16         <a target="_blank" href="http://www.facebook.com/sharer.php?u=<?php echo ur
17           <div class="social_icon_div">
18             <?php echo image_tag('icons/face.png', array('alt' => '')) ?>
19
20             <br/>Facebook
21           </div>
22         </a>
23       </div>
24     </td>
25     <td>
26       <div style="margin: auto; width: 65px;">
27         <a target="_blank" href="http://twitter.com/home?status=RT visita Inmobilia
28           <div class="social_icon_div">
29             <?php echo image_tag('icons/twitter.png', array('alt' => '')) ?>
30
31             <br/>Twitter
32           </div>
33         </a>
34       </div>
35     </td>
36   </tr>
37 </table>
```

Figura 4.25: Código para botones de redes sociales (capa vista)

Para esta funcionalidad sólo se requiere implementación en la capa de la vista, colocando enlaces con parámetros necesarios establecidos por cada red social como se puede ver en las líneas 14 a 35.

Para el caso de la funcionalidad de **enviar a un amigo**, se debe implementar un formulario, un controlador y una vista.

El formulario permite manipular la información ingresada por el usuario el cuál ingresará el valor clave de la funcionalidad, el correo electrónico destino. A continuación el código del formulario de envío:

MailDetalleForm.class.php - Editor

MailDetalleForm.class.php

```
10  class MailDetalleForm extends sfForm
11  {
12    public function configure()
13    {
14      //Widgets
15      $this->widgetSchema['nombre'] = new sfWidgetFormInputText();
16      $this->widgetSchema['mail']   = new sfWidgetFormInputText();
17      $this->widgetSchema['mail_destino'] = new sfWidgetFormInputText();
18      $this->widgetSchema['comentarios']  = new sfWidgetFormTextarea();
19
20      //Validators
21      $this->validatorSchema['nombre'] = new sfValidatorString(array('required' => tr
22      $this->validatorSchema['mail']   = new sfValidatorEmail(array('required' => tru
23      $this->validatorSchema['mail_destino'] = new sfValidatorEmail(array('required'
24      $this->validatorSchema['comentarios']  = new sfValidatorString(array('required'
25
26      //Labels
27      $this->widgetSchema->setLabels(array(
28        'nombre' => 'Su nombre <span class="texto_rojo">*</span>',
29        'mail'   => 'Su correo electrónico <span class="texto_rojo">*</span>',
30        'mail_destino' => 'Correo destino <span class="texto_rojo">*</span>',
31        'comentarios'  => 'Comentarios <span class="texto_rojo">*</span>'
32      ));
33
34      $this->widgetSchema->setNameFormat('mail[%s]');
35    }
36  }
```

Figura 4.26: Código de la clase del formulario (enviar a un amigo)

En esta porción de código se declaran el tipo de campos que tendrá el formulario, entre las líneas 15 y 18. También se declara el tipo de validación a realizar para cada uno de los anteriores, entre las líneas 21 y 24.

En el controlador se procesa y crea el contenido del correo electrónico a enviar a través del siguiente código:

actions.class.php - Editor

actions.class.php

```
63   public function executeDetalleMail(sfWebRequest $request)
64   {
65     //$this->setLayout('layout');
66     $this->enviado = false;
67     $this->form = new MailDetalleForm();
68
69     if($request->isMethod('post')){
70       $this->form->bind($request->getParameter($this->form->getName()), $request->g
71       if($this->form->isValid()){
72         $message = $this->getMailer()->compose();
73         $message->setSubject($this->form->getValue('nombre').' le ha enviado un inm
74         $message->setTo($this->form->getValue('mail_destino'));
75         $message->setFrom($this->form->getValue('mail'));
76
77         $html = $this->getPartial('mail/mailEnviarAmigo', array('nombre' => $this->
78         $message->setBody($html, 'text/html');
79         $this->getMailer()->send($message);
80
81         $this->enviado = true;
82       }
83     }
84
85   }
```

Figura 4.27: Código para procesamiento de formulario para “enviar a un amigo” (capa controlador)

En la imagen se valida que el formulario sea correcto (línea 71), y también se crea el correo electrónico a enviar con la información ingresada por el usuario (líneas 72 y 79).

- Pruebas

Las pruebas para las redes sociales se realizaron verificando que al seleccionar cualquier opción de red social, el contenido aparece publicado en una cuenta de prueba, con el enlace para acceder al inmueble mencionado. Esto se repitió con varios inmuebles y en todos los casos se obtuvo el resultado esperado.

Para la funcionalidad de enviar a un amigo, se enviaron varios inmuebles a distintos correos Gmail, Hotmail y se validó que llegara el correo con la información y el formato esperado. Se obtuvo el resultado esperado.

4.4.11. Iteración 10: Slider de imágenes y mapas geográficos en el detalle del inmueble (recodificación)

En esta iteración se desea recodificar el componente de imágenes del inmueble para agregar la posibilidad de visualizar las imágenes de mayor tamaño. También se plantea implementar la funcionalidad para ver un mapa geográfico, con la ubicación del inmueble en caso de que se haya provisto la ubicación del mismo.

- Planificación

Nro.	Fecha Inicio	Descripción
38.0	17/03/13	Investigación e implementación de *slider* de visualización de imágenes manteniendo compatibilidad con *JQuery Mobile.*
39.0	20/03/13	Investigación e implementación de mapa geográfico manteniendo compatibilidad con *JQuery Mobile.*

- Diseño

Slider de imágenes: La aplicación web Inmobilia.com, posee imágenes de los inmuebles con un tamaño máximo de 400px, cargar cada una de estas imágenes en un dispositivo móvil puede ser muy lento, dependiendo en la velocidad de conexión. Cada inmueble puede llegar a tener varias imágenes (máximo 12), por lo que para agilizar la carga de las mismas en el dispositivo móvil se desea implementar un *slider* que cargue cada imagen sólo si el usuario elige ver la misma (carga *on demand*).

Mapa geográfico: Para poder ofrecer una visualización geográfica de la ubicación del inmueble, se plantea utilizar mapas de Google, los cuáles se cargan en el momento que el usuario lo desee.

- Codificación

A continuación se muestra el código para la funcionalidad del *slider* de imágenes:

_imagenes_inmueble.php - Editor

_imagenes_inmueble.php

```
1  <?php $sources = inmobilia::getRutaTodasImagenesInmueble($inmueble); ?>
2  <?php $sourcesGrandes = inmobilia::getRutaTodasImagenesInmuebleGrande($inmueble); ?>
3
4  <?php if($sources): ?>
5    <link rel="stylesheet" type="text/css" media="screen" href="/css/photoswipe/photoswi
6    <script type="text/javascript" src="/js/photoswipe/klass.min.js"></script>
7    <script type="text/javascript" src="/js/photoswipe/photoswipe.min.js"></script>
8
9    <?php $hash = md5(date("H:i:s")); ?>
10   <script type="text/javascript">
11     //Inicializa la galeria
12     $(document).ready(function(){
13       var options = {captionAndToolbarAutoHideDelay: 99999};
14       $('#gallery<?php echo $hash; ?> a').photoSwipe(options);
15
16     });
17   </script>
18
19   <div style="overflow: auto;">
20     <div id="gallery<?php echo $hash; ?>" class="gallery-page" style="width: <?php ech
21       <?php $i = 0; ?>
22       <?php foreach($sources as $source): ?>
23         <?php $text = 'Imagen '.($i+1).' / '.count($sources); ?>
24         <a href="<?php echo '/'.$sourcesGrandes[$i]; ?>" rel="external">
25           <?php echo image_tag($source, array('class' => 'detalle_imagen', 'align' =>
26         </a>
27         <?php $i++; ?>
28       <?php endforeach; ?>
29     </div>
30   </div>
31 <?php endif; ?>
```

Figura 4.28: Código para cargar el *slider* de imágenes (capa vista)

Este *slider* es un componente adicional compatible con *JQuery Mobile*, el mismo se utiliza por medio de los archivos *klass.js* y *photoswipe.js* (ver líneas 6 y 7). El *slider* trabaja sobre la estructura del DOM HTML existente, identificando los objetos que serán manejados en el *slider*. En las líneas 22 a 28 se observa como se carga la estructura de las imágenes y sus enlaces necesarios para hacer el llamado al *slider*.

A continuación se muestra el código para la funcionalidad de **mapa geográfico:**

detalleGmapsSuccess.php - Editor

detalleGmapsSuccess.php

```
1  <script type="text/javascript" src="http://maps.google.com/maps/api/js?sensor=false"></
2
3  <div id="basic_map" data-role="page">
4    <div data-role="content">
5      <div class="ui-bar-c ui-corner-all ui-shadow" style="padding:1em;">
6        <div id="map">
7          <!-- Aqui en donde se va a desplegar el google map -->
8        </div>
9      </div>
10   </div>
11
12   <div data-role="footer">
13     <?php echo link_to('Volver', '@homepage', array('data-rel' => 'back', 'data-role'
14   </div>
15 </div>
16
17 <script type="text/javascript">
18   var map = null;
19   var latitud = <?php echo $inmueble->getLatitud(); ?>;
20   var longitud = <?php echo $inmueble->getLongitud(); ?>;
21
22   //Call this function when the page has been loaded
23   function initialize()
24   {
25     var mapOptions = {
26       center: new google.maps.LatLng(latitud, longitud),
27       zoom: 14,
28       mapTypeId: google.maps.MapTypeId.ROADMAP,
29       language: 'es',
30       scaleControl: true
31     };
32     map = new google.maps.Map(document.getElementById("map"), mapOptions);
33
34     var latLng = new google.maps.LatLng(latitud, longitud);
35     new google.maps.Marker({
36       position: latLng,
37       map: map,
38       title:" El punto Lat: " +  latitud  + " Long: " + longitud
```

Figura 4.29: Código para cargar el mapa geográfico (capa vista)

En este código se observa la lógica necesaria para presentar al usuario un mapa geográfico basado en la herramienta Google Maps. El mismo muestra al usuario la ubicación del inmueble que se está observando. En la línea 1 se realiza el llamado al API de Google Maps para poder utilizarla. En las líneas 6 a 8 se encuentra el contenedor donde se muestra el mapa y en las líneas 23 en adelante se encuentra la función "*initialize*" la cuál se encarga de instanciar el Google Maps y asignarle los parámetros necesarios para mostrar la ubicación del inmueble.

- Pruebas

Slider de imágenes: Para realizar las pruebas correspondientes se plantean los siguientes escenarios:

1. Las imágenes deben cargarse y el *slider* debe funcionar correctamente. Para probar este punto se verifica cualquier inmueble que tenga imágenes asociadas comprobando así, que en realidad se cargaron las imágenes del mismo. Luego se procede a probar cada botón del *slider* verificando también que los mismos realizaron la funcionalidad que le corresponda (pasar a siguiente imagen, imagen anterior e iniciar presentación).

2. Se debe validar que las imágenes cargadas pertenezcan al inmueble, es decir, que sólo se vean imágenes pertenecientes al inmueble. Es punto se comprueba directamente a nivel del directorio del proyecto. Cada inmueble posee un directorio particular en donde se almacenan sus imágenes. Para este escenario se validó que las imágenes que se muestran en el *slider* son las que se encuentran en el directorio perteneciente al inmueble.

Mapa geográfico: Para realizar las pruebas correspondientes se plantean los siguientes escenarios:

1. El mapa debe cargarse mostrando la zona indicada por el inmueble. Esto se validó utilizando Google maps, introduciendo las coordenadas (latitud y longitud), pertenecientes a una muestra de inmuebles y comprobando que cargue la zona geográfica correcta (país, estado, ciudad, calle/avenida.).

2. Sobre el mapa debe aparecer un punto que indique la ubicación exacta del inmueble. Igual al escenario anterior se comprueba con Google maps que el punto marcado esté en la zona correcta pero comprobando directamente en la base de datos que los valores de latitud y longitud pertenecen al inmueble que se está visualizando actualmente.

De acuerdo con esta explicación se puede decir que el *slider* y los mapas han pasado las pruebas correctamente.

4.4.12. Iteración 11: Implementación de manejo de sesión de usuario

En esta iteración se implementan las funcionalidades necesarias para manejar la sesión del usuario. Las funcionalidades a implementar son: registro de usuario, inicio de sesión, cerrar sesión y menú lateral de opciones.

- Planificación

Nro.	Fecha Inicio	Descripción
40.0	24/03/13	Cambio en navegación de la página inicial de la aplicación (*homepage*).
41.0	25/03/13	Desarrollo del registro de usuario.
42.0	27/03/13	Implementación de la creación y manejo de la sesión de usuario.
43.0	28/03/13	Desarrollo de inicio de sesión.
44.0	30/03/13	Codificación del "Menú lateral de opciones".

- Diseño

Dada la cantidad de funcionalidades en esta iteración se desea facilitar al usuario el manejo de las mismas, razón por la cuál se plantea realizar una página inicial que le permita al usuario elegir cómo va a utilizar la aplicación. Lo puede hacer escogiendo entre 3 opciones: continuar sin iniciar sesión, iniciando sesión o registrarse en el sistema (en caso de que lo requiera).

En el caso de **"Continuar sin sesión"** la aplicación permite al usuario utilizar la aplicación normalmente. Puede realizar búsquedas de inmuebles, visualizar la información de los mismos, entre otras funcionalidades.

En el caso de **"Iniciar sesión"** y **"Registrarse"** se maneja la información de los usuarios por medio de una nueva tabla en base de datos llamada "*mobile_usuarios*", esto para diferenciar entre los usuarios que se han registrado en la aplicación móvil y los que estén registrados en la aplicación web normal. También se debe implementar una clase de formulario para cada una, esto para cumplir con la estructura de desarrollo provista por Symfony. Ambos formulario al ser llenados con datos válidos tendrán el mismo resultado, el cuál es que el usuario inició sesión. Además se debe desarrollar el código necesario para procesar la carga y procesamiento del formulario.

Menú lateral: Este es un menú de acceso rápido a opciones varias, las cuáles pueden cambiar dependiendo de si el usuario ha iniciado sesión o no. Las opciones que puede tener son: iniciar sesión, cerrar sesión y opciones varias de inmuebles.

- Codificación

A continuación se puede ver la imagen del código necesario para manipular el **registro de usuarios**:

RegistrarseForm.class.php - Editor

RegistrarseForm.class.php

```
10  class RegistrarseForm extends sfForm
11  {
12    public function configure()
13    {
14      //Widget
15      $this->widgetSchema['id_usuario'] = new sfWidgetFormInputHidden();
16      $this->widgetSchema['nombre']     = new sfWidgetFormInputText(array(), array('class' =
17      $this->widgetSchema['correo']     = new sfWidgetFormInputText(array(), array('class' =
18      $this->widgetSchema['password']   = new sfWidgetFormInputPassword(array(), array('clas
19
20      //Validores
21      $this->validatorSchema['id_usuario'] = new sfValidatorString(array('required' => false
22      $this->validatorSchema['nombre']     = new sfValidatorString(array('required' => true)
23      $this->validatorSchema['correo']     = new sfValidatorEmail(array('required' => true),
24      $this->validatorSchema['password']   = new sfValidatorString(array('required' => true)
25
26      //Labels
27      $this->widgetSchema->setLabels(array(
28        'nombre'   => 'Nombre <span class="texto_rojo">*</span>',
29        'correo'   => 'Correo electrónico <span class="texto_rojo">*</span>',
30        'password' => 'Contraseña <span class="texto_rojo">*</span>'
31      ));
32
33      $this->widgetSchema->setNameFormat('registrarse[%s]');
34
35      //Valida que el correo sea unico
36      $this->validatorSchema->setPostValidator(
37        new sfValidatorCallback(array('callback' => array($this, 'verificaCorreoUnico')))
38      );
39    }
40
41    public function verificaCorreoUnico($validator, $values)
42    {
43      $usuario = MobileUsuariosPeer::existeUsuario($values['correo']);
44      if($usuario){
45        $error = new sfValidatorError($validator, 'El correo electrónico ya está registrado'
46        throw new sfValidatorErrorSchema($validator, array('Error' => $error));
47      }
```

Figura 4.30: Código de la clase del formulario de registro de usuario

En la figura se observa la declaración del formulario (siguiendo la estructura de Symfony), para el registro de usuarios en la aplicación. En la líneas 14 a 24 se definen los campos del formulario. En el caso de realizar un registro de usuario se debe verificar que el usuario no esté registrado aún, esto se realiza verificando que el correo electrónico introducido no exista, esta validación se puede observar en las líneas 41 a 47.

A continuación se presenta el código necesario del formulario para **iniciar sesión**:

LoginForm.class.php - Editor

LoginForm.class.php

```
10  class LoginForm extends sfForm
11  {
12    public function configure()
13    {
14      //Widget
15      $this->widgetSchema['correo']   = new sfWidgetFormInputText(array(), array('class' => '
16      $this->widgetSchema['password'] = new sfWidgetFormInputPassword(array(), array('class'
17
18      //Validores
19      $this->validatorSchema['correo']   = new sfValidatorEmail(array('required' => true), ar
20      $this->validatorSchema['password'] = new sfValidatorString(array('required' => true), a
21
22      //Labels
23      $this->widgetSchema->setLabels(array(
24        'correo'   => 'Correo electrónico <span class="texto_rojo">*</span>',
25        'password' => 'Contraseña <span class="texto_rojo">*</span>'
26      ));
27
28      $this->widgetSchema->setNameFormat('login[%s]');
29
30      //Validador del usuario
31      $this->validatorSchema->setPostValidator(
32        new sfValidatorCallback(array('callback' => array($this, 'verificaUsuario')))
33      );
34    }
35
36    public function verificaUsuario($validator, $values)
37    {
38      if(!empty($values['correo']) && !empty($values['password'])){
39        $usuario = MobileUsuariosPeer::existeUsuario($values['correo'], $values['password']);
40        if($usuario){
41          //Obtiene la sesion del usuario y le agrega informacion
42          $current_action = sfContext::getInstance()->getActionStack()->getLastEntry()->getAc
43          $user = $current_action->getUser();
44          $user->setAttribute('id_usuario', $usuario->getIdUsuario());
45          $user->setAttribute('nombre', $usuario->getNombre());
46          $user->setAttribute('email', $usuario->getCorreo());
47        }else{
```

Figura 4.31: Código de la clase del formulario para iniciar sesión

En la figura se observa la declaración del formulario (siguiendo la estructura de Symfony), para iniciar la sesión del usuario en la aplicación. En las líneas 14 a 20 se definen los campos del formulario. En el caso de intentar iniciar sesión se debe verificar que el usuario esté registrado, esto se realiza verificando que el correo electrónico introducido exista (línea 43). Si el usuario existe en la base de datos se crea la sesión del mismo (ver líneas 40 a 46).

A continuación se presenta el código necesario de la **página de inicio** de la aplicación:

indexSuccess.php - Editor

indexSuccess.php

```
13  <?php echo link_to('Continuar sin sesión', '@main?action=home', array('data-transition' =>
14
15 <div data-role='collapsible-set" data-content-theme="c' data-collapsed-icon='arrow-r' data-
16   <div class="ui-corner-bottom" data-role='collapsible' data-theme='<?php echo sfConfig::ge
17     <h3>Iniciar sesión</h3>
18
19     <?php echo $formLog->renderGlobalErrors(); ?>
20     <span class='texto_rojo">(*) Datos Requeridos</span><br/><br/>
21
22     <form id="login' action='<?php echo url_for('@main?action=iniciarSesionPost'); ?>' meth
23       <table>
24         <tr>
25           <td>
26             <?php echo $formLog['correo']->renderLabel() ?>
27             <?php echo $formLog['correo'] ?>
28             <?php echo $formLog['correo']->renderError() ?>
29           </td>
30         </tr>
31         <tr>
32           <td>
33             <?php echo $formLog['password']->renderLabel() ?>
34             <?php echo $formLog['password'] ?>
35             <?php echo $formLog['password']->renderError() ?>
36             <?php echo $formLog->renderHiddenFields(false) ?>
37           </td>
38         </tr>
39       </table>
40
41       <button name='enviar" type='submit" data-inline='true' data-theme='<?php echo sfConfi
42     </form>
43   </div>
44
45   <?php //Coloca el acordeon con el formulario de registro ?>
46   <div data-role="collapsible' data-theme='<?php echo sfConfig::get('app_theme'); ?>' <?php
47     <h3>Registrarse</h3>
48
49     <?php echo $formReg->renderGlobalErrors(); ?>
50     <span class='texto_rojo">(*) Datos Requeridos</span><br/><br/>
```

Figura 4.32: Código para la página de inicio (capa vista)

La página de inicio permite al usuario iniciar sesión o registrarse según desee. Para manejar esta funcionalidades en una sola página se distribuyen los formularios dentro de acordeones de *JQuery Mobile* (ver línea 16 y 46), en los mismos se cargan los formularios explicados anteriormente.

A continuación se presenta el código necesario para el **menú lateral** de la aplicación:

_menuFooter.php - Editor

_menuFooter.php

```
12 <div id="popupPanel" class="menu_lateral" data-role="popup" data-corners="false" data-theme
13   <ul data-role="listview" data-divider-theme="c" data-theme="a" data-ajax="false">
14     <?php //Menu para manejar la cuenta ?>
15     <li data-role="list-divider" style="padding-top: 20px;">
16       Mi cuenta
17       <?php if($sf_user->isAuthenticated()): ?>
18         <div style="height: 20px; overflow: hidden;"><span class="texto_azul" style="font-s
19       <?php endif; ?>
20     </li>
21
22     <?php if($sf_user->isAuthenticated()): ?>
23       <li data-icon="delete"><?php echo link_to('Cerrar sesión', '@main?action=logout', arr
24     <?php else: ?>
25       <li data-icon="check"><?php echo link_to('Iniciar sesion', '@main?action=index&aco=lo
26       <li data-icon="grid"><?php echo link_to('Registrarse', '@main?action=index&aco=reg',
27     <?php endif; ?>
28
29     <?php //Menu para manejar los inmuebles del usuario ?>
30     <li data-role="list-divider" style="padding-top: 20px;">Mis inmuebles</li>
31
32     <li data-icon="back"><?php echo link_to('Ultimos visitados', '@main?action=listado&tipo
33     <?php if($sf_user->isAuthenticated()): ?>
34       <li data-icon="star"><?php echo link_to('Favoritos', '@main?action=listado&tipo=5', a
35     <?php endif; ?>
36   </ul>
37 </div>
```

Figura 4.33: Código para la página de inicio (capa vista)

En este código se define el menú lateral como un *popup* basándose en los objetos de *JQuery Mobile* (utilizando la instrucción *data-role="popup"*), ver línea 1. Entre las líneas 13 a 36 se definen los botones a mostrar al usuario, los mismos varían si el usuario ha iniciado sesión. Los botones que aparecen sin estar autenticado son: Iniciar sesión, Registrarse y Últimos visitados. Estando autenticado aparecerán los botones: Cerrar sesión, Últimos visitados y Favoritos.

- Pruebas

Funcionalidad para registro de usuario: Para probar esta tarea se realiza el envío del formulario con datos nulos o vacíos comprobando que las validaciones del formulario funcionan correctamente. Cuando el formulario es válido se verifica directamente en la base de datos que se ha creado correctamente el registro correspondiente al nuevo usuario. También se comprobó que una vez registrado el usuario, este inició sesión automáticamente y se asignaron los valores de sesión pertinentes.

Inicio de sesión: Igual que en formulario anterior se verifica que funcione correctamente la validación del mismo, colocándole valores nulos o vacíos en los campos. Luego se valida que la sesión de usuario y sus atributos hayan sido creados utilizando las herramientas de desarrollo provistas por Symfony.

Interfaz gráfica de la página de inicio: En esta interfaz se comprobó con varios navegadores y varios dispositivos móviles que la interfaz luzca como se diseñó inicialmente y todos sus elementos gráficos funcionen correctamente (acordeones y validaciones).

Menú lateral: Este menú se debe validar a nivel de interfaz gráfica. Validando con varios navegadores y varios dispositivos móviles se pudo comprobar que la interfaz lateral luzca como se diseñó inicialmente y todos sus elementos gráficos funcionen correctamente (botones, transición y cierre de menú).

4.4.13. Iteración 12: Marcar inmueble como favorito en el detalle del inmueble (recodificación)

En esta iteración se desea agregar las funcionalidades de marcar inmueble como favorito y agregar comentario al inmueble.

- Planificación

Nro.	Fecha Inicio	Descripción
45.0	31/03/13	Implementación de marcado del inmueble.
46.0	03/04/13	Implementación de *dialog* para presentación de formulario de comentario.
47.0	04/04/13	Implementación de funcionalidad para agregar comentarios al inmueble.

- Diseño

Funcionalidad marcar favoritos: El usuario, una vez autenticado, tiene la opción de mantener un listado de inmuebles particular el cuál sea de su interés. Esto se logra, marcando cada inmueble como favorito según sea su criterio. Para poder hacer esto se habilita un botón en el detalle del inmueble el cuál (utilizando Ajax), crea un registro en la base de datos asociando el inmueble con el usuario, permitiéndole acceder fácilmente a los mismos.

Funcionalidad agregar comentario: Con esta tarea se desea crear un formulario el cuál permita al usuario plasmar su opinión sobre un inmueble conocido. Para crear la sensación de que agregar un comentario al inmueble es una tarea simple, se crean unas ventanas denominadas "*dialog*" en *JQuery Mobile* para mostrar el formulario al usuario sin necesidad de abandonar el inmueble que está visualizando. Una vez se haya completado la información del comentario el mismo aparecerá en el inmueble como publicado.

- Codificación

A continuación se puede ver la imagen del código necesario para marca un inmueble como **favorito:**

actions.class.php - Editor

actions.class.php

```
247   //Marca un inmueble como favorito
248   public function executeDetalleMarcarFavorito(sfWebRequest $request)
249   {
250     $idUsuario = $this->getUser()->getAttribute('id_usuario');
251     $rel = new MobileFavoritos();
252     $rel->setIdUsuario($idUsuario);
253     $rel->setIdInmueble($request->getParameter('id_inmueble'));
254
255     if(MobileFavoritosPeer::existeRelacion($idUsuario, $request->getParameter('i
256       //Elimina la relacion
257       $rel->delete();
258     }else{
259       //Crea la relacion
260       $rel->save();
261     }
262     exit;
263   }
```

Figura 4.34: Código para marcar un inmueble como favorito (capa controlador)

La funcionalidad de **marcar como favorito** realiza una petición al servidor utilizando AJAX. En esta petición se llama a la función que se muestra en la imagen, la misma crea un registro en la base de datos para asociar al usuario con el inmueble y de esta manera almacenar la información. En la línea 255 se verifica si ya se encuentra asociado el inmueble con el usuario, en caso de que la relación exista la misma es eliminada (ver línea 257) y en caso de que la relación no exista la misma se crea (ver línea 260).

A continuación se puede ver la imagen del código necesario para **agregar un comentario** al inmueble:

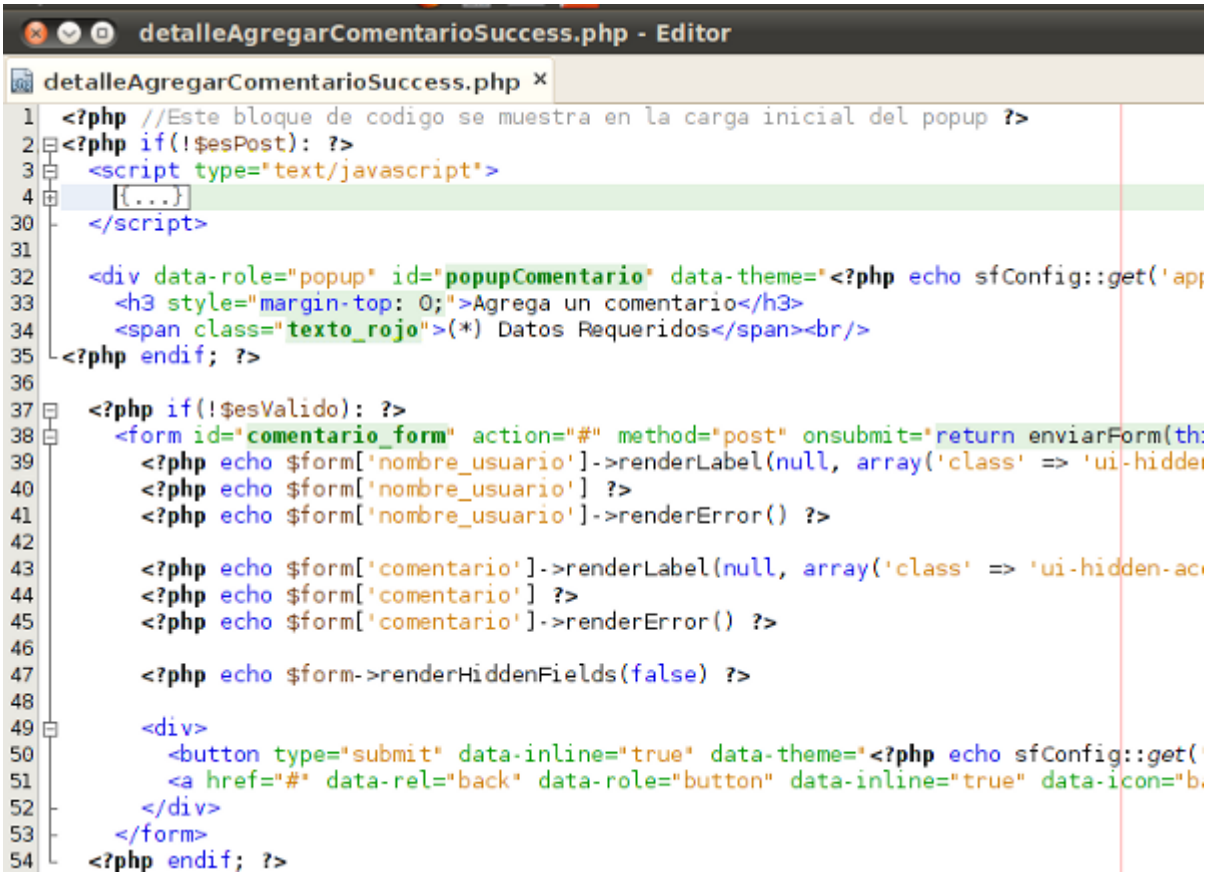


Figura 4.35: Código para agregar un comentario al inmueble (capa vista)

Con este código se muestra al usuario una ventana del tipo "*dialog*" (ver línea 32), donde se encuentra un formulario con datos para agregar el comentario a un inmueble. En las líneas 39 a 53 se carga el formulario y por medio de Javascript se maneja el guardado y validación de los datos en el mismo.

- Pruebas

Para el caso de **marcar como favorito** se prueban los siguientes componentes:

1. **El botón para marcar el inmueble:** Se verificó su funcionamiento con distintos navegadores y dispositivos móviles, validando con la base de datos que se haya creado el registro correspondiente y que el botón cambie de aspecto para diferenciar que el inmueble está marcado.
2. **El listado de inmuebles favoritos:** Para validar este listado se ingresó a la sección correspondiente y se verifico que los inmuebles listados sean los mismos que están asociados al usuario en la base de datos.
3. **Para el caso de agregar un comentario:** Se realizó la prueba de la funcionalidad con diferentes navegadores y dispositivos móviles, validando que se cree correctamente el registro correspondiente en la base de datos y que el comentario recién creado pueda ser visualizado en el inmueble.

Al realizar estas pruebas se puede decir que las funcionalidades correspondientes a la iteración trabajaron correctamente según el comportamiento esperado.

4.4.14. Iteración 13: Cambios en funcionalidad "Enviar a un amigo" en el detalle del inmueble (recodificación)

A continuación se describen las actividades necesarias para realizar cambios en la funcionalidad existente de "Enviar a un amigo", para realizar el manejo del mismo utilizando ventanas "*Dialog*" provistas por *JQuery Mobile*.

- Planificación

Nro.	Fecha Inicio	Descripción
44.0	07/04/13	Adaptación de funcionalidad con el cambio a "*Dialogs*".

- Diseño

Se desea realizar el cambio a "*Dialog*" para unificar las interfaces de manejo de formulario en el detalle del inmueble para brindar la sensación al usuario de que no ha abandonado el inmueble en el que se encuentra. Este cambio de interfaz requiere pocos cambios a nivel funcional. Se mantendrá el envío de correo con la información del formulario y sólo varía la interfaz gráfica de la funcionalidad para manejar la manipulación del formulario.

- Codificación

A continuación se puede ver la imagen del código necesario para procesar la solicitud de **enviar a un amigo**:

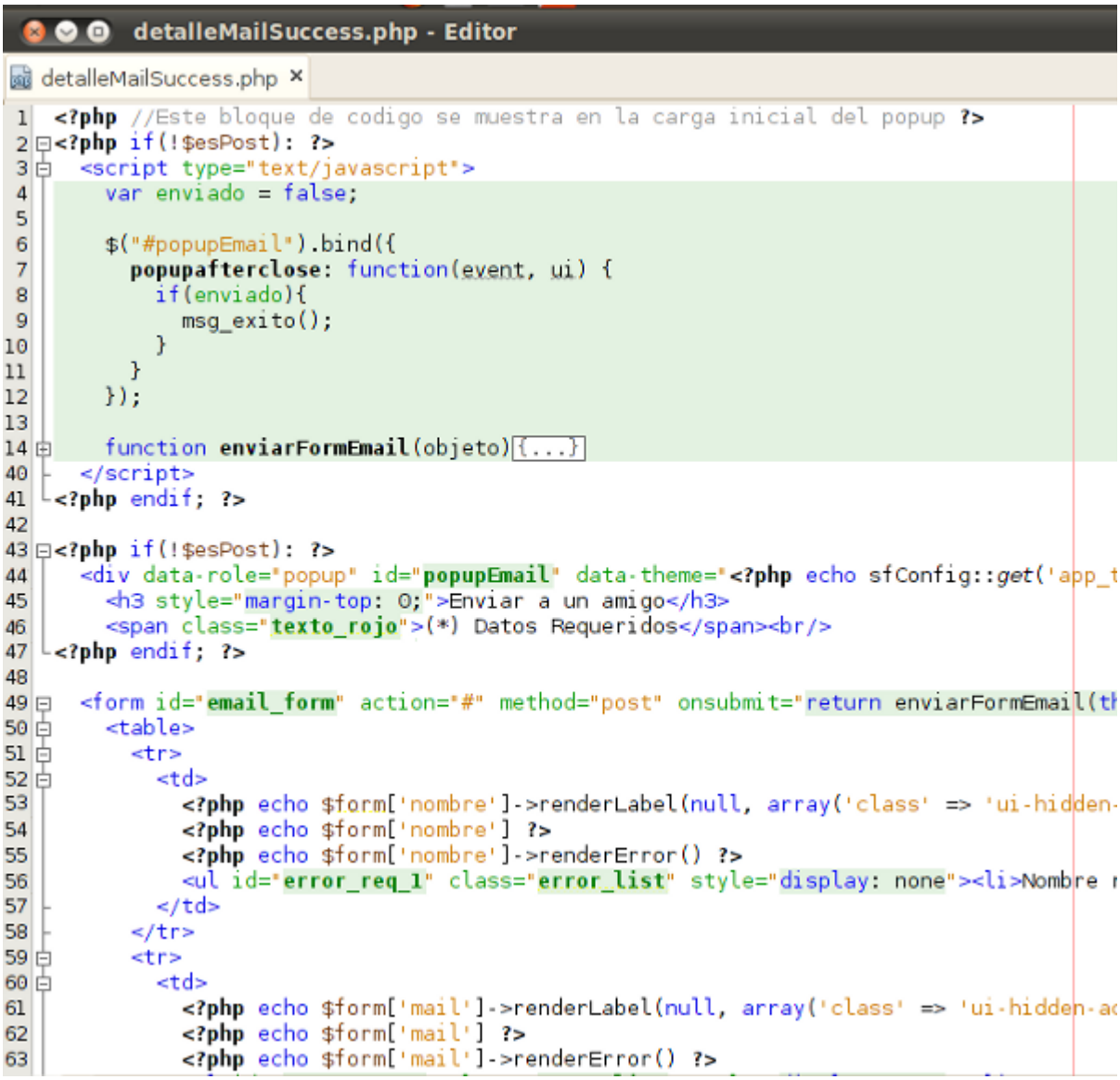


Figura 4.36: Código para procesar la función de enviar a un amigo (capa controlador)

Este código muestra al usuario una ventana de tipo "*Dialog*" (ver línea 44), la cuál contiene un formulario con los datos necesarios para enviar el inmueble a un amigo por medio de correo electrónico. En las líneas 3 a 14 se encuentra el código Javascript encargado de mostrar errores de validación en el formulario y también muestra el comentario recién creado al momento de cerrar la ventana de formulario.

- Pruebas

Para comprobar el correcto funcionamiento de esta tarea se realizaron pruebas de la misma con diferentes navegadores y diferentes dispositivos móviles validando que se enviara correctamente el correo electrónico con la referencia al inmueble y validando el funcionamiento de la navegación del *dialog*.

4.4.15. Iteración 14: Herramienta de prueba y correcciones del sistema

Para realizar una prueba final de la aplicación completa, se realiza una encuesta utilizando las "Encuestas de Google". Esta herramienta permite realizar un cuestionario a través de Internet con preguntas de cualquier tipo, la encuesta permite el acceso a la misma para poder recopilar la opinión del público sobre algún tema en específico.

La encuesta está orientada a la prueba de las funcionalidades de la aplicación y cómo se comporta la misma en diferentes dispositivos móviles, permitiendo al usuario dar su opinión según lo observado al utilizar la aplicación en su dispositivo móvil.

La encuesta está compuesta por 14 preguntas de selección simple donde se realiza la prueba de las funcionalidades de la aplicación y una pregunta de desarrollo en donde el usuario explica su experiencia al utilizar la aplicación. A partir de las respuestas del público se puede determinar el éxito de la aplicación.

Resultados de la encuesta

Al cumplirse un periodo de tiempo de prueba por parte de los usuarios se recopilan los datos obtenidos de 21 usuarios y se analiza sus respuestas. Los mismos se muestran a continuación:

Los dispositivos utilizados por los usuarios fueron: iPhone 4s, iPhone 4, Motorola Atrix 4G, Samsung, iPod, BlackBerry Pearl 9100, Samsung Galaxy S2, Motorola Blur MB525, BlackBerry 9320, iPad3, BlackBerry Bold 9900, BlackBerry Curve, Nokia, Samsung Galaxy Ace GT-S5830L.

1) Pregunta: ¿Pudo registrarse correctamente?

A continuación se observa la gráfica de resultados para esta pregunta:

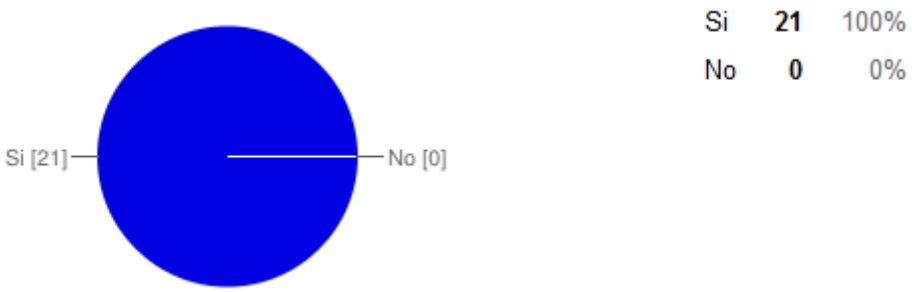


Figura 4.37: Representación gráfica de las respuestas (pregunta 1)

A través de los datos obtenidos se puede decir que el 100 % de los usuarios que intentaron registrarse al sistema lo completaron con éxito.

2) Pregunta: ¿Le ha parecido fácil el proceso de registro?

A continuación se observa la gráfica de resultados para esta pregunta:

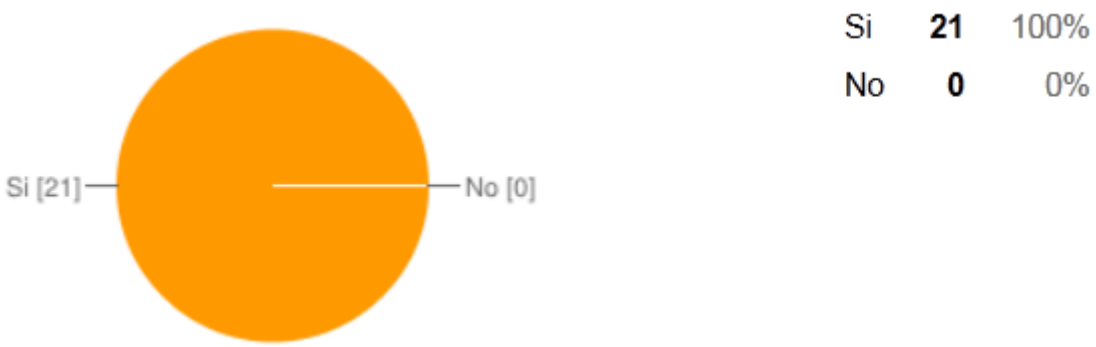


Figura 4.38: Representación gráfica de las respuestas (pregunta 2)

Según la respuesta de los usuarios se puede decir que el proceso de registro de usuarios es fácil de utilizar.

3) Pregunta: ¿Pudo realizar la búsqueda correctamente?

A continuación se observa la gráfica de resultados para esta pregunta:

Figura 4.39: Representación gráfica de las respuestas (pregunta 3)

Para esta funcionalidad se determina que el 100 % de los usuarios han podido utilizar correctamente el buscador de inmuebles.

4) Pregunta: ¿Los inmuebles mostrados en el resultado de búsqueda cumplen con los parámetros seleccionados?

A continuación se observa la gráfica de resultados para esta pregunta:

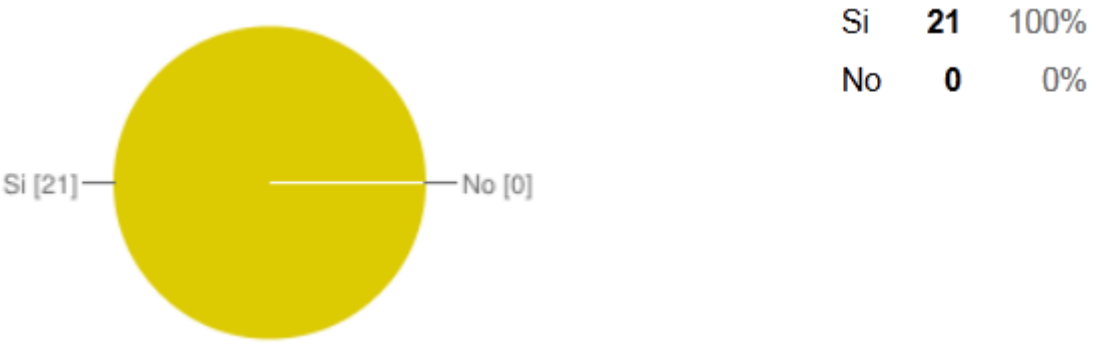


Figura 4.40: Representación gráfica de las respuestas (pregunta 4)

Según las respuestas de los usuarios, se puede decir que el buscador de inmuebles maneja correctamente los parámetros al realizar una búsqueda, devolviendo un listado con sólo los inmuebles que cumplan con los parámetros seleccionados por el usuario.

5) Pregunta: ¿El proceso para realizar una búsqueda le ha parecido?

A continuación se observa la gráfica de resultados para esta pregunta:

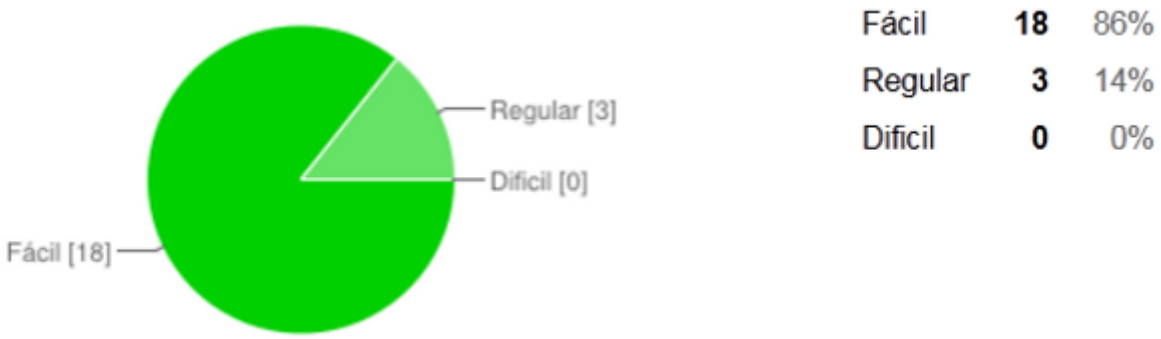


Figura 4.41: Representación gráfica de las respuestas (pregunta 5)

Dadas las respuestas de los usuarios se puede decir que, en su mayoría (86 %), consideran sencillo realizar una búsqueda en la aplicación.

6) Pregunta: Según su criterio ¿Cómo califica usted la disposición de la información en la sección?

A continuación se observa la gráfica de resultados para esta pregunta:

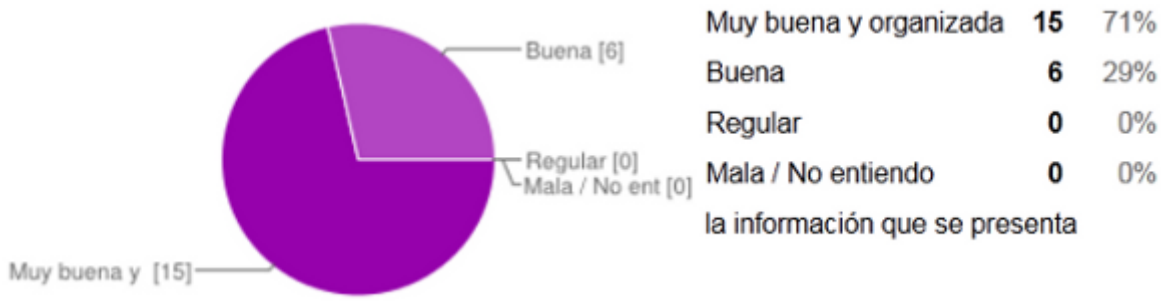


Figura 4.42: Representación gráfica de las respuestas (pregunta 6)

Los usuarios en su totalidad han considerado como “buena” y “muy buena” la disposición de la información del inmueble por lo que se considera que la presentación de la misma se entiende claramente.

7) Pregunta: ¿Cómo califica usted la visualización de las imágenes del inmueble?

A continuación se observa la gráfica de resultados para esta pregunta:

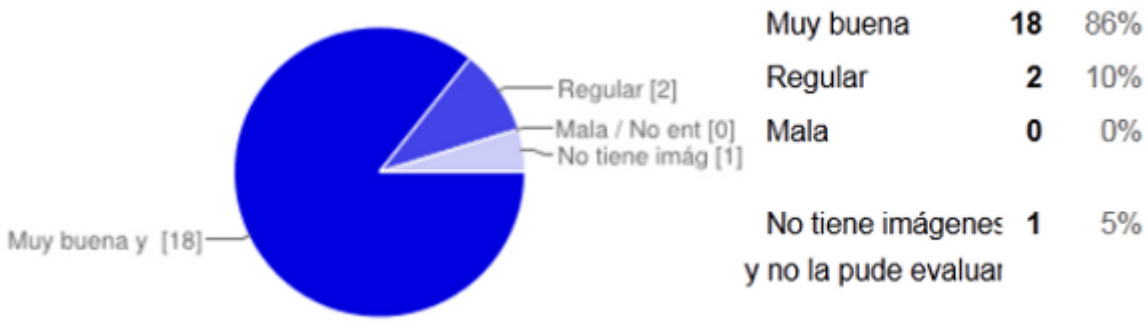


Figura 4.43: Representación gráfica de las respuestas (pregunta 7)

En su mayoría los usuarios indicaron que la visualización de las imágenes es muy buena y organizada.

8) Pregunta: Envíe el inmueble a un amigo utilizando la funcionalidad de "Enviar a un amigo". ¿Pudo enviar el inmueble correctamente?

A continuación se observa la gráfica de resultados para esta pregunta:

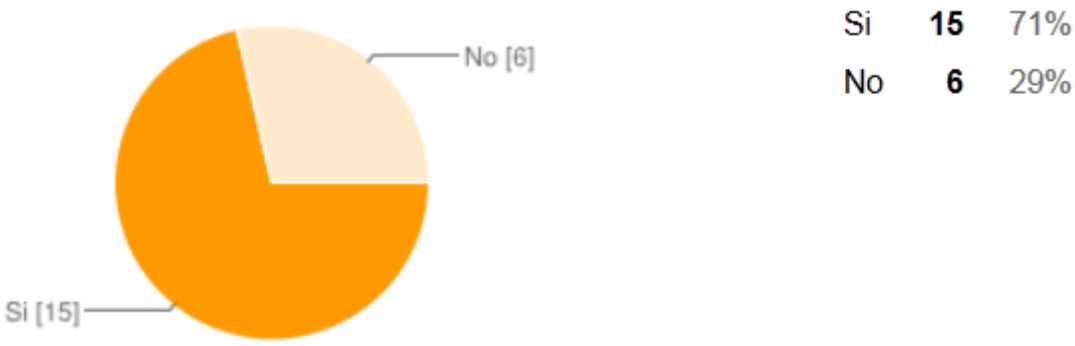


Figura 4.44: Representación gráfica de las respuestas (pregunta 8)

De acuerdo a los datos obtenidos, la funcionalidad de "Enviar a un amigo", funciono correctamente en el 71 % de los casos. Debido al alto porcentaje de usuarios que indicaron que no funcionó correctamente se realizó una entrevista personal para obtener información para determinar las causas de la falla. Se determinó que en ninguno de los casos se debió a algún error de la aplicación, sucedió que los correos llegaron a bandejas de spam (por colocar la misma dirección de remitente y destinatario), o fueron enviados a direcciones de correo electrónico inexistentes.

9) Pregunta: Comparta un inmueble en alguna de las redes sociales (Facebook, Twitter), ¿Pudo compartir el contenido correctamente?

A continuación se observa la gráfica de resultados para esta pregunta:

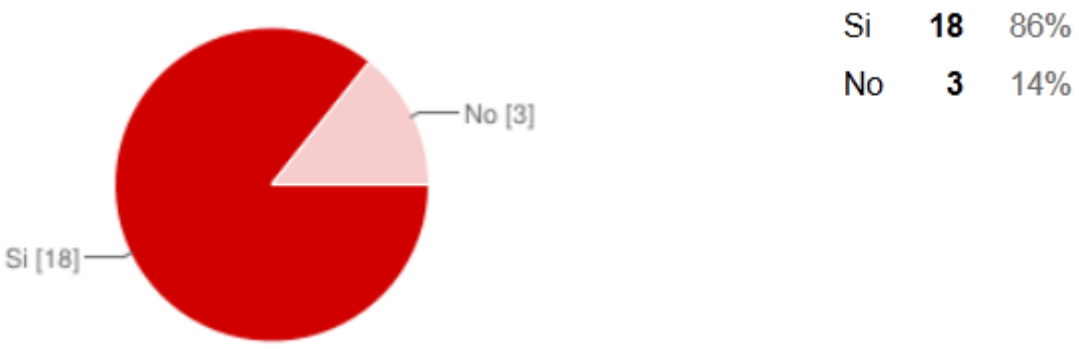


Figura 4.45: Representación gráfica de las respuestas (pregunta 9)

Según los datos obtenidos, la mayoría de los usuarios pudo compartir el inmueble en alguna red social. Por lo que se puede decir que la funcionalidad trabaja correctamente.

10) Pregunta: Marque / Desmarque el inmueble como favorito. ¿Pudo marcar / desmarcar el mismo?

A continuación se observa la gráfica de resultados para esta pregunta:

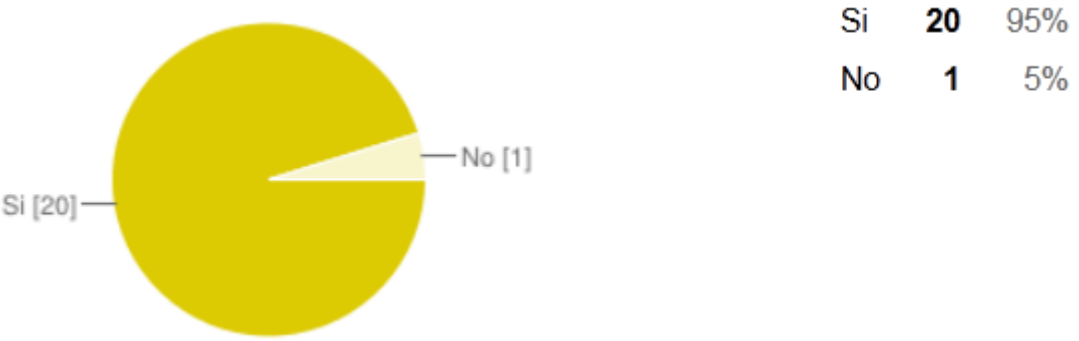


Figura 4.46: Representación gráfica de las respuestas (pregunta 10)

Según las respuestas obtenidas casi el 100 % de los usuarios pudo marcar y desmarcar el inmueble correctamente, por lo que se puede decir que la funcionalidad trabaja correctamente.

11) Pregunta: Realice un comentario en el inmueble. ¿Cómo calificaría el manejo de los comentarios?

A continuación se observa la gráfica de resultados para esta pregunta:

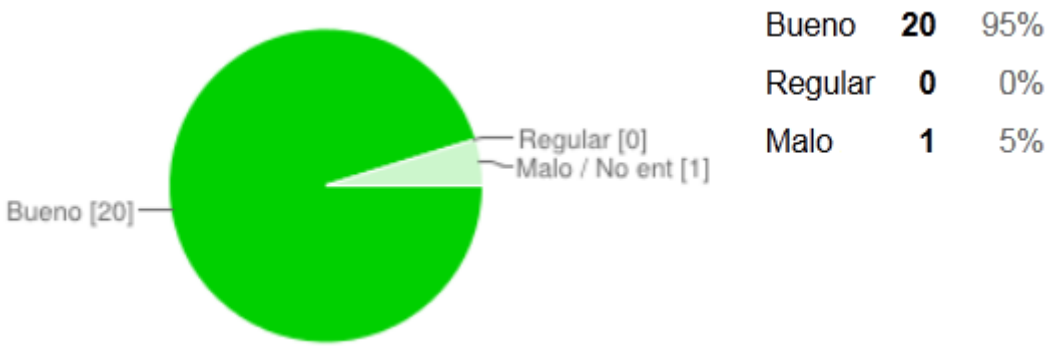


Figura 4.47: Representación gráfica de las respuestas (pregunta 11)

Con las respuestas obtenidas por el usuario se observa que casi el 100 % de los mismos pudieron utilizar correctamente los comentarios de los inmuebles, entonces se concluye que el manejo de los mismos se entiende claramente por los usuarios.

12) A lo largo de la aplicación. ¿Qué tal le ha parecido la disposición gráfica de los elementos?

A continuación se observa la gráfica de resultados para esta pregunta:

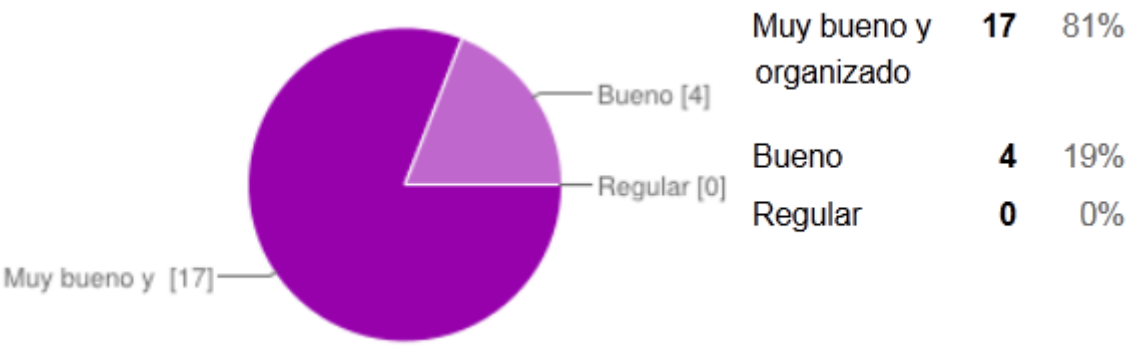


Figura 4.48: Representación gráfica de las respuestas (pregunta 12)

En general se puede decir que los elementos gráfico utilizados en la aplicación fueron de agrado para la mayoría de los usuarios.

13) ¿Qué tal le pareció el manejo de la aplicación?

A continuación se observa la gráfica de resultados para esta pregunta:

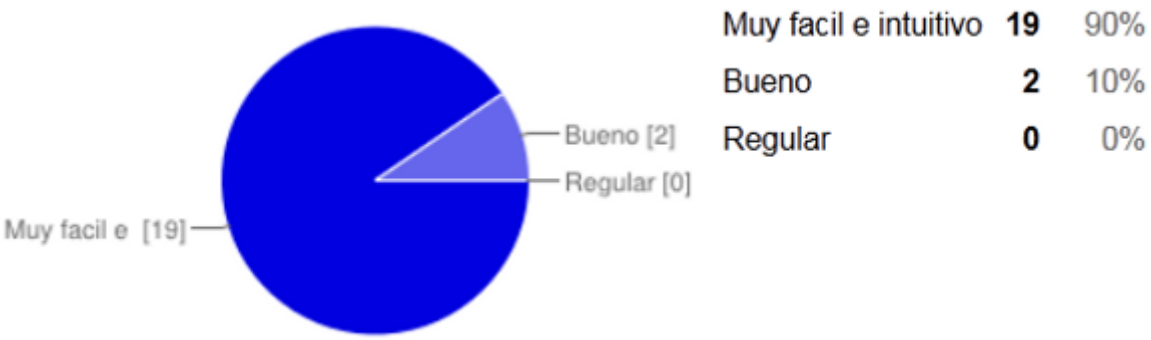


Figura 4.49: Representación gráfica de las respuestas (pregunta 13)

Según las respuestas de los usuarios se puede decir que la aplicación pudo ser utilizada correctamente por los usuarios y su manejo es intuitivo, lo que facilita la manipulación de la misma.

14) Observaciones y comentarios

Esta sección en la encuesta ha sido colocada para que el usuario exprese su opinión general sobre la aplicación y además para que pueda indicar cualquier tipo de error, sugerencia o duda que tenga.

Los comentarios obtenidos en esta parte en su mayoría fueron de aprobación hacia la aplicación y sólo algunos casos se encontraron sugerencias y mejoras menores, como es el caso de: colocar los íconos del listado de inmueble un poco más separados para que se entiendan mejor y colocar algún indicador que le haga entender al usuario que está autenticado en la aplicación.

Todos los comentarios realizados por los usuarios fueron evaluados y verificados para determinar si se debían realizar correcciones. En los casos que aplicaba, se realizaron las correcciones necesarias para corregir la situación.

La realización de esta encuesta ha permitido validar la aplicación en distintos dis-positivos móviles, abarcando los tres principales sistemas operativos móviles del mercado: BlackBerry OS, Android y iOS.

En este capítulo se ha explicado cómo se ha podido adaptar el método XP para el desarrollo de la aplicación web móvil *InmobiliaMobile*. Se define la metáfora de la misma, se levantan requerimientos y se realiza la división en iteraciones de las funcionalidades de la aplicación, permitiendo así la organización de las actividades a realizar al momento de implementar el software.

Parte IV

Conclusiones y recomendaciones

Conclusiones

La investigación en el marco teórico y el desarrollo realizado en este Trabajo Especial de Grado dieron como resultado la aplicación llamada *InmobiliaMobile*, la cuál es una aplicación web móvil para los usuarios de Inmobilia.com. La aplicación tiene acceso a la información de los inmuebles que se encuentran en la base de datos de Inmobilia.com utilizando dispositivos móviles (celulares, tabletas, iPods), además la aplicación es independiente de la aplicación web Inmobilia.com.

El logro de los objetivos propuestos se debe a las bases del marco teórico (estudio de las tecnologías móviles, dispositivos y tópicos importantes para el desarrollo móvil), y a la adaptación de la Programación Extrema (XP). Siguiendo las prácticas propuestas por XP (pequeñas versiones, diseños simples, planificación), se logró una comprensión clara de los requerimientos y rapidez en el desarrollo del software, manteniendo un alto nivel de calidad del mismo.

Para desarrollar una aplicación para dispositivos móviles se deben tener en cuenta varios aspectos adicionales a los que se tuviera para un desarrollo web típico. Se debe investigar sobre tipos de dispositivos, resoluciones de pantalla, sistemas operativos existentes en el mercado, sistemas operativos más usados por el público, tecnologías de desarrollo para móviles, recomendaciones de otros desarrolladores en cuanto a tecnologías y las buenas prácticas para una aplicación web móvil, recopilando así una base de conocimiento sobre la cuál se pueda planear y diseñar la aplicación a realizar.

Durante el proceso de selección de las herramientas a utilizar se evaluaron varias tecnologías las cuáles fueron rechazadas por varias razones. Una de las principales tecnologías evaluadas fue PhoneGap, la cuál utilizando HTML5, CSS y JS permite crear aplicaciones nativas para celulares y tablets. En la aplicación *InmobiliaMobile* se espera que continuamente se agreguen nuevas funcionalidades según lo solicite el cliente por lo que implementar la aplicación como una aplicación nativa exigiría a los usuarios actualizar a menudo la misma, lo cuál podría ser complicado. Además, se tiene como idea principal que la aplicación debe ser multiplataforma, es decir, que la aplicación pueda ser utilizada correctamente en varios sistemas operativos (por ejemplo: Android, iOS, Blackberry OS entre otros), de lo contrario se tendrían que desarrollar y mantener varias aplicaciones nativas distintas (una por cada sistema operativo). Según esta evaluación se identifica que implementar la aplicación *InmobiliaMobile* como una aplicación web para móviles es ideal para permitir a los desarrolladores agregar funcionalidades y realizar mantenimientos en la misma sin afectar a los usuarios. Por esta razón se selecciona la herramienta *JQuery Mobile* dado que está orientada al desarrollo de aplicaciones web para dispositivos móviles.

En el desarrollo se combinaron herramientas de desarrollo web como por ejemplo, el

lenguaje interpretado PHP sobre el *framework* Symfony y el *framework* de desarrollo para móviles *JQuery Mobile*, esto condujo a obtener una serie de experiencias y nuevas visiones en la programación de aplicaciones. Las tecnologías mencionadas pudieron trabajar en conjunto muy fácilmente una vez que se comprendió como trabajan correctamente cada una de las mismas. Siguiendo su estructura y combinándolas se pudo implementar código que requiere ejecutar la menor cantidad de instrucciones necesarias en cada petición, para poder realizar respuestas rápidas al dispositivo móvil. La estructura de *JQuery Mobile* permite cargar en el cliente una cantidad mucho menor de código desde el servidor, dado que trabaja utilizando peticiones asíncronas (AJAX), la respuesta obtenida es incrustada en el DOM ya existente en el cliente y se realiza una transición hacia la nueva porción de código cargada. Esto mejora el cambio gráfico entre una página y otra simulando el funcionamiento de una aplicación nativa y mejorando ampliamente los tiempos de respuesta de la aplicación.

Otro beneficio de estructura de *JQuery Mobile* es la reducción en la cantidad de archivos (js, css, otros), que se deben cargar para ejecutar una página. Cada archivo sólo será cargado la primera vez que se entre a la aplicación mejorando la velocidad de carga de la aplicación en general.

Es muy importante destacar la capacidad de expansión de la aplicación a nivel de desarrollo, ya que su diseño y estructura de software está basada en el modelo de software MVC el cuál permite la adaptación a los cambios. La incrementabilidad se garantiza porque el sistema permite agregar más código y estructuras sin que se vean afectadas las funciones y estructuras ya existentes en gran medida. Por ejemplo se puede agregar una nueva funcionalidad al sistema simplemente creando una nueva acción en el controlador y un nuevo *template* con el código HTML a mostrar, sin tener que tocar alguna funcionalidad ya existente.

La realización de este Trabajo Especial de Grado ha aportado, a mi persona, varios conocimientos importantes acerca del desarrollo para móviles, debido a que se tuvo que investigar y evaluar varias tecnologías. De estas tecnologías se debía elegir la que se adapte mejor a la aplicación considerando aspectos como: cantidad de dispositivos que soporta, tiempo de aprendizaje requerido y la compatibilidad con las tecnologías web. Además el haber seguido las prácticas de XP durante la implementación marcó una diferencia en el producto final desarrollado, dado que se siguió un camino para realizar las actividades logrando detectar y clasificar claramente los requerimientos, cuál es la importancia de cada uno, la prioridad y el orden de ejecución de los mismos.

Con respecto a la adaptación XP realizada para este trabajo, se logró aplicar la mayoría de las prácticas, sin embargo no todas pudieron adaptarse. Por ejemplo la programación en parejas para este caso no aplica debido a que el grupo está conformado por sólo una persona. Tampoco se requirió de la presencia constante del cliente ya que los requerimientos de la aplicación (por estar basados en una aplicación existente), estaban claros y bien definidos. Debido a esta ventaja se pudo agilizar el proceso de clasificación de prioridades de cada actividad. Entre las prácticas utilizadas se encuentran:

- La metáfora: La cuál permitió tener una visión general de la aplicación que se desea realizar.
- Pequeñas versiones: Todos los requerimientos contemplados para la aplicación fueron divididos en iteraciones cortas las cuáles permiten concentrar los esfuerzos en un grupo determinado de funcionalidades.
- El diseño: Con la premisa de "siempre realizar diseños simples", se pudo limitar la cantidad de código a implementar para que cumpla con la funcionalidad que se está realizando, logrando así que no se extienda el tiempo de desarrollo de manera innecesaria.
- El desarrollo: Al seguir las prácticas de XP durante el desarrollo (recodificación y estándares de codificación), se tuvo la confianza de implementar el código necesario para cada iteración sabiendo que, en caso de ser requerido, se puede realizar una recodificación en el área determinada sin que esto signifique un costo elevado de tiempo.
- Las pruebas: Realizando las pruebas del software según lo estipulo XP, permitió realizar una aplicación robusta y de calidad.

Con la nueva aplicación móvil de Inmobilia.com se obtiene como beneficio el acceso cómodo a la información de los inmuebles desde una gran variedad de dispositivos móviles, aumentando así los puntos de acceso al sistema. Adicionalmente se resume la interfaz gráfica de la aplicación, excluyendo elementos que no son tan relevantes para el usuario (publicidad, opciones de navegación, textos largos), lo cuál permite mostrar al usuario la información de los inmuebles organizada y en un espacio físico reducido.

Para probar el funcionamiento de la aplicación *InmobiliaMobile* se realizó una encuesta donde participó un grupo de usuarios, los cuáles dieron su opinión acerca de las funcionalidades principales del sistema. Los resultados a partir de esta encuesta fueron satisfactorios, los usuarios en su mayoría concuerdan en que la aplicación fue fácil de usar, presentaba la información de forma clara y que cumplía con la funcionalidad que se estaba evaluando. Con esta información se puede decir que se ha cumplido el objetivo general de este Trabajo Especial de Grado.

Recomendaciones

InmobiliaMobile es una aplicación web móvil creada como punto inicial para estudiar (por parte de Inmobilia), los beneficios que se puedan obtener al tomar en cuenta al número de usuarios que navega en la aplicación por medio de dispositivos móviles. *InmobiliaMobile* tiene sólo un subconjunto de las funcionalidades principales de Inmobilia.com, por lo que para mejorar la aplicación existente en el futuro se plantean los siguientes puntos:

1. Expansión del manejo de países, para soportar todos los existentes en Inmobilia.com (actualmente son 16 países). Esto permitirá que la aplicación pueda mostrar los inmuebles de cada país.

2. Integrar el proyecto *InmobiliaMobile* con su proyecto padre. Actualmente la aplicación trabaja con imágenes que no están actualizadas dado que las aplicaciones se encuentran en proyectos separados. Lo ideal sería que pudieran compartir los archivos para que al agregar un inmueble en Inmobilia.com las imágenes del mismo puedan ser visualizadas en *InmobiliaMobile* también.

3. Permitir al usuario establecer contacto directo con el usuario a través de la aplicación móvil. Esta funcionalidad no está contemplada actualmente debido a que la aplicación trabaja con datos reales de los contactos de los inmuebles, entonces podría generar problemas al intentar contactar a los mismos utilizando esta aplicación en desarrollo.

Parte V

Anexos

Encuesta de evaluacion funcional de la aplicación Inmobilia Mobile

*Obligatorio

Indique el modelo de su dispositivo móvil: *

Pruebas en home:

Ingrese con su dispositivo móvil a la url: http://inmobiliamobile.dyndns.org

Regístrese en el sistema (Al registrarse iniciara sesión automáticamente). Responda lo siguiente:

1) ¿Pudo registrarse correctamente? *

- Si
- No

2) ¿Le ha parecido fácil el proceso de registro? *

- Si
- No

Pruebas en el buscador por localidad:

Ingrese a la sección de búsqueda de inmuebles, presionando el botón "Buscar". Realice una búsqueda por localidad. Responda lo siguiente:

3) ¿Pudo realizar la búsqueda correctamente? *

- Si
- No

4) ¿Los inmuebles mostrados en el resultado de búsqueda cumplen con los parámetros seleccionados? *

- Si
- No

5) ¿El proceso para realizar una búsqueda le ha parecido: *

- Fácil
- Regular
- Dificil

Figura 4.50: Formulario: Encuesta de pruebas de aceptación (parte: 1 de 3)

Pruebas en el detalle del inmueble

Ingrese a la información detallada de cualquier inmueble. Responda lo siguiente:

6) Según su criterio ¿Cómo califica usted la disposición de la información en la sección? *

- Muy buena y organizada
- Buena
- Regular
- Mala / No entiendo la información que se presenta

7) ¿Cómo califica usted la visualización de las imágenes del inmueble? *

- Muy buena y organizada
- Regular
- Mala / No entiendo cómo funciona la visualización de imágenes
- No tiene imágenes y no la pude evaluar

8) Envíe el inmueble a un amigo utilizando la funcionalidad de "Enviar a un amigo". ¿Pudo enviar el inmueble correctamente? *

- Si
- No

9) Comparta un inmueble en alguna de las redes sociales (Facebook, Twitter), ¿Pudo compartir el contenido correctamente? *

Presione el boton con un icono de la red social de su preferencia

- Si
- No

10) Marque / Desmarque el inmueble como favorito. ¿Pudo marcar / desmarcar el mismo? *

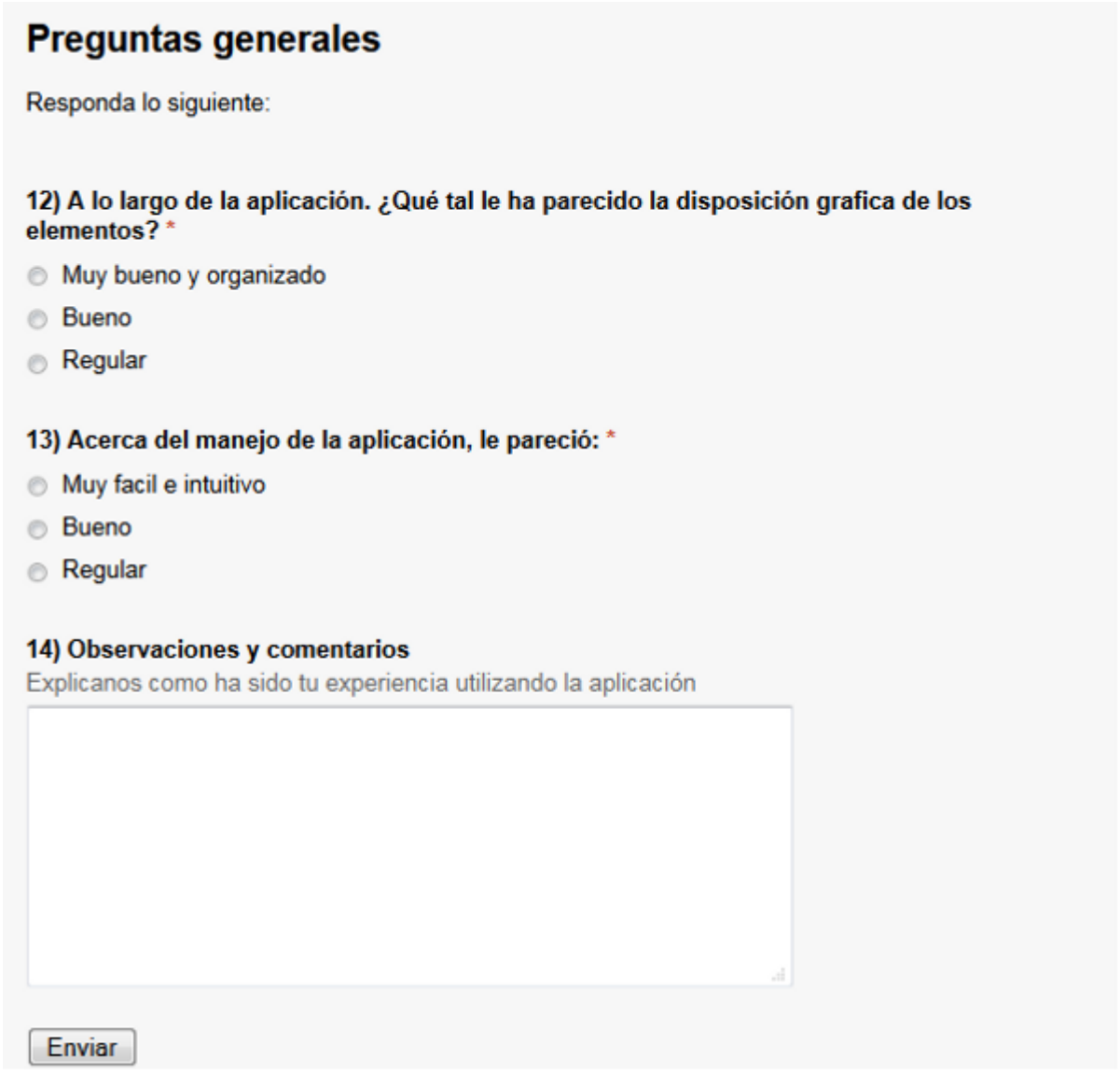
Presione el boton con un icono de estrella

- Si
- No

11) Realice un comentario en el inmueble. ¿Como calificaria el manejo de los comentarios? *

- Bueno
- Regular
- Malo / No entiendo cómo funcionan los comentarios

Figura 4.51: Formulario: Encuesta de pruebas de aceptación (parte: 2 de 3)



Preguntas generales

Responda lo siguiente:

12) A lo largo de la aplicación. ¿Qué tal le ha parecido la disposición grafica de los elementos? *

- Muy bueno y organizado
- Bueno
- Regular

13) Acerca del manejo de la aplicación, le pareció: *

- Muy facil e intuitivo
- Bueno
- Regular

14) Observaciones y comentarios

Explicanos como ha sido tu experiencia utilizando la aplicación

Enviar

Figura 4.52: Formulario: Encuesta de pruebas de aceptación (parte: 3 de 3)

Bibliografía

[Beck, 2011] Beck, K. y Beedle, M. (2001). *Principios del Manifiesto Ágil*. Recuperado de: http://agilemanifesto.org/iso/en/principles.html

[Beck, 1999] Beck, K. (1999). *Extreme Programming Explained, embrace change*. Primera edición.

[The jQuery Foundation, 2013] The jQuery Foundation. (2013). *jQuery Mobile*. Recuperado de: http://jquerymobile.com

[The jQuery Foundation, 2013] The jQuery Foundation. (2013). *jQuery Mobile: Demos and Documentation*. Recuperado de: http://jquerymobile.com/demos/1.2.1/

[Angelov, 2011] Angelov M. (2011). *Building a Website with PHP, MySQL and jQuery Mobile, Part 1*. Recuperado de: http://tutorialzine.com/2011/08/jquery-mobile-product-website

[Adobe Systems Inc, 2013] Adobe Systems Inc. (2013). *PhoneGap*. Recuperado de: http://phonegap.com

[Adobe Systems Inc, 2013] Adobe Systems Inc. (2013). *PhoneGap API Documentation*. Recuperado de: http://docs.phonegap.com/en/2.0.0/guide_getting-started_index.md.html

[Leggett, 2011] Leggett D. (2011). *Considerations for Mobile Design*. Recuperado de: http://www.uxbooth.com/blog/considerations-for-mobile-design-part-1-speed

[SensioLabs, 2012] SensioLabs. (2012). *Día 1: Comenzando el Proyecto Symfony*. Recuperado de: http://www.symfony-project.org/jobeet/1_4/Propel/es/01

[Antón, 2012] Antón O. (2012). HTML5 Mobile (Estadisticas de uso de sistemas operativos moviles). Recuperado de: http://www.moleculartes.com/demos/html5-mobile/#/

[W3Schools, 2013] W3Schools. (2013). *HTML Summary*. Recuperado de: http://www.w3schools.com/html/html_summary.asp

[Hemmendinger, 1998] Hemmendinger D., Ralston A. (1998). *Encyclopedia of Computer Science*. Primera edición.

[W3C, 2013] W3C. (2013). *HTML & CSS*. Recuperado de: http://www.w3.org/standards/webdesign/htmlcss

[W3Schools, 2013] W3Schools. (2013). *HTML5 Introduction*. Recuperado de: http://www.w3schools.com/html/html5_intro.asp

[W3Schools, 2013] W3Schools. (2013). *HTML Introduction*. Recuperado de: http://www.w3schools.com/html/html_intro.asp

[W3Schools, 2013] W3Schools. (2013). *JavaScript Introduction*. Recuperado de: http://www.w3schools.com/js/js_intro.asp

[W3Schools, 2013] W3Schools. (2013). *AJAX Tutorial*. Recuperado de: http://www.w3schools.com/ajax/default.asp

[The jQuery Foundation, 2013] The jQuery Foundation. (2013). *jQuery*. Recuperado de: http://www.jquery.com

[The Apache Software Foundation, 2012] The Apache Software Foundation. (2012). *About the Apache HTTP Server Project*. Recuperado de: http://httpd.apache.org/ABOUT_APACHE.html

[The PHP Group, 2013] The PHP Group. (2013). *PHP: Hypertext Preprocesor*. Recuperado de: http://www.php.net

[The PHP Group, 2013] The PHP Group. (2013). *PHP: What is PHP?*. Recuperado de: http://www.php.net/manual/en/intro-whatis.php

[Oracle Corporation, 2013] Oracle Corporation. (2013). *MySQL: About MySQL*. Recuperado de: http://www.mysql.com/about/

[Silva, 2006] Silva H. (2006). *Sistema Manejador de Base de Datos*. Recuperado de: http://www.emagister.com/curso-procesamiento-datos-oracle/sistema-manejador-base-datos

[W3C, 2013] W3C. (2013). *Standards for web Applications on Mobile: current state and roadmap*. Recuperado de: http://www.w3.org/Mobile/mobile-web-app-state/

[W3C, 2013] W3C. (2013). *Mobile web Application Best Practices*. Recuperado de: http://www.w3.org/TR/2010/REC-mwabp-20101214/#bp

[Opera Software ASA, 2013] Opera Software ASA. (2013). *Opera for developers*. Recuperado de: http://www.opera.com/developer

[GitHub Inc, 2011] GitHub Inc. (2011). *Propel: The fast PHP5 ORM*. Recuperado de: http://propelorm.org

[SensioLabs, 2012] SensioLabs. (2012). *High performance PHP framework for web development*. Recuperado de: http://symfony.com

[SensioLabs, 2012] SensioLabs. (2012). *Symfony: Inside the model layer*. Recuperado de: http://symfony.com/legacy/doc/book/ 1_0/en/08-Inside-the-Model-Layer

[Code Computerlove Ltd, 2012] Code Computerlove Ltd. (2012). Photo Swipe: Image gallery for mobile devices. Recuperado de: http://www.photoswipe.com

[García, 2012] García E. (2012). *Guía para elaborar citas y referencias en formato APA*. Recuperado de: http://www.magisteriolalinea.com/home/carpeta/pdf/MANUAL_APA_ULACIT_actualizado_2012.pdf

[Sun, 2000] Sun Microsystems. (2000). *Desarrollo de Pruebas Unitarias*. Recuperado de: http://www.cjavaperu.com/pdf/pruebas_unitarias.pdf

[Linkedin, 2011] Linkedin. (2011). *¿Qué es una Prueba de Aceptación?*. Recuperado de: http://www.linkedin.com/groups/Qu %C3 %A9-es-Prueba-Aceptaci %C3 %B3n-3636186.S.48805747