UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

Rivence: Aplicación móvil de servicios al ciudadano

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Carlos Quinzá Pérez

Tutor: Germán Moltó Martínez

2015/2016

Resumen

Con el objetivo de cubrir el mercado relacionado con el turismo de lujo en diferentes ciudades se desarrolla la aplicación “Rivence”. En un futuro se espera que forme parte de un modelo de negocio basado en la contratación de servicios de los sectores: hostelería, alquiler de vehículos y ocio.

La estructura del negocio se ve reflejada en la agrupación de componentes dentro del sistema desarrollado para esta memoria. Existe un cliente que, por medio de mensajería chat implementada a medida, transmite al agente asignado qué servicios desea contratar o a qué eventos tiene la intención de asistir. En consecuencia, el agente se encarga de realizar las acciones necesarias para reservar los distintos servicios. Tanto la información acerca de qué se ofrece y el seguimiento de los pagos realizados se visualiza en los respectivos apartados del menú de la aplicación.

El software está desarrollado para dispositivos Android, con una estructura Modelo-Vista-Controlador a nivel de sistema. A nivel de interfaz contiene el patrón de diseño de Android (RecyclerView – Cardview). El entorno de desarrollo ha estado compuesto por el *framework* “Android Studio” (para la capa lógica y el diseño de interfaz), el programa “Workbench” para la gestión de base de datos y el sistema de control de versiones “GitHub”.

La finalidad de este proyecto se basa en la generación de información acerca de cómo desarrollar una aplicación móvil concretando sobre los problemas obtenidos y las soluciones implementadas.

Palabras clave: Android, Workbench, GitHub, Cardview, Recycler View, móvil, aplicación.

Abstract

The "Rivence" application has been developed with the purpose of addressing the luxury tourism market focused in several cities. As a long-term objective it will be part of a particular business model based on the procurement of expert services in travel and holiday industries: catering, leisure and vehicle rental.

The business structure is reflected in the particular component grouping shown in the system developed. By using the custom instant messaging tool implemented, a client transmits to his personal agent assigned which services wants to hire or to which events wants to attend. As a consequence, the agent performs the necessary actions to book the corresponding services. Every relevant information and status tracking of bookings and payment made can be accessed via the different sections of the application menu.

The software has been developed for Android devices, using a Model-View-Controller structure, at system level. At the interface level, it contains Android design pattern (RecyclerView - Cardview). The development environment included "Android Studio" framework (for logical layer and interface design), "Workbench" program for database management and "GitHub" as version control system.

The main goal of this project is based on the generation of information related to the development of a mobile application, focused on how the problems detected have been solved and the detail of solutions implemented.

Key words: Android, Workbench, GitHub, Cardview, Recycler View, mobile, application

Resum

L’aplicació "Rivence" ha sigut desenvolupada per a cobrir el mercat del turisme de luxe en diferents ciutats. En el futur, s'espera que siga part d'un model de negoci basat en la contractació de serveis dels sectors d’hostatgeria, lloguer de vehicles i oci.
L'estructura de negoci es veu reflectida en l’agrupació de components desenvolupada. Existeix un client que, mitjançant el servei de missatgeria xat implementada, transmet a l'agent assignat quins serveis dessitja contractar i a quins events vol assistir. En consequència, l'agent s'encarrega de fer les gestions oportunes per a reservar els serveis. La informació dels serveis oferits, així com el seguiment dels pagaments realitzats es visualitza des de les diferents seccions del menu de l’aplicació.
El software ha sigut desenvolupat per a dispositius Android, amb una estructura Model-Vista-Controlador, a nivell de sistema. A nivell d'interfaç conté el patró de disseny d'Android (RecyclerView - Cardview). L'entorn de desenvolupament ha estat compost pel framework "Android Studio" (per a la capa lògica i el disseny d'interficie), el programa "Workbench" per a la gestió de base de dades i el controlador de versions "GitHub".
La finalitat d'aquest projecte es basa en la generació d'informació focalitzada en com desenvolupar una aplicació mòbil, concretament sobre els problemes obtinguts i les solucions implementades.

Paraules clau: Android. Workbench, GitHub, Cardview, Recycler View, mmòbil, aplicació.

Tabla de contenidos

1. Esquema de figuras 7
2. Introducción 8
3. Plataformas utilizadas 10
4. Patrones implementados 16
4.1 Modelo – Vista - Controlador 16
4.1.1 Vista 18
4.1.2 Controlador 20
4.1.3 Modelo 21
4.2 Patrón Singleton 22
4.3 Patrón ViewHolder 23
4.4 Patrón de Fragmentos 26
4.5 Programación para tamaños de pantalla distintos 30
5. Trabajando con base de datos 32
5.1 Diseñando la estructura 32
5.2 Generando código SQL e insertándolo en nuestra Base de datos 32
5.3 Rellenado de batería de datos 35
5.4 Creando puerta de acceso en la Base de datos 36
5.5 Consultas a Base de datos desde JAVA 39
6. Técnicas de refactorización y mantenimiento 43
6.1 Paleta general de colores 43
6.2 Nomenclaturas 45
6.3 Documentación Javadoc 47
7. Estado final de la aplicación 49
8. Futuras implementaciones 50
9. Conclusiones del proyecto 51
10. Librerías usadas y agradecimientos 52
11. Bibliografía 53

1. Esquema de figuras

Ilustración 1 Ocupación hotelera general (2008 – 2013) _____ 8
Ilustración 2 Sectores implementados en la aplicación _____ 9
Ilustración 3 Captura de Android Studio _____ 11
Ilustración 4 Interfaz del programa GitHub Desktop para OS X _____ 12
Ilustración 5 Diagrama inicial realizado en MySQL Workbench _____ 13
Ilustración 6 Captura de la herramienta Trello día 07/06/2016 _____ 14
Ilustración 7 Captura de imagen de Adobe Illustrator _____ 15
Ilustración 8 Representación del patrón MVC _____ 16
Ilustración 9 Ejemplo básico patrón MVC _____ 17
Ilustración 10 Posición de la interfaz en el árbol de documentos _____ 18
Ilustración 11 Fragmento de layout en formato XML _____ 19
Ilustración 11 Iconos utilizados en la aplicación _____ 19
Ilustración 13 Referencia a la Vista desde Controlador _____ 20
Ilustración 14 Diagrama de clases del programa _____ 20
Ilustración 15 Definicion de variable de tipo Cliente _____ 22
Ilustración 16 Constructor privado _____ 22
Ilustración 17 Método getInstance de Cliente _____ 22
Ilustración 18 Diagrama representativo patrón ViewHolder _____ 23
Ilustración 19 Reperesentación de fragmentos _____ 26
Ilustración 20 Ventana main.xml _____ 27
Ilustración 21 Equivalencias entre distintos tamaños de pantallas _____ 30
Ilustración 22 Categoría en función de tamaño de pantalla y densidad Fuente: Stackoverflow) _____ 31
Ilustración 23 Acceso Panel de Administración____ **¡Error! Marcador no definido.**
Ilustración 25 XML "Paleta de colores" _____ 44
Ilustración 26 Ejemplo asignación de colores _____ 44
Ilustración 27 Visión del árbol de interfaces _____ 45
Ilustración 28 Captura Javadoc del Proyecto _____ 47

2. Introducción

El turismo en la Comunidad Valencia siempre ha sido uno de los pilares económicos más grandes, y es un sector que se nutre del buen clima, posicionamiento geográfico y en consecuencia existe una gran inversión por parte de los organismos públicos y las empresas privadas.
Es inevitable que cada vez más la tecnología, en este caso la informática, tome un camino en la facilitación de este tipo de actividades. En concreto vamos a focalizar el punto de vista en el turismo de lujo, ese turismo que se mantiene invulnerable a la recesión económica y solamente sigue el camino del incremento y los beneficios.

GRADO DE OCUPACIÓN HOTELERA (%)

	2008	2009	2010	2011	2012	2013
Alicante	60,4	55,7	57,1	60,5	58,5	61,2
Castellón	46,4	48,2	47,0	49,1	47,3	47,3
Valencia	47,8	43,0	44,8	*43,8*	43,8	44,8
C. Valenciana	*54,5*	*50,7*	*51,9*	*53,8*	*52,5*	*54,3*
España	53,5	49,3	51,0	53,5	52,0	53,1

Fuente: I.N.E.

Ilustración 1 Ocupación hotelera general (2008 – 2013)

Varias iniciativas han sido arrancadas en la zona mediterránea relacionadas con este turismo de lujo. Desde buscadores específicos de alojamiento (p.e Valencia Luxury – creada en 2007) hasta agrupaciones de empresarios (p.e Valencia Premium Group – creada en 2014) pasando por infraestructuras tanto públicas como privadas (Marina Beach Club -2016- , America's Cup Valencia, Circuito de Fórmula 1 en Marina Real).

Sin embargo, no hay ninguna plataforma que informe al turista de alto nivel económico sobre los eventos y servicios que ofrece la ciudad levantina (Hospedaje, centros de ocio, restaurantes, eventos culturales, etc.). Con este contexto se desarrolla la aplicación "Rivence", un sistema que pretende agrupar cada una de las actividades existentes de la ciudad, tal y como indica la siguiente representación.

Este proyecto ha sido idea personal, y me he decidido a llevarlo a cabo como reto personal y motivado por la intención de aprender en el desarrollo de aplicaciones móviles, y en un futuro considerar la puesta en marcha de este negocio.

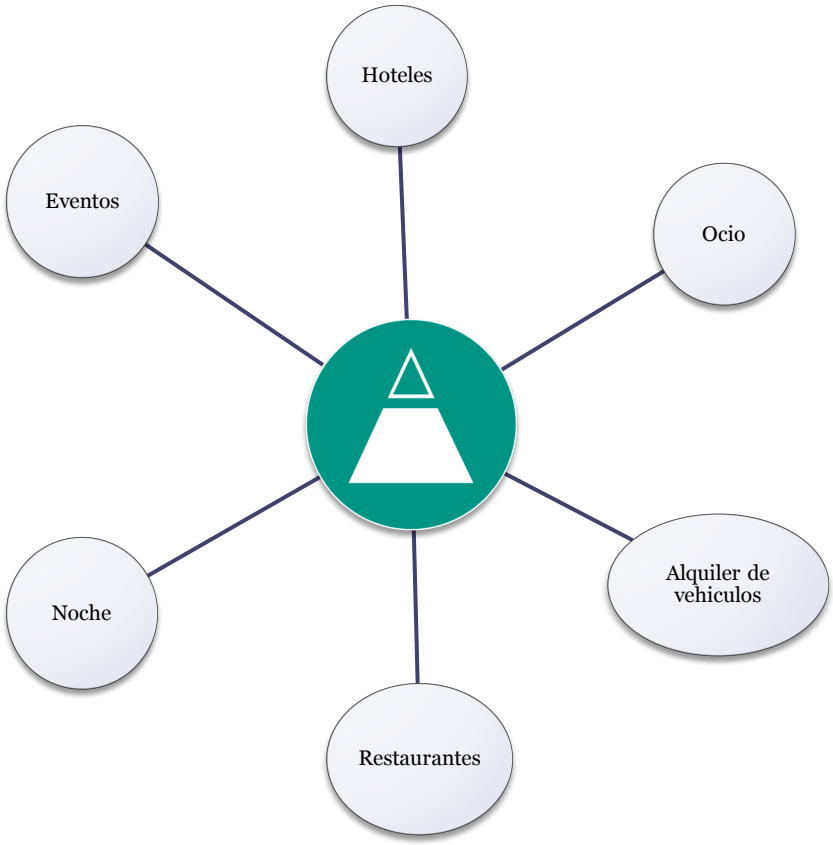


Ilustración 2 Sectores implementados en la aplicación

En esta memoria se recopila toda la información relevante a cerca del desarrollo de la aplicación. Contiene: una vista general de los entornos de desarrollo y herramientas que han sido utilizadas, los patrones arquitectónicos bajo los que se asienta el código de la aplicación, técnicas de refactorización, hechos relevantes ocurridos durante el periodo de programación y una visión acerca de las futuras implementaciones que se podrían añadir fuera de este contexto académico.

El objetivo de esta memoria es reunir toda la información que sirve para comprender el proceso de desarrollo que se ha llevado a cabo, asimilar la estructura de la aplicación, informar de los problemas que han aparecido y por último, aprender sobre las soluciones y optimizaciones utilizadas.

Para ello, el orden lógico de este documento comienza desde una vista general, desde la cual se introducen los conceptos más relevantes que, posteriormente son utilizados en los diferentes apartados, hasta la vista concreta a nivel de código en zonas puntuales que requieren especial atención y explicación.

3. Plataformas utilizadas

En este apartado se describe qué plataformas y herramientas se han utilizado para desarrollar el Proyecto.

Android Studio.

El famoso entorno de desarrollo integrado para los sistemas "Android" del cual es propietaria la empresa del sector informático "Google".

Ofrece varios fragmentos que son de gran utilidad a la hora de desarrollar aplicaciones. Su gestor y compilador de Java para la intercepción de eventos de la interfaz, la consola que captura de forma dinámica la información de rendimiento y su *framework* para el diseño de interfaces son algunos de sus puntos fuertes. Es la alternativa al *plugin* ADT de Eclipse, también de software libre. Pasó a ser el IDE recomendado por Google desde que fue liberada la versión estable (el 8 de Diciembre de 2014) hasta la fecha.

Únicamente con la descarga del programa ya se disponen de todas las herramientas para el desarrollo de aplicaciones para la plataforma Android, evitando la búsqueda de *frameworks* y *plugins* adicionales.

Además, su nueva forma de construir los paquetes .apk mediante el uso "Gradle" proporciona una reutilización de código y recursos para poder compilar desde línea de comandos (si no tenemos disponible un entorno de desarrollo) y la distribución de código para poder trabajar en equipo.

Es por ello, que en este proyecto se ha decidido utilizarlo tanto para la interfaz como la lógica. Estos conceptos serán explicados posteriormente en un apartado específico.

Sin embargo, también existe un inconveniente a la hora de usar Android Studio. El hecho de no haber trabajado con este IDE en el grado y de la inserción de "Gradle" produce una curva de aprendizaje más lenta donde, por consecuencia, se exige mayor tiempo de dedicación para un mismo resultado que en otras plataformas.

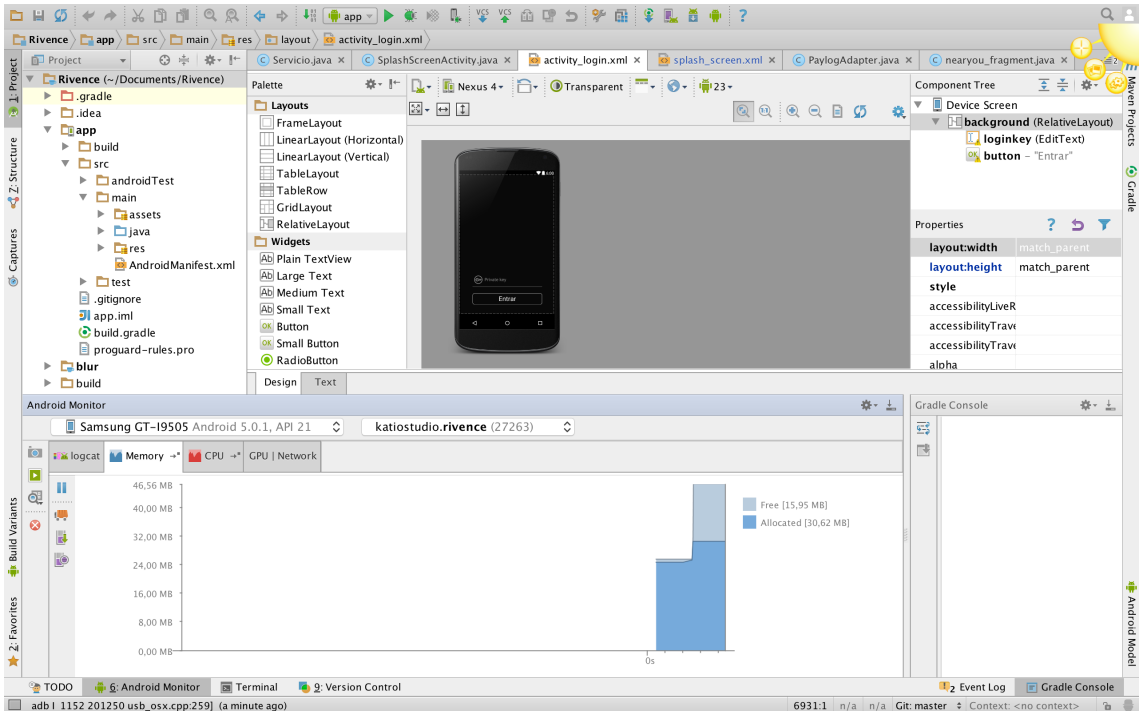


Ilustración 3 Captura de Android Studio

Como puede apreciarse en la Ilustración 3, se distinguen las partes mencionadas anteriormente. En la parte izquierda de la ventana se posiciona el árbol de documentos del proyecto. En la parte inferior las diferentes opciones de seguimiento de ejecución del programa y por último, en la parte central, el entorno de desarrollo de interfaces (que puede realizarse tanto a nivel de diseño como de texto).

GitHub. Enlace del proyecto (https://github.com/carquipe/Rivence)

En todo proyecto software surge la necesidad de mantener y llevar control del código que se va programando, conservando todos los estados. Es por ello que es necesario un sistema de control de versiones cuando se trabaja colectivamente o independientemente. Nos centraremos solamente en las ventajas cuando se trabaja individualmente ya que es el contexto en el que se ha desarrollado el proyecto:

- Posibilidad de comparar el código de un archivo, de modo que se puede ver las diferencias entre versiones
- Capacidad de restaurar versiones antiguas
- Permite fusionar cambios entre distintas versiones
- Posibilita trabajar con distintas ramas de un proyecto, por ejemplo la de producción y desarrollo

Existen muchos tipos de sistemas de control de versiones: Subversion, Git, CVS entre otros. En nuestro caso, se busca un sistema distribuido, donde en cada equipo que se trabaje se almacene una copia local del repositorio completo, debido a que se ha trabajado en dos sistemas diferentes (PC Windows y MacBook Pro).

El siguiente planteamiento que surgió fue si instalar Git en el ordenador o utilizar GitHub que nos ofrece almacenar en la nube nuestro repositorio.

GitHub es uno de los sistemas de control de versiones más destacados en la actualidad. Las dos características más relevantes son: Su gran compatibilidad con otros entornos (p.e Android Studio proporciona varias utilidades dentro de su plataforma para añadir, borrar, modificar los documentos del repositorio automáticamente) y su programa independiente de control de versiones.

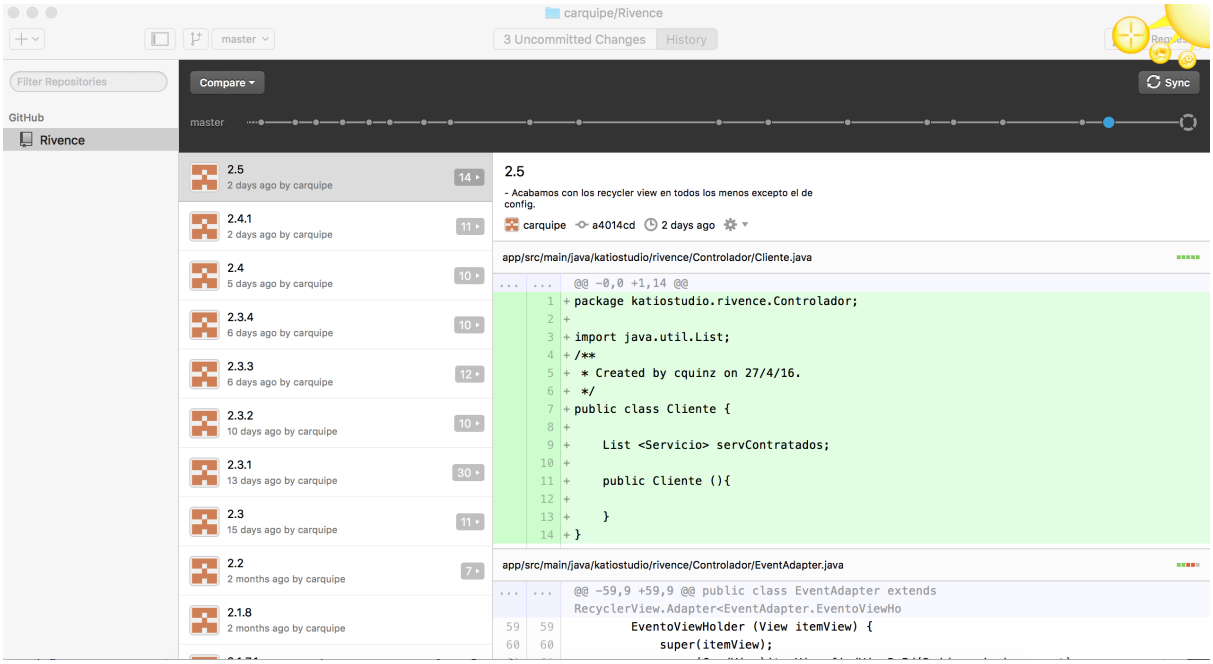


Ilustración 4 Interfaz del programa GitHub Desktop para OS X

En la Ilustración 4 se puede observar el seguimiento de versiones que ha adquirido la aplicación y los cambios de cada una a nivel de código y de documentos. En el proyecto ha servido de gran ayuda para tener un control de cada avance y también para añadir portabilidad a la hora de trabajar desde diferentes ordenadores. Es indispensable para proyectos medianamente grandes en los cuales se va a necesitar un control de los cambios en el sistema y en cualquier momento poder volver atrás en los pasos sin gran esfuerzo.

MySQL Workbench.

> "*MySQL Workbench es una herramienta visual de diseño de bases de datos que integra desarrollo de software, Administración de bases de datos, diseño de bases de datos, creación y mantenimiento para el sistema de base de datos MySQL.*" – **Wikipedia (2016)**

Tal y como se ha explicado anteriormente, Workbench proporciona una gestión fácil y visual de la base de datos de nuestra aplicación.

En primera instancia se utilizó para el diseño del diagrama de la estructura de la persistencia y posteriormente la generación automática del código SQL de creación en base a las tablas diseñadas.

Más tarde, a pesar de no haber sido utilizada con gran frecuencia, el mantenimiento y la observación de la información almacenada se ha realizado a través de este programa.

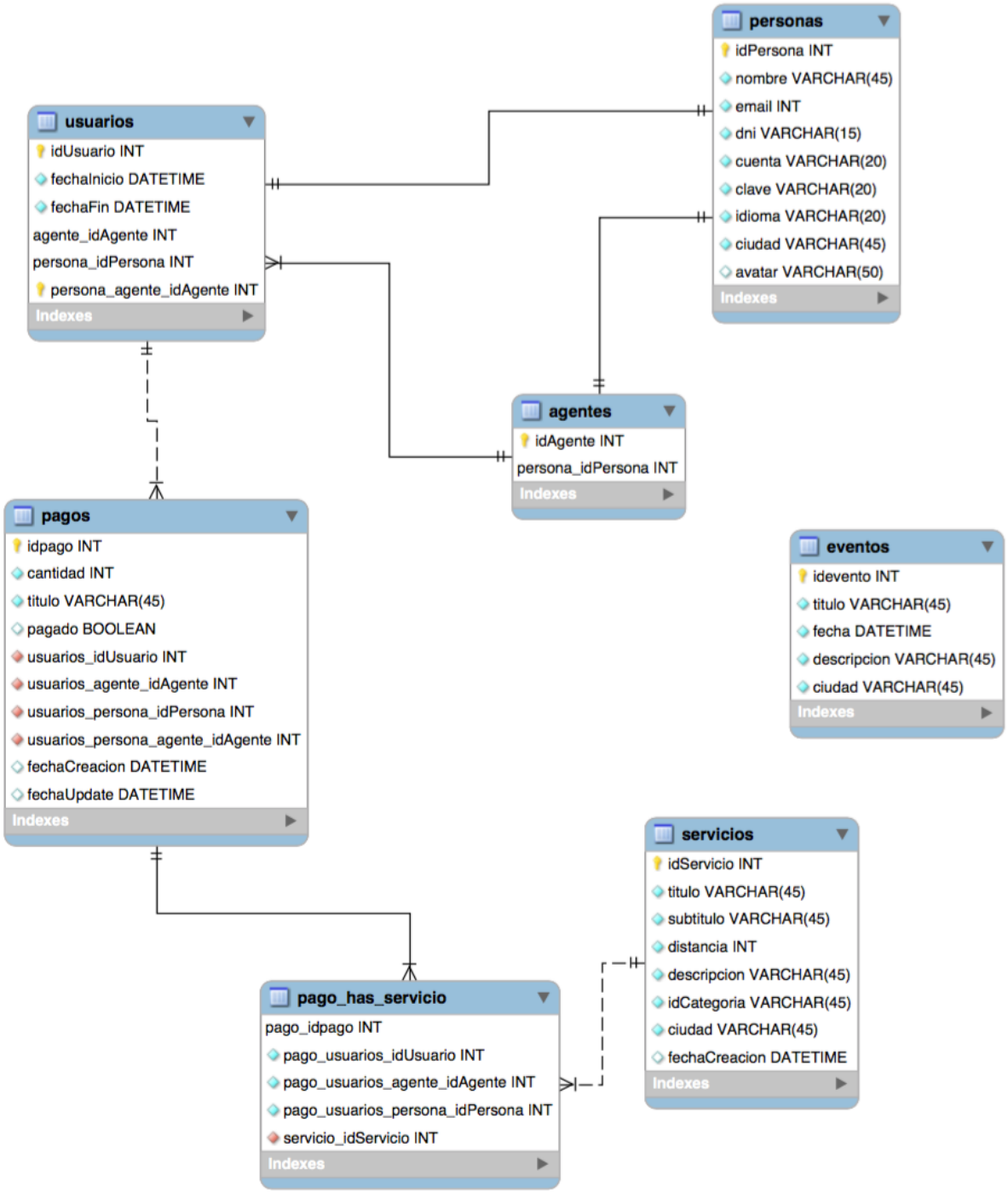


Ilustración 5 Diagrama inicial realizado en MySQL Workbench

Trello.

Es una herramienta online que permite tener varios *boards* separados según tu criterio personal. Está muy ligado a la metodología ágil por su posible separación de tareas y el seguimiento continuo de éstas,

Un *board* de Trello es básicamente una página web que contiene listas dispuestas de manera horizontal de modo que puedas apreciar, de un vistazo, todo lo que hay en tu proyecto. Los ítems dentro de las listas, llamados *cards*, pueden arrastrarse y soltarse en otras listas o reordenarse. Estas listas pueden llamarse según el criterio del usuario y son totalmente personalizables.

En el contexto del proyecto, se ha utilizado un *board* con las listas:

1- **Brainstormed:** Ideas o tareas que en el momento actual no se ha decidido si implementarse.

2- **To Do:** Tareas pendientes por hacer, normalmente se mantienen más tiempo aquí aquellas cuya prioridad es baja.

3- **In progress:** Acciones que se están investigando o llevando a cabo.

4- **Review:** Toda tarea necesita su revisión y sus pruebas. Los *cards* que se encuentran en esta lista aún no se han validado.

5- **Done:** Finalizado. Puede descartarse.

La aportación que ha ofrecido esta herramienta ha sido principalmente similar a la de una agenda: una mejor organización y estimación de tiempos (gracias a las fechas de vencimiento y los checklists de cada elemento)

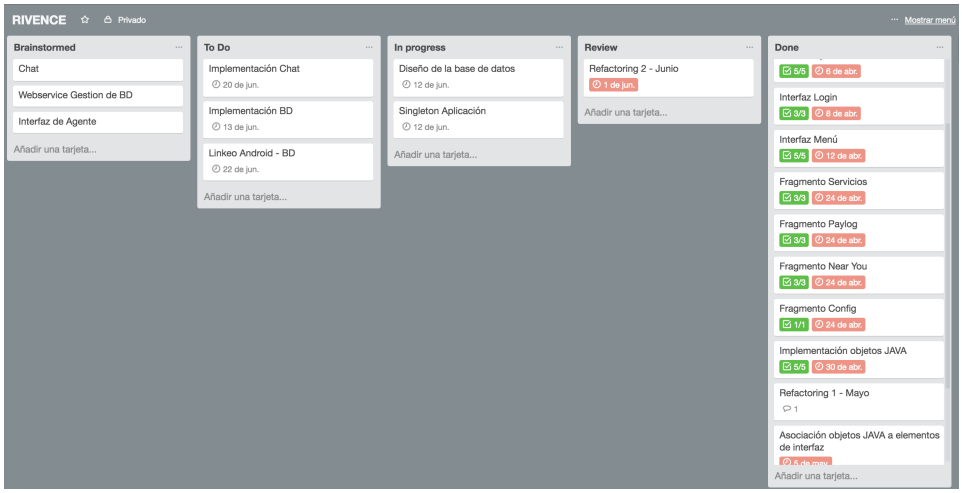


Ilustración 6 Captura de la herramienta Trello día 07/06/2016

Adobe Illustrator.

A pesar de no estar estrictamente relacionado con la finalidad de esta memoria, cabe mencionar brevemente este programa ya que se ha utilizado para la gran mayoría de los componentes gráficos situados en la interfaz. El diseño vectorial y a distintos tamaños ha sido una pieza clave a la hora de mantener una visualización correcta en todo tipo de pantallas. En caso contrario, el pixelado de los iconos y las imágenes hubiese sido un problema significativo en la percepción del usuario.

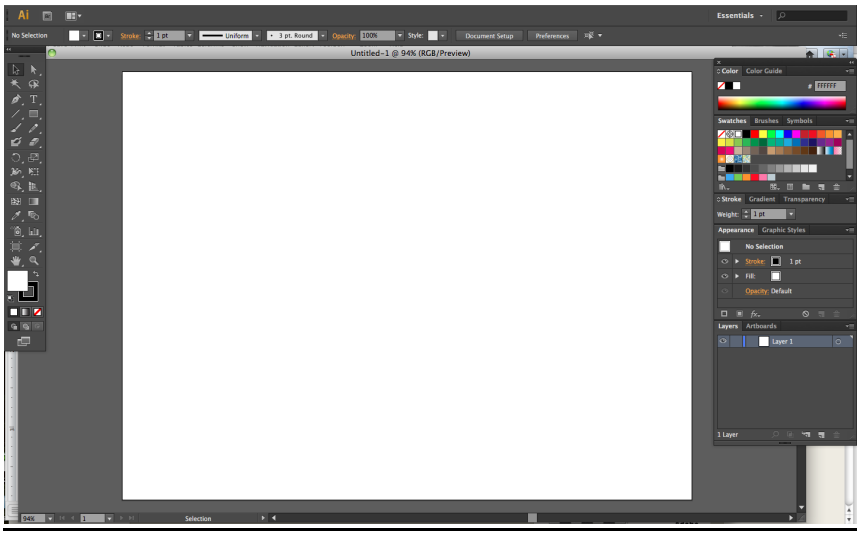


Ilustración 7 Captura de imagen de Adobe Illustrator

4. Patrones implementados

4.1 Modelo – Vista - Controlador

El proyecto sigue el patrón de arquitectura denominado "Modelo Vista Controlador". Es un patrón que organiza independientemente sus tres componentes permitiendo modificaciones en cualquiera de las partes sin que haya que hacer posteriormente cambios en las otras dos.

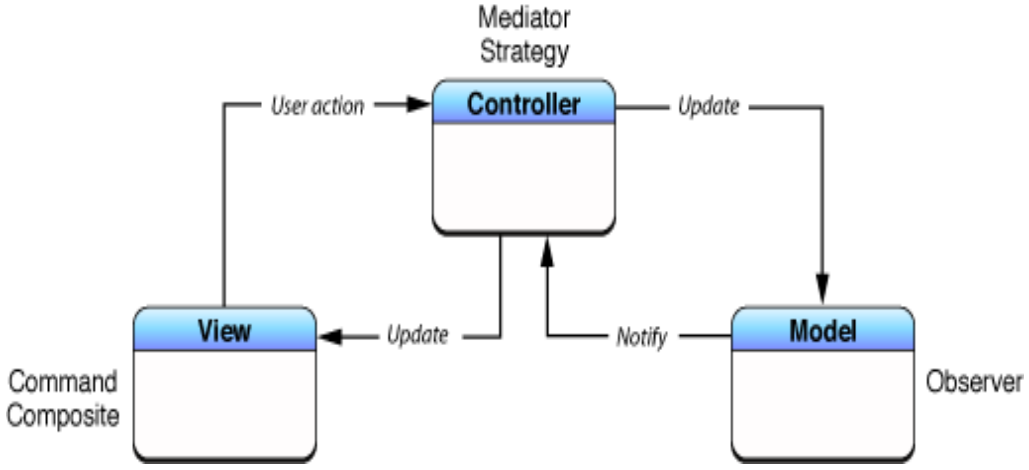


Ilustración 8 Representación del patrón MVC

Los tres componentes que se describen en los sub-apartados están comunicados mediante eventos y llamadas desde las clases del controlador. Existen dos tipos de mensajes: de consulta o de modificación. Los primeros consisten en recibir información de una parte (modelo, vista o controlador) y los segundos en modificar algún objeto/elemento. Pero, la modificación de un componente en uno de los tres apartados no debe provocar la parada de funcionamiento de las otros apartados, en este caso no se estaría consiguiendo la finalidad de este patrón.

Unas de las múltiples ventajas de este patrón son las siguientes:

- Hace el proyecto más escalable, ya que divide la lógica de negocio del diseño.
- La extensibilidad y mantenibilidad del sistema es mayor que en sistemas que no tienen implementado este patrón, ya que al modificar una capa no ejerce influencia directa en otra (Modularidad).
- A la hora de trabajar varias personas, hace más sencillo el trabajo debido al número bajo de dependencias.

Veamos un ejemplo básico:

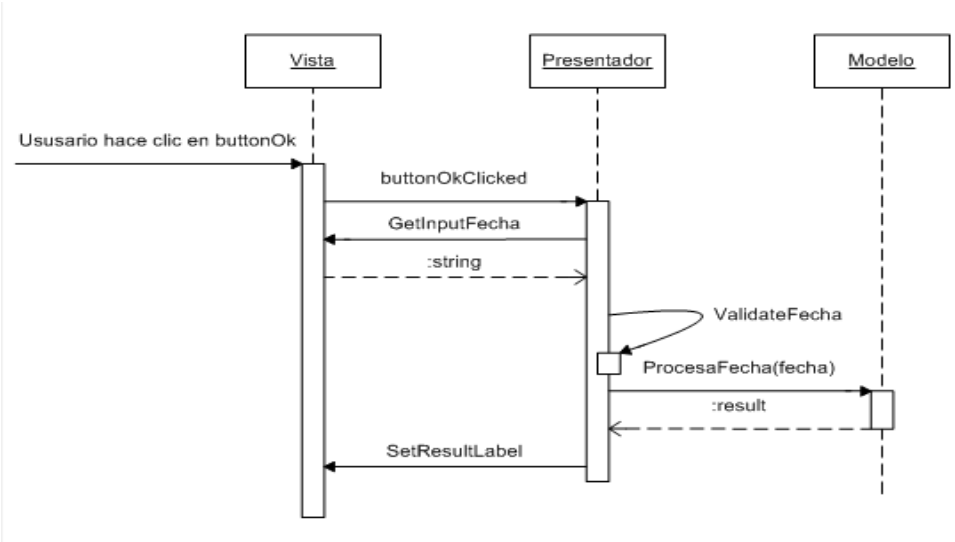


Ilustración 9 Ejemplo básico patrón MVC

En este diagrama de secuencia se puede ver a rasgos generales el proceso de un evento generado a partir de la pulsación de un botón en una aplicación de móvil que tendrá como objetivo modificar la fecha en la base de datos y en la pantalla de visualización.

Está acción la registra el presentador (Controlador en nuestro caso) y es el que ejecuta la rutina que consiste en obtener el valor que el usuario ha introducido en un contenedor visual en formato fecha.

Posteriormente, se ejecuta el método “ValidateFecha” que realiza una llamada al modelo mediante el método “ProcesaFecha(fecha)” que es el que conecta la base de datos con el controlador en este caso. Se modifica en la capa de persistencia y cuando esta acción se realiza correctamente sin fallos, se modifica en la vista el valor introducido al principio.

4.1.1 Vista

La vista la componen todos las plantillas del proyecto (los archivos XML) que se dedican a dar una visión al usuario de todo lo que compone el Modelo.

Ésta contiene varios tipos de objetos "Layouts" en los que en su interior se generan Textos, Imágenes, Botones, Tablas etc. Estos Layouts pueden contenerse entre ellos. Son muy importantes a la hora de hacer un diseño de interfaz que se acople a una gran variedad de pantallas debido a su flexibilidad y alta personalización de cada uno.

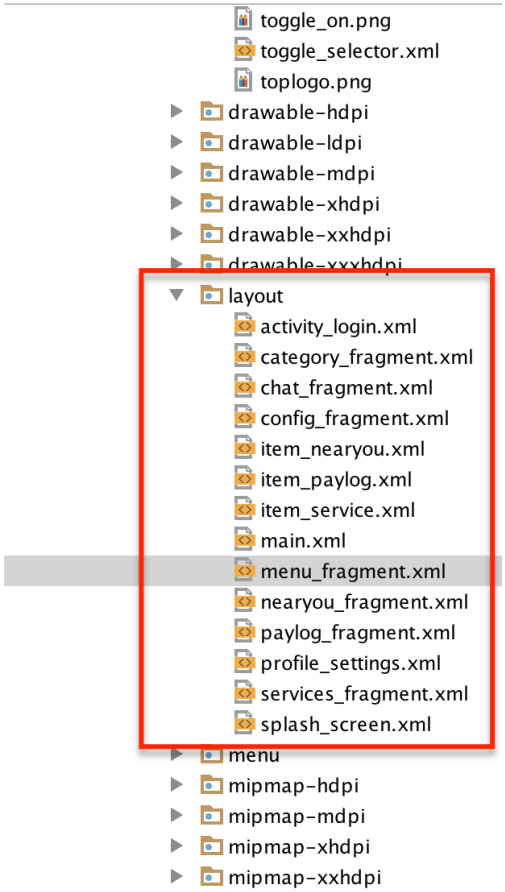


Ilustración 10 Posición de la interfaz en el árbol de documentos

Estos archivos XML se pueden diseñar de dos maneras diferentes si utilizamos la herramienta Android Studio. Como ya hemos comentado en el apartado "2. Plataformas utilizadas", existe una herramienta gráfica con varios menús de soporte y la otra posibilidad, es mediante código tal como se puede observar en la Ilustración 10. En este proyecto se ha optado por

una mezcla de las dos posibilidades, dependiendo de las necesidades del diseño.

```
        <TextView
            android:id="@+id/subtitle"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textSize="12sp"
            android:layout_below="@+id/service_title"
            android:layout_alignParentLeft="true"
            android:layout_alignParentStart="true"
            android:textColor="@color/white" />

        <TextView
            android:id="@+id/distance"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:textSize="12sp"
            android:layout_below="@+id/subtitle"
            android:layout_alignParentLeft="true"
            android:layout_alignParentStart="true"
            android:textColor="@color/white" />

        <Button
            android:id="@+id/read_more_pay"
            android:layout_width="30dp"
            android:layout_height="30dp"
            android:textSize="10sp"
            android:background="@drawable/moreinfo_ic"
            android:layout_below="@+id/service_title"
            android:layout_alignParentRight="true"
            android:layout_alignParentEnd="true" />

    </RelativeLayout>
```

Ilustración 11 Fragmento de layout en formato XML

En la parte de la Vista, también se incluyen todo tipo de archivos que forman parte de la visualización del usuario. Es decir, en este caso nos referimos a: Imágenes, iconos y descriptores externos en formato XML. En efecto, si en algún momento alguno de estos se elimina, reemplaza o modifica, cambiará en lo que perciba el usuario.

Ilustración 12 Iconos utilizados en la aplicación

4.1.2 Controlador

Los Controladores son un enlace entre los modelos y las vistas. Se encargan de controlar los eventos e interacciones de la vista y proporcionar los datos necesarios del modelo para su funcionamiento. Éstos son los encargados de cargar una vista y sus recursos cuando se necesiten y eliminarlos cuando sea necesario.

Los archivos que componen este apartado, en nuestro caso, están basados en el lenguaje de programación Java. Por una parte tienen variables que referencian a elementos de la interfaz y se activan "Listeners" sobre ellos:

```
final ImageButton menu1 = (ImageButton) rootView.findViewById(R.id.Menu1);
final ImageButton menu2 = (ImageButton) rootView.findViewById(R.id.Menu2);
final ImageButton menu3 = (ImageButton) rootView.findViewById(R.id.Menu3);
final ImageButton menu4 = (ImageButton) rootView.findViewById(R.id.Menu4);


menu1.setOnClickListener(this);
menu2.setOnClickListener(this);
menu3.setOnClickListener(this);
menu4.setOnClickListener(this);
```

Ilustración 13 Referencia a la Vista desde Controlador

Y por la otra, contienen métodos y referencias al modelo:

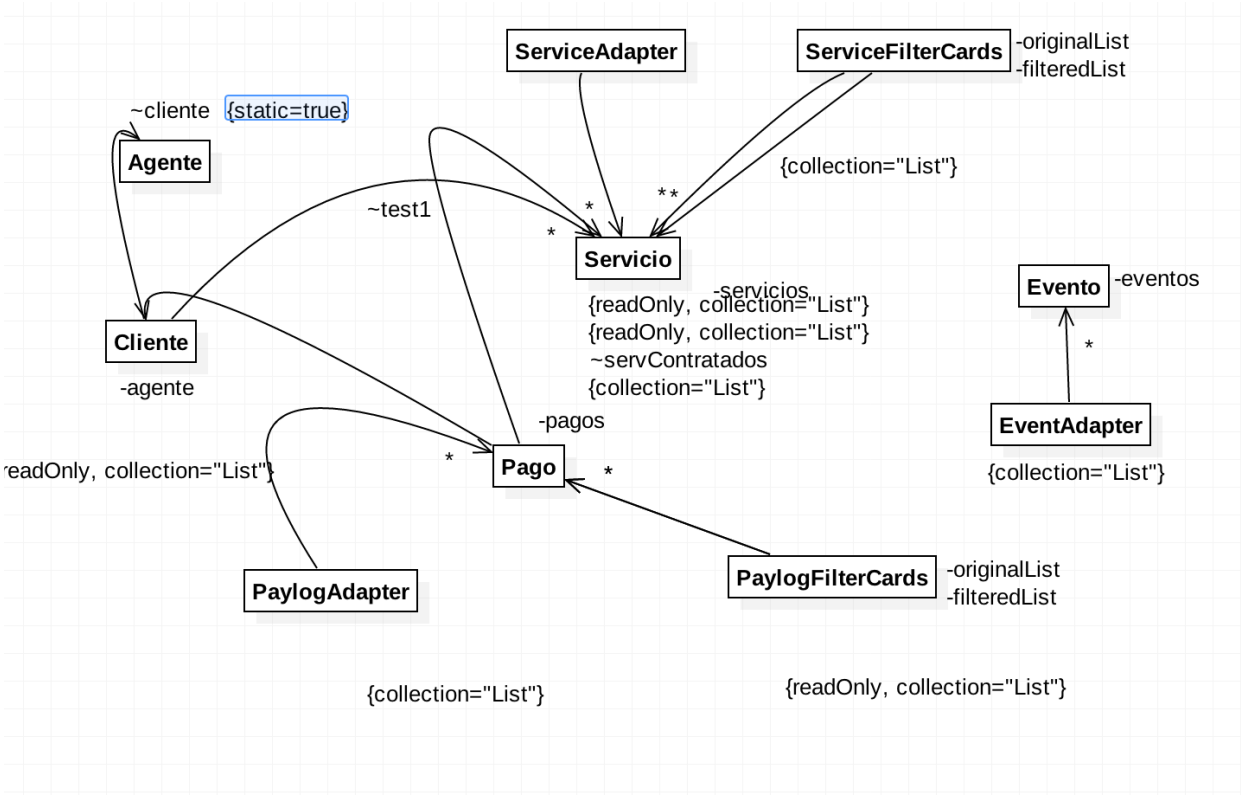


Ilustración 14 Diagrama de clases del programa

4.1.3 Modelo

Contiene el dominio de la aplicación encapsulando su estado y conservando los datos de los objetos de negocio. En nuestro caso es la base de datos que almacena toda la información de los objetos. Esta capa contiene una estructura estrechamente relacionada con el funcionamiento del proceso. Contiene los objetos: Servicio, Pago, Cliente, Agente, Evento. Cada una de las tablas de objeto incluye toda la información real.

Nota: La ilustración 5, contiene el esquema de la base de datos. También puede encontrarse en los archivos adicionales, en el apartado de mantenimiento.

El tratamiento de esta capa contiene tres segmentos diferentes. El primero consiste en el tratamiento de la estructura de tablas con el gestor de base de datos *MySQL Workbench (Añadir/Modificar/Eliminar tablas y establecer relaciones entre ellas)*. En segundo lugar, el tratamiento de datos (insertar valores en las tablas previamente establecidas) se ha hecho desde una interfaz basada con el framework visual "Bootstrap" y personalizada especialmente para este proyecto (la explicación de este segmento se hace más adelante). Por último, la transferencia de valores entre la aplicación y la base de datos consiste en un archivo index.php que contiene un conjunto de apartados distinguidos por un título distintivo. Cada uno de estos apartados realiza una consulta diferente a la de los demás (en SQL) y devuelve un objeto de tipo JSON (Java Script Object Notation) que será tratado por la lógica.

```
switch($tag){
            case 'login':
            $user= NULL;
            $key = $_POST['key'];

        // comprobamos el usuario
        $user = $db->GetUserByKey($key);
        if ($user != NULL) {
            echo json_encode($user) ;
        } else {
            echo $user ;
        }
            break;

            case 'pagos':
            $cliente = $_POST['cliente'];
            $pagos = $db->GetPagosbyCliente($cliente);
            echo json_encode($pagos);
            break;

            case 'servicios':
            $ciudad = $_POST['ciudad'];
            $servicios = $db->GetServiciosByCiudad($ciudad);
            echo json_encode($servicios);
            break
```

Fragmento del fichero index.php

4.2 Patrón Singleton

Hay veces, en las que se crean clases de objeto en las que solo queremos que se instancien una vez, por ejemplo, si estamos diseñando la lógica del planeta Tierra, existirá una clase llamada "Sol" que contendría atributos como: calor, posición y órbita. Además se pueden implementar rutinas dentro de ella, a rasgos hipotéticos: aparecer(), desaparecer(), enfriarse(), desplazarse(), cambiarOrbita().

Ahora bien, ¿Sería correcto poder crear más de un Sol? Según la lógica real, no. En este punto toma importancia el patrón Singleton, que consiste precisamente en restringir la creación de un objeto a una solamente en todo el sistema. Aplicado este patrón sobre la clase Sol, en ningún momento de la ejecución se podrá tener más de una instancia a la clase Sol.

Para poder conseguir este comportamiento, basta con cumplir estos tres puntos:

- Crear un atributo con el mismo tipo de la clase que la contiene.
- El constructor de la clase debe de ser privado, para que solo se pueda llamar desde dentro de la misma.
- Debe existir un método llamado "getInstance()" que se encargue de: si la variable mencionada en el primer punto es nula, crear el objeto y devolverlo. En caso contrario devolverlo únicamente.

En este proyecto, se ha aplicado en dos clases diferentes: En el Cliente, ya que solamente puede haber un cliente ejecutando la aplicación durante el tiempo de vida de ésta. También en el Agente, porque solo puede haber uno de estos asociado a cada Cliente. Para poder acceder a ellos en cualquier punto del proyecto, basta con llamar al método "getInstance()", lo cual hace muy fácil la referencia a estos.

```
public class Cliente {

    /*  Declaración de variables Globales   */

    private static String ciudad, fechaFin, clave;
    private static Cliente cliente;
    private static Agente agente;
```

Ilustración 15 Definicion de variable de tipo Cliente

```
private Cliente(String idPerson, String city, String endDate, String key, String name) {
    id = idPerson;
    ciudad = city;
    fechaFin = endDate;
    clave = key;
    agente = agente.getInstance();
    nombre = name;
    pagos = new ArrayList<>();
    servicios = new ArrayList<>();
}
```

Ilustración 16 Constructor privado

```
public static Cliente getInstance() {return cliente;}
```

Ilustración 17 Método getInstance de Cliente

4.3 Patrón ViewHolder

En las etapas iniciales del proyecto, el consumo de memoria de procesamiento era significativamente elevado, situación que provocaba la detención de la aplicación debido al error en ejecución: “Out of Memory”.

Una de las medidas adoptadas ha sido incluir el patrón “ViewHolder” en cada uno de los fragmentos del sistema que muestran una lista de elementos consiguiendo una reducción de procesamiento y un mejor tratamiento de los recursos por parte de Android.

Básicamente, se trata de crear un número acotado de interfaces de ítems a visualizar *layouts* en vez de crear tantos elementos gráficos como objetos reales existen en el programa.

Por ejemplo, teniendo una lista de 100 servicios a ofertar, en vez de crear 1 *layout* por objeto, se reduce a 7 instancias, que se referencian a cada uno de los objetos que se visualizan en pantalla. En caso de que el usuario realice un desplazamiento y un apartado desaparezca del campo de visión, éste se referencia al siguiente objeto por la parte inferior (suponiendo que se hace Scroll hacia abajo). Evitando así el coste de volver a construir otra plantilla más cuando tendríamos otras sin utilizar en la parte no visible. Tal y como se describe en la siguiente ilustración.

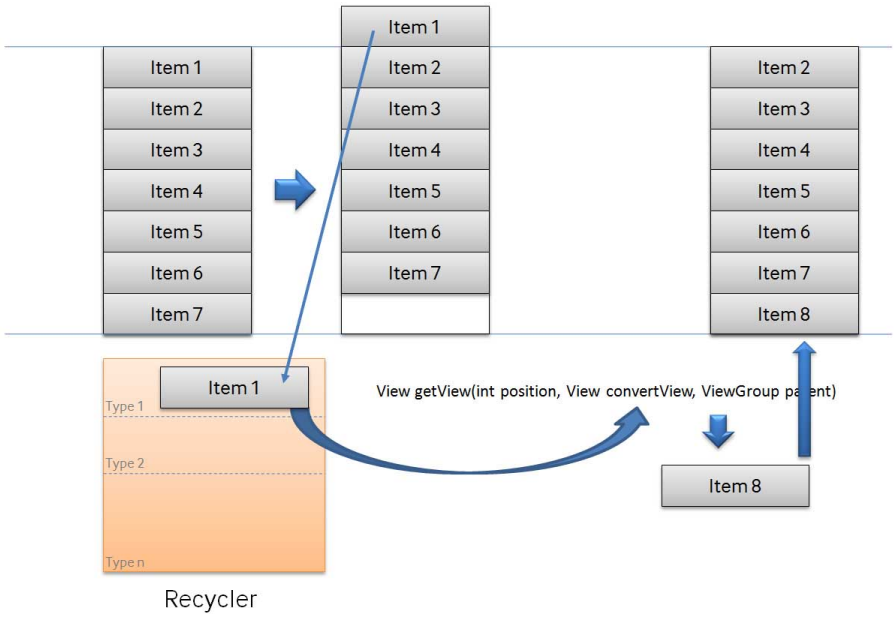


Ilustración 18 Diagrama representativo patrón ViewHolder

A continuación se va a realizar un seguimiento a nivel de código de este patrón. Los archivos necesarios son:

- *Layout de elemento.*
- *Contenedor de la lista de elementos.*
- *Clase ViewHolder.*
- *Adaptador entre interfaz y capa lógica.*

En primer lugar, ha de existir una plantilla referente que es sobre la que se van a referenciar los atributos de los objetos. En formato XML.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent">


    <android.support.v7.widget.CardView xmlns:card_view="http://schemas.android.com/apk/res-auto"
        android:id="@+id/card_view_paylog"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginBottom="8dp"
        android:layout_marginLeft="8dp"
        android:layout_marginRight="8dp"
        card_view:cardCornerRadius="4dp"
        card_view:cardElevation="4dp"
        card_view:cardBackgroundColor="@color/dark_background"
        android:layout_marginTop="6dp">

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="horizontal">

            <RelativeLayout
                android:layout_width="5dp"
                android:layout_height="match_parent">
```

En segundo lugar, se necesita un contenedor a modo de interfaz que es la que contiene la lista de elementos gráficos. Este contenedor lo hemos definido como RecyclerView en formato XML también.

```
<android.support.v7.widget.RecyclerView
    android:layout_width="match_parent"
    android:layout_height="480dp"
    android:id="@+id/paylog_fragment_layout" />
```

En adición, se necesita la clase ViewHolder que se encarga de realizar un primer acercamiento entre la capa lógica y la interfaz. Como podemos observar contiene tantos atributos como elementos deseamos que se modifiquen dinámicamente en la vista.

```
/********
 * Clase View Holder
 **********/


public static class PagoViewHolder extends RecyclerView.ViewHolder {
    CardView cv;
    TextView paylogTitle;
    TextView quant;
    Button more;
    ImageView status;



    PagoViewHolder (View itemView) {
        super(itemView);
        cv = (CardView)itemView.findViewById(R.id.card_view_paylog);
        paylogTitle = (TextView)itemView.findViewById(R.id.paylog_title);
        quant = (TextView)itemView.findViewById(R.id.paylog_quant);
        more = (Button) itemView.findViewById(R.id.read_more_pay);
        status = (ImageView) itemView.findViewById(R.id.status);
    }

}
```

Y por último, donde el patrón adquiere utilidad, es en la clase Adaptador donde existen dos formas distintas de construir los objetos gráficos:

1. Crear un elemento desde cero si no existen cartas liberadas para ser modificadas.

```
@Override
public PagoViewHolder onCreateViewHolder(ViewGroup viewGroup, int i) {
    View v = LayoutInflater.from(viewGroup.getContext()).inflate(R.layout.item_paylog, viewGroup, false);
    return new PagoViewHolder(v);
}
```

2. Modificar uno existente que está en desuso a nivel de visualización.

```
@Override
public void onBindViewHolder(PagoViewHolder pagoViewHolder, int i) {

    pagoViewHolder.paylogTitle.setText(pagos.get(i).titulo);
    pagoViewHolder.quant.setText(pagos.get(i).cantidad);

    //Color según el estado del pago
    if(pagos.get(i).pagado == true) {
        pagoViewHolder.status.setBackgroundResource(R.color.green);
    }else {
        pagoViewHolder.status.setBackgroundResource(R.color.red);
    }
```

En la segunda forma de construcción es donde se consigue esa reducción de consumo de recursos que se comentaba anteriormente, reutilizando los espacios en memoria de las variables y cambiando un proceso de creación de objetos por uno de modificación de variables.

4.4 Patrón de Fragmentos

En la versión 3.0 (API 11) de Android apareció la necesidad de programar para diferentes tamaños de pantallas ya que cada vez más los tamaños de los móviles en el mercado eran más dispares y además aparecían dos tipos de orientación de visualización: horizontal (landscape) y vertical (portrait). Para ello se insertaron los denominados fragmentos.

"Un fragmento es una sección "modular" de interfaz de usuario embebida dentro de una actividad anfitriona, el cual permite versatilidad y optimización de diseño. Se trata de miniactividades contenidas dentro de una actividad anfitriona, manejando su propio diseño (un recurso layout propio) y ciclo de vida"

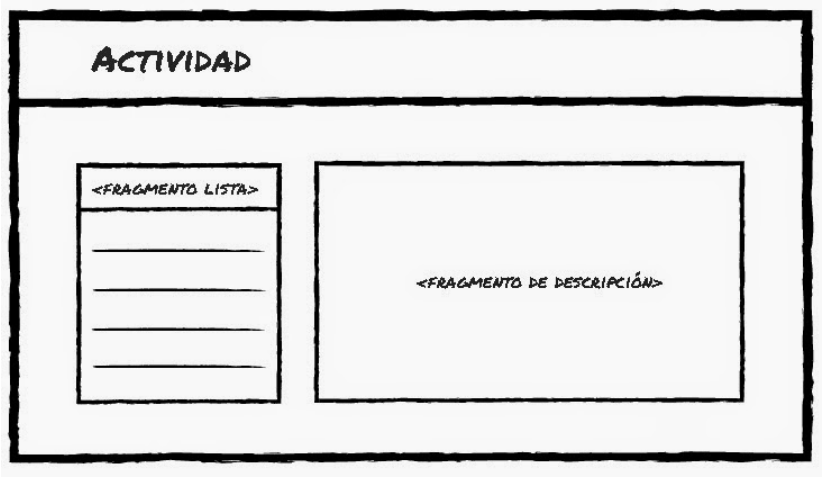


Ilustración 19 Reperesentación de fragmentos

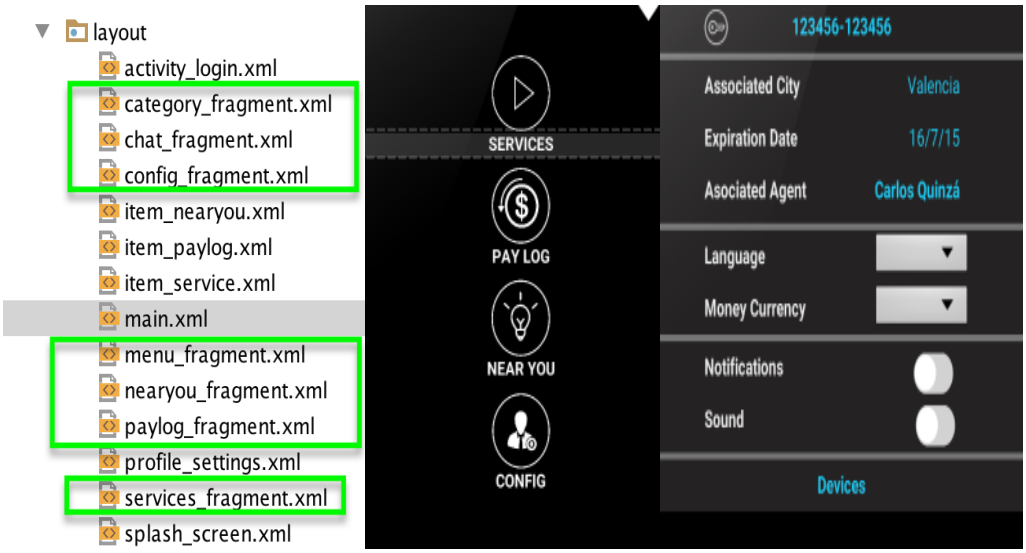
Aunque no se ha elegido este patrón en el proyecto por su facilidad de programar la interfaz para las diferentes pantallas sino por el principio de modularidad (diversas secciones de código) , en un futuro sería más fácil adaptar la aplicación para diferentes tamaños de visualización.

En concreto, nuestra interfaz general (sin contar la pantalla de "splash" y "login") se compone de dos elementos primitivos:

1. Main XML (Ocupa la parte estática en la interfaz mientras se cambia de menú o apartado. Está compuesto por Título, chat y, cuando estamos dentro de un apartado, el botón atrás para retornar al menú)

Ilustración 20 Ventana main.xml

2. Fragmento variable (Referido a la parte de la pantalla que se modifica según en qué estado de la aplicación estemos. P.e. Menú, Paylog, Near You, Config)



Las ventajas que se ofrecen en nuestro caso en particular son:

- En comparación con los Activity, la transición entre apartados de la aplicación es instantánea sin cortes en la pantalla

- El consumo de recursos entre apartados es menor, ya que solamente se recarga una pequeña parte en vez de toda la pantalla.
- Cada fragmento tiene un ciclo de vida independiente lo cual permite llevar un control más concreto de cada uno de ellos.

Implementación a nivel de código:

Clase main: Es la clase que contiene el layout main.xml explicado anteriormente y además guarda una referencia al fragmento que se va a usar en cada caso (un Fragcontainer)

```
boolean closed = true;  //Variable que captura el estado del chat (abierto o cerrado)
public static int Fragcontainer = R.id.content; //Contenedor principal que se rellena según el menu
public static FragmentManager fragmentManager;
```

Clase menu_fragment: Una vez iniciada sesión en la aplicación, este es el primer fragmento que se visualiza. En función del botón seleccionado se sustituirá por otro.

```
@Override
public void onClick(View v) {

    FragmentTransaction fragmentTransaction = main.fragmentManager.beginTransaction();

    switch (v.getId()) {

        case R.id.Menu1:
            Fragment services = new services_fragment();
            Fragment categories = new category_fragment();

            //fragmentTransaction.setCustomAnimations(R.anim.fade_in,0,0, R.anim.fade_out);
            fragmentTransaction.replace(Fragcontainer, services);
            //fragmentTransaction.setCustomAnimations(R.anim.enter_from_right, 0, 0, 0);
            fragmentTransaction.add(Catcontainer, categories);

            fragmentTransaction.addToBackStack(null);
            fragmentTransaction.commit();

            eventCenter.submenuCreated();

            break;
```

```
case R.id.Menu3:
    Fragment nearyou = new nearyou_fragment();

    fragmentTransaction = main.fragmentManager.beginTransaction();
    //fragmentTransaction.setCustomAnimations(R.anim.fade_in, 0, 0, R.anim.fade_out);
    fragmentTransaction.replace(Fragcontainer, nearyou);

    fragmentTransaction.addToBackStack(null);
    fragmentTransaction.commit();

    eventCenter.submenuCreated();

    break;
```

Es importante disponer de un Fragment Manager, que es el que controlará el ciclo de vida de todo fragmento activo en ese momento. En caso contrario, solamente

podríamos mostrar y ocultar fragmentos teniendo un consumo de memoria dinámica muy elevado.

Clase fragmento concreto: Es la clase Java que controla todo lo que ocurre dentro del fragmento en concreto.

```
public class config_fragment extends Fragment  {

    Cliente cliente;

    public config_fragment() {
        // Empty constructor required for fragment subclasses
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {
        View rootView = inflater.inflate(R.layout.config_fragment, container, false);

            cliente = cliente.getInstance();

            TextView keyText =(TextView) rootView.findViewById(R.id.keyText);
            keyText.setText(cliente.getClave());

            TextView cityText = (TextView) rootView.findViewById(R.id.cityText);
            cityText.setText(cliente.getCiudad());

            TextView dateText = (TextView) rootView.findViewById(R.id.dateText);
            dateText.setText(cliente.getFechaFin());

            TextView agentText = (TextView) rootView.findViewById(R.id.agentText);
            agentText.setText(cliente.getAgente().getName());

        return rootView;
    }
```

Como puede observarse en la ilustración superior, se referencia al *Layout* de configuración y contiene una variable por cada tipo de elemento de interfaz que se desea modificar dinámicamente. Es como un *Activity* convencional.

4.5 Programación para tamaños de pantalla distintos

"Desde la versión 1.6, Android admite varias densidades y resoluciones de pantalla. Los desarrolladores pueden decidir si las aplicaciones aparecerán o no en los dispositivos de cada grupo y controlar su apariencia a través de una serie de herramientas incluidas en el SDK y en las API del framework de Android. " – Soporte de Google.

Al contrario que Apple, el conjunto de dispositivos que utilizan Android tienen una diversidad de dimensiones de pantalla entre ellos que provoca cierto problema a la hora de diseñar una interfaz única (con sus iconos e imágenes). Por ello, tal como enunció Google, a partir de la versión 1.6 aparecieron unas herramientas para eliminar este inconveniente.
En nuestro caso, nos centraremos en la categorización de pantallas diferenciadas por una serie de medidas estándar. Esto, hace posible crear imágenes para pantallas distintas y que automáticamente Android decida según el dispositivo cuales importar.

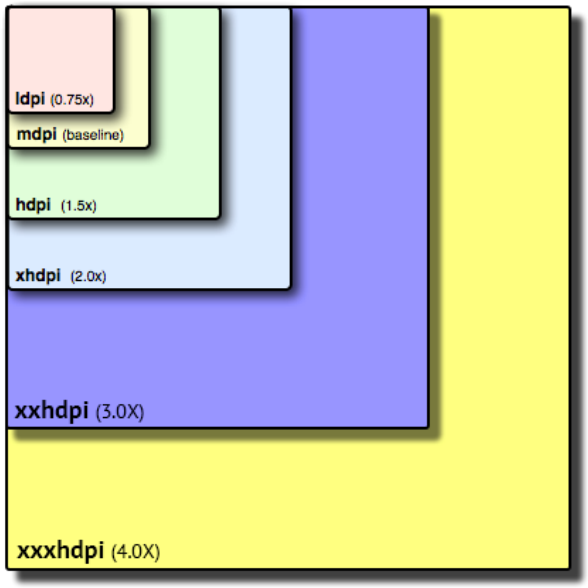


Ilustración 21 Equivalencias entre distintos tamaños de pantallas

A la hora de desarrollar esta ilustración es muy importante, pues sabemos que un botón o icono que queramos mostrar en xhdpi (pantalla extra larga) debe de ser el doble de grande que el que mostraremos en una pantalla mediana (mdpi). Pero, ¿Cuál es la clasificación que establece Android sobre que tamaños equivalen a cada categoría?

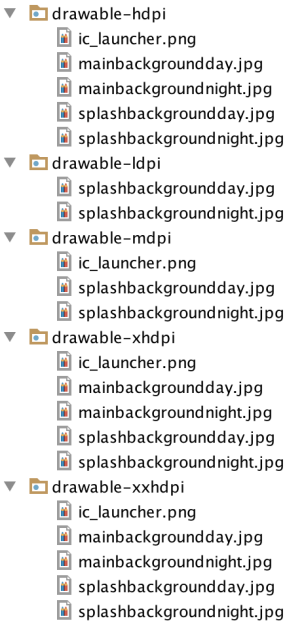
	Low density (120), *ldpi*	Medium density (160), *mdpi*	High density (240), *hdpi*	Extra high density (320), *xhdpi*
Small screen	**QVGA (240x320)**		480x640	
Normal screen	**WQVGA400 (240x400)** **WQVGA432 (240x432)**	**HVGA (320x480)**	**WVGA800 (480x800)** **WVGA854 (480x854)** 600x1024	640x960
Large screen	**WVGA800** (480x800)** **WVGA854** (480x854)**	**WVGA800* (480x800)** **WVGA854* (480x854)** 600x1024		
Extra Large screen	1024x600	**WXGA (1280x800)**† 1024x768 1280x768	1536x1152 1920x1152 1920x1200	2048x1536 2560x1536 2560x1600

* To emulate this configuration, specify a custom density of 160 when creating an AVD that uses a WVGA800 or WVGA854 skin.
** To emulate this configuration, specify a custom density of 120 when creating an AVD that uses a WVGA800 or WVGA854 skin.
† This skin is available with the Android 3.0 platform

Ilustración 22 Categoría en función de tamaño de pantalla y densidad Fuente: Stackoverflow)

Como ya hemos comentado anteriormente, para que Android trabaje eficientemente en este aspecto, basta con usar las carpetas del proyecto Android Studio (drawable/mipmap – ldpi/mdpi/hdpi/xhdpi/xxhdpi/xxxhdpi) con archivos llamados exactamente igual. El sistema se encargará de acceder a la carpeta correcta en función del dispositivo.

En nuestro caso, se ha hecho uso para los fondos de pantalla ya que son los recursos que se pixelaban ó deformaban de forma drástica usando solo un archivo de tamaño predeterminado al cambiar de dispositivo con una resolución diferente. Por lo tanto, nuestro proyecto se ve de la siguiente manera:



En los iconos no ha hecho falta realizar esta técnica, ya que se han creado de forma vectorial en el programa "Adobe Illustrator (Explicado en el apartado 2 "Plataformas utilizadas") en vez de en formato de pixeles (como ocurriría con el editor de imágenes Adobe Photoshop).

5. Trabajando con base de datos

Una de las etapas finales del proyecto ha sido dejar de trabajar con datos locales (almacenados en memoria) y proceder a guardar la información dinámica en internet para posteriormente recibirla en el dispositivo. Esta acción da ventajas importantes, dota de funcionalidad con agentes externos en vez de ser simplemente una aplicación de consulta.

Imaginemos, por ejemplo, que se añaden servicios en una ciudad determinada. Para que los usuarios de la aplicación viesen los cambios deberían actualizar (esto implica volver a descargar) toda la aplicación otra vez. Lo mismo ocurriría con los eventos y los pagos. Este proyecto necesita conexión a la nube de forma obligatoria.

5.1 Diseñando la estructura

El primer paso es crearse un esquema sobre las tablas que necesitamos, sus columnas y las relaciones entre ellas. Se puede obviar este paso, y comenzar directamente con la escritura del código SQL, pero, en el momento en el que haya que hacer alguna modificación, o que otra persona trate de entender esta estructura, se perderá mucho más tiempo que si tenemos un diagrama y trabajamos sobre él.

MySQL Workbench (explicado en la sección 3), permite crear un esquema de base de datos y generar el código SQL asociado para construir en cualquier servidor esta persistencia. Es importante revisar siempre las claves ajenas, claves primarias y tipos de datos de cada columna. Puede ahorrar tiempo a la larga, evitando fallos, y ayuda a afianzar la lógica de tu sistema.

5.2 Generando código SQL e insertándolo en nuestra Base de datos

Una vez tenemos el código SQL generado, se debe de contar con un servidor donde poder almacenar nuestra Base de datos. En el caso de este proyecto, se ha tenido el privilegio de dotar de un servidor prestado por una empresa privada, pero con una búsqueda simple en el buscador “Google” hay infinidad de empresas que prestan los servicios necesarios.

El código SQL de la base de datos de “Rivence” tiene este aspecto:

```
CREATE TABLE IF NOT EXISTS `qvx920`.`personas` (
  `idPersona` INT NOT NULL AUTO_INCREMENT,
  `nombre` VARCHAR(45) NOT NULL,
  `email` INT NOT NULL,
  `dni` VARCHAR(15) NOT NULL,
  `cuenta` VARCHAR(20) NOT NULL,
  `clave` VARCHAR(20) NOT NULL,
  `idioma` VARCHAR(20) NOT NULL,
  `ciudad` VARCHAR(45) NOT NULL,
  `avatar` VARCHAR(50) NULL,
  PRIMARY KEY (`idPersona`),
  UNIQUE INDEX `DNI_UNIQUE` (`dni` ASC),
  UNIQUE INDEX `Clave_UNIQUE` (`clave` ASC),
  UNIQUE INDEX `idPersona_UNIQUE` (`idPersona` ASC))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `qvx920`.`agentes`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `qvx920`.`agentes` (
  `idAgente` INT NOT NULL,
  `persona_idPersona` INT NOT NULL,
  PRIMARY KEY (`idAgente`, `persona_idPersona`),
  UNIQUE INDEX `idagente_UNIQUE` (`idAgente` ASC),
  INDEX `fk_agente_persona1_idx` (`persona_idPersona` ASC),
  CONSTRAINT `fk_agente_persona1`
    FOREIGN KEY (`persona_idPersona`)
    REFERENCES `qvx920`.`personas` (`idPersona`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `qvx920`.`usuarios`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `qvx920`.`usuarios` (
  `idUsuario` INT NOT NULL,
  `fechaInicio` DATETIME NOT NULL,
  `fechaFin` DATETIME NOT NULL,
  `agente_idAgente` INT NOT NULL,
  `persona_idPersona` INT NOT NULL,
  `persona_agente_idAgente` INT NOT NULL,
  PRIMARY KEY (`idUsuario`, `agente_idAgente`, `persona_idPersona`,
`persona_agente_idAgente`),
```

```
  UNIQUE INDEX `idUsuario_UNIQUE` (`idUsuario` ASC),
  INDEX `fk_usuario_agente_idx` (`agente_idAgente` ASC),
  INDEX `fk_usuario_persona1_idx` (`persona_idPersona` ASC,
`persona_agente_idAgente` ASC),
  CONSTRAINT `fk_usuario_agente`
    FOREIGN KEY (`agente_idAgente`)
    REFERENCES `qvx920`.`agentes` (`idAgente`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `fk_usuario_persona1`
    FOREIGN KEY (`persona_idPersona`)
    REFERENCES `qvx920`.`personas` (`idPersona`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `qvx920`.`servicios`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `qvx920`.`servicios` (
  `idServicio` INT NOT NULL,
  `titulo` VARCHAR(45) NOT NULL,
  `subtitulo` VARCHAR(45) NOT NULL,
  `distancia` INT NOT NULL,
  `descripcion` VARCHAR(45) NOT NULL,
  `idCategoria` VARCHAR(45) NOT NULL,
  `ciudad` VARCHAR(45) NOT NULL,
  `fechaCreacion` DATETIME NULL,
  PRIMARY KEY (`idServicio`))
ENGINE = InnoDB;
```

Hay dos opciones: Conectarse al servidor mediante una plataforma interna del servidor o seguir con el programa MySQL Workbench gestionando la construcción desde ahí. En nuestro caso se ha seguido utilizando la herramienta externa, ya que como se ha tratado con ella en asignaturas del Grado, era más rápido e intuitivo.

5.3 Rellenado de batería de datos

Hay que dotar de información a la estructura vacía de base de datos que tenemos, para posteriormente procesarla en la aplicación. Para el rellenado de datos hemos implementado una aplicación web que, mediante HTML, CSS y PHP, ofrece una interfaz agradable, una funcionalidad que cualquier persona sin saber programación puede usar y que permite insertar datos en cualquiera de las tablas. Este sistema podría denominarse "Panel de Administración".

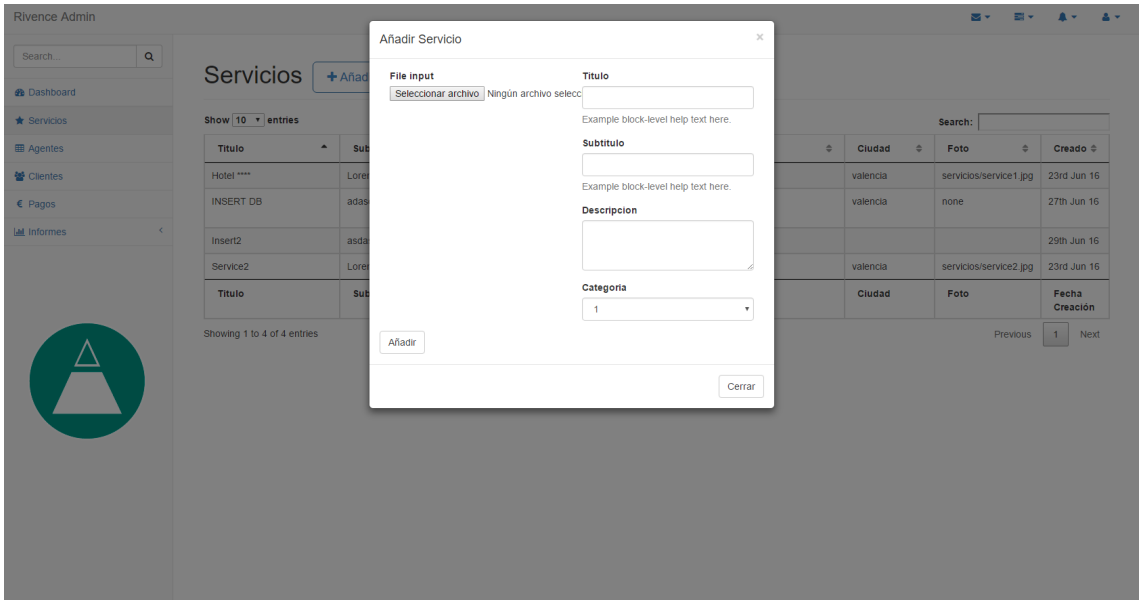


Ilustración 23 Inserción de Servicio en el Panel de Administración

Como se puede observar, nos permite añadir Servicios, Pagos, Clientes, Agentes, etc. En definitiva los datos necesario para que el sistema funcione correctamente.

Se necesita usuario y contraseña para poder acceder, por lo que solamente se comparten capturas de imagen de este recurso, para evitar que cualquier persona pueda alterar la base de datos.

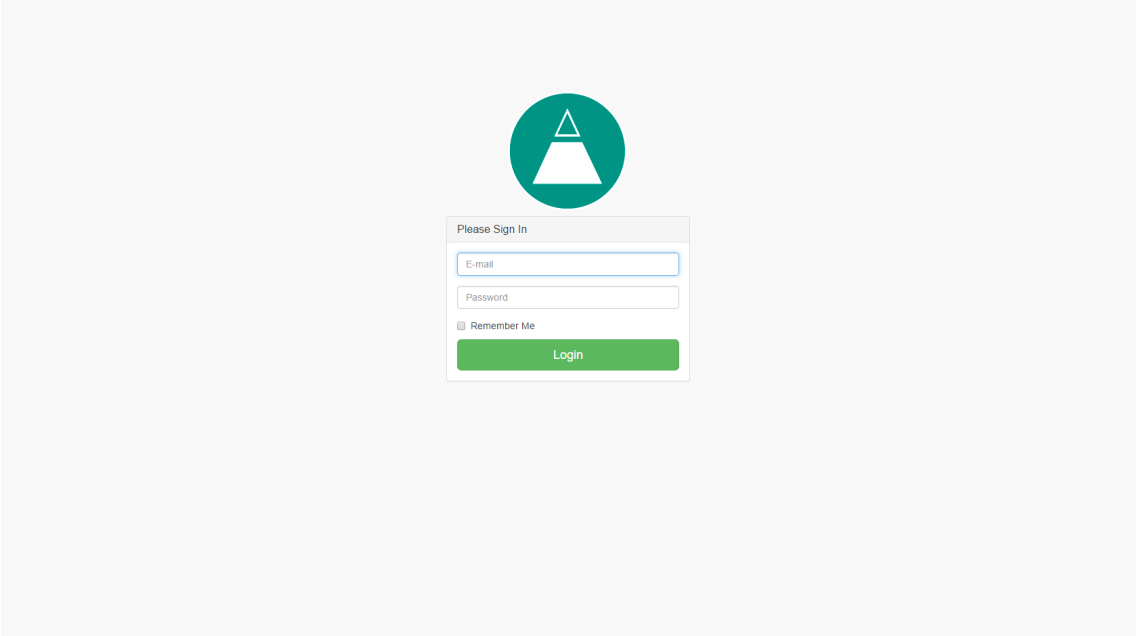


Ilustración 24 Acceso Panel de Administración

5.4 Creando puerta de acceso en la Base de datos

Una vez ya tenemos preparados los puntos anteriores, con el objetivo de proporcionar una puerta de acceso estándar, independiente de qué tipo de sistema está accediendo a la base de datos, se ha creado un servicio implementado en PHP. Con esta estructura, el código Java de la aplicación solamente tiene que hacer una consulta a la capa de persistencia pasándole un parámetro que determina que consulta ha de hacerse en la Base de datos, evitando así consultas SQL dentro del código.

¿Cómo se estructura este servicio PHP del que estamos hablando? Muy fácil, en primer lugar, existe un documento index.php, disponible en el repositorio en GitHub del proyecto), que hace función de diccionario, es decir, selecciona qué consulta hacer y devuelve información a la aplicación, con un "switch" del tag (nombre descriptivo de lo que se necesita) que envía la aplicación.

```
switch($tag){
        case 'login':
        $user= NULL;
        $key = $_POST['key'];
```

```
        // comprobamos el usuario
        $user = $db->GetUserByKey($key);

        if ($user != NULL) {

            echo json_encode($user) ;
        } else {

            echo $user ;
        }
            break;


            case 'pagos':
            $cliente = $_POST['cliente'];
            $pagos = $db->GetPagosbyCliente($cliente);
            echo json_encode($pagos);
            break;

            case 'servicios':
            $ciudad = $_POST['ciudad'];

            $servicios = $db->GetServiciosByCiudad($ciudad);
            echo json_encode($servicios);
            break;
case 'eventos':
            $ciudad = $_POST['ciudad'];

            $eventos = $db->GetEventosByCiudad($ciudad);
            echo json_encode($eventos);
            break;

            }
```

A continuación procedemos a explicar el código:

Como habíamos comentado existe un *switch($tag)* donde $tag lo envía la aplicación al iniciar la consulta y es lo que identifica que necesita. Los *case* son: login, pagos, servicios y eventos. Cada uno de estos tiene tres partes definidas:

1. Almacenar parámetros adicionales pasados por la aplicación
   [$cliente = $_POST['cliente'];]

2. Llamada a la consulta en otro archivo y almacenado de la respuesta en variable
   [$pagos = $db->GetPagosbyCliente($cliente);]

3. Envío en formato JSON a la aplicación el resultado de la consulta.
   [echo json_encode($pagos);]

Ahora mismo, nos centramos en el punto 2, la llamada por método a otro archivo donde están las consultas.

```
public function GetPagosByCliente($cliente){
$return_arr = array();
$result = mysql_query("SELECT `titulo`,`cantidad`,`pagado` FROM `pagos` WHERE usuarios_persona_idPersona = $cliente")or die(mysql_error());

while ($row = mysql_fetch_array($result, MYSQL_ASSOC)) {


   $row_array['titulo'] = $row['titulo'];
   $row_array['cantidad'] = $row['cantidad'];
$row_array['pagado'] = $row['pagado'];

   array_push($return_arr,$row_array);
}

return $return_arr;

}
```

Aquí se muestra uno de los métodos (GetPagosByCliente), que como el nombre indica obtiene los pagos relacionados a un cliente, el que usa la aplicación.

- $return_arr = Array donde se rellenara como JSON, el valor que se devuelve

- $result = Devolución de la consulta SQL que incluye

- Bucle while = Recorre todas las líneas rellenando el array JSON por cada uno de los pagos que existen para el $cliente.

Una vez que esto funciona correctamente, solamente falta trabajar sobre el código Java de la aplicación para realizar las consultas con el parámetro mencionado anteriormente.

5.5 Consultas a Base de datos desde JAVA

En este ámbito, hay varios estándares y librerías para realizar consultas. En nuestro caso hemos usado la librería Volley.

***"Volley** es una librería desarrollada por Google para optimizar el envío de **peticiones http** desde las aplicaciones Android hacia servidores externos. Este componente actúa como una interfaz de alto nivel, liberando al programador de la administración de hilos y procesos tediosos de parsing, para permitir publicar fácilmente resultados en el hilo principal."* -
HermosaProgramacion.com (02-2015)

En esta definición se menciona el por qué se ha elegido esta opción. Libera al programador de la administración de hilos y procesos tediosos de parsing (transformar la respuesta del servidor para procesarlo desde código Java). Ya que el conocimiento de este sector por parte de los desarrolladores de este proyecto es escaso, viene perfecto para no perder tiempo en aprendizaje sobre consultas a servidores.

En el proyecto, se han creado dos clases relacionadas con las consultas, presentes en el paquete "persistencia"). Estas son:
/persistencia/MySocialMediaSingleton.java
/persistencia/MySocialMediaRequests.java

MySocialMediaSingleton : El objetivo es tener solamente una cola de peticiones, que, dentro de su buffer, se organizan en función de prioridades.
Esta clase tiene un constructor, y proporciona métodos de adición de peticiones a la cola *"**public void** addToRequestQueue(Request req) { getRequestQueue().add(req); }"*, construcción del objeto (privado) y getInstance() de acuerdo al patrón Singleton. (Similar a lo que hemos comentado en el apartado 4.2)

```
public static synchronized MySocialMediaSingleton getInstance(Context
context) {
   if (singleton == null) {
      singleton = new MySocialMediaSingleton(context);
   }
   return singleton;
}
```

MySocialMediaRequests : Esta clase se ha creado con el fin de tener todas las consultas reunidas en una misma clase, por consecuencia, esta clase debe tener implementado el patrón Singleton. Así, si hay que modificar algún tipo de consulta, sabemos que ha de acceder a esta apartado. Toda consulta tiene aquí dos métodos:
Nota: (XXXX = Pago, Servicio o Evento)

- **public void** initializeXXX()
- **public** List<XXXX> parseJsonXXXX(String entrada) **throws** JSONException .

Ejemplo en "Pagos":

```
public void initializePagos() {

    StringRequest jsArrayRequest = new StringRequest(
            Request.Method.POST,
            Config.LOGIN_URL,
            new Response.Listener<String>() {
                @Override
                public void onResponse(String response) {
                    Log.d(TAG, "Respuesta Volley:" +
response.toString());
                    try {

Cliente.getInstance().setPagos(parseJsonPagos(response));
                    } catch (JSONException e) {
                        e.printStackTrace();
                    }
                }


            },

            new Response.ErrorListener() {

                @Override
                public void onErrorResponse(VolleyError error) {
                    Log.d(TAG, "Error:" + error.getMessage());

                }
            }) {
        @Override
        protected Map<String, String> getParams() throws
AuthFailureError {
            Map<String, String> params = new HashMap<>();
            //Adding parameters to request
            params.put(Config.KEY_TAG, "pagos");
            params.put("cliente",Cliente.getInstance().getId());


            //returning parameter
            return params;
        }

        @Override
        public Map<String, String> getHeaders() throws
AuthFailureError {
            HashMap<String, String> headers = new HashMap<String,
String>();
            headers.put("Content-Type", "application/x-www-form-
urlencoded; charset=utf-8");
            headers.put("User-agent", "My useragent");
            return headers;
```

```
        }

    };

    // Añadir petición a la cola

MySocialMediaSingleton.getInstance(getContext()).addToRequestQueue(jsA
rrayRequest);


}
```

InitializePagos() es el encargado de crear la petición, enviando los parámetros necesarios:

```
“params.put("cliente",Cliente.getInstance().getId());
      params.put(Config.KEY_TAG, "pagos");”
```

Esto es, se envía la identificación del cliente, para poder obtener sus pagos asociados y también el TAG “pagos” para que entre en el *case* ‘pagos’ respectivo en el diccionario PHP mencionado en el anterior apartado. Y añadiendo la petición a la cola:

```
MySocialMediaSingleton.getInstance(getContext()).addToRequestQueue(jsA
rrayRequest);
```

Pero, hemos obviado un paso, que es realmente importante, ¿Qué se hace con la respuesta del servidor? Aquí entra en juego el método **parseJsonPagos(String entrada)**

```
public List<Pago> parseJsonPagos(String entrada) throws JSONException
{
    /*  Variables locales  */
    List<Pago> pagosL = new ArrayList();
    JSONArray jsonArray = null;

    // Obtener el array del objeto
    jsonArray = new JSONArray(entrada);

    for (int x = 0; x < jsonArray.length(); x++) {

        try {
            JSONObject objeto = jsonArray.getJSONObject(x);
            boolean pagado = false;
            String titulo = objeto.getString("titulo");
            int cantidad = objeto.getInt("cantidad");
            int pagadoInt = objeto.getInt("pagado");
            if (pagadoInt == 1) pagado = true;
            else pagado = false;

            Pago pago = new Pago(titulo, cantidad, pagado);


            pagosL.add(pago);
        } catch (JSONException e) {
            Log.e(TAG, "Error de parsing: " + e.getMessage());
        }
```

```
    }

    return pagosL;
}
```

Este método se encarga de recibir el String (en formato Array de JSON), hacer las acciones oportunas con él (en este caso crear todos los objetos Pago que manda el servidor) y devolver la lista de Pagos entera para que el método que lo llama almacene en el Cliente esta lista.

Con todo este proceso, la aplicación es funcional y correcta, pero, aún falta hacer la carga de imágenes. En el caso concreto de pagos no existe porque ningún pago tiene asociado a una imagen. Vamos a explicar la carga de imagen del avatar del Agente, que es más simple y extrapolable al caso de Servicios y Eventos, que también tienen.

El apartado de carga de las imágenes, es un punto muy importante en el ahorro de consumo de memoria ya que éstas ocupan gran tamaño en comparación con el texto, y el hecho de almacenarlas en la nube, provoca una reducción considerable. Esta parte, es fácil de implementar. Con unas pocas líneas se obtiene la imagen y se utiliza mientras que Volley por dentro trata el almacenamiento eficiente en memoria caché. Veamos el código de la consulta para la imagen del Agente:

```
// Obtener el image loader
ImageLoader imageLoader=
MySocialMediaSingleton.getInstance(getActivity().getApplicationContext
()).getImageLoader();
// Petición
imageLoader.get(Config.IMAGE_URL + Agente.getInstance().getFotoURL(),
ImageLoader.getImageListener(avatar,
        R.drawable.loading, R.drawable.error));
```

En primera instancia, ImageLoader es una clase que aporta la librería Volley, simplemente tenemos que obtener el objeto donde tenemos la creación de este objeto (Clase MySocialMediaSingleton) para después pasarle en el imageLoader.get() los siguientes parámetros:

- **Config.IMAGE_URL :** Dirección absoluta donde se almacenan las imágenes. Por ejemplo: www.rivence.com/images/

- **Agente.getInstance().getFotoURL():** Dirección relativa del recurso. En nuestro caso, esta variable se almacena cuando se hace la primera consulta del agente como String. P.e. agentePepe.jpg

- **ImageLoader.getImageListener(avatar** (elemento de interfaz donde ha de introducirse la imagen)**,R.drawable.loading**(Recurso de imagen que se utiliza mientras se descarga la imagen del servidor)**, R.drawable.error**(Recurso de imagen que se utiliza si hay error de descarga)**));**

6. Técnicas de refactorización y mantenimiento

En este apartado, se enuncian todas las técnicas utilizadas relacionadas con: el aumento de la mantenibilidad, la reducción de la memoria dinámica y la ayuda de la comprensión de la estructura de la aplicación. Todas estas acciones, tienen como objetivo favorecer el posterior mantenimiento y escalado del sistema.

Nota: Dentro de los documentos adicionales del trabajo, se incluirá una carpeta denominada Mantenimiento con todos los documentos relacionados con este apartado.

6.1 Paleta general de colores

Al comienzo del desarrollo de la aplicación, en cada elemento de la interfaz se insertaban los colores en hexadecimal sin seguir ninguna regla. Entonces, surgían dos problemas básicos: En primer lugar, no había una sintonía entre elementos gráficos, es decir, cada texto de cuerpo tenía su propia tonalidad de blanco y los títulos, colores diferentes ente ellos, lo cual hacía difícil identificar qué era un título o un texto de cuerpo. En segundo lugar, cada vez que queríamos modificar un color de un elemento, se debía acceder al archivo XML que contiene el elemento con el color a cambiar y buscar un valor hexadecimal concreto.

Para solucionar estos problemas, se insertó una **paleta de colores**. Esto consiste básicamente en un archivo XML en el cual se realiza una serie de relaciones identificación (id) → valor hexadecimal.

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <!--Colores de textos-->
    <color name = "white">#eeeeee</color>
    <color name = "double_white">#ffffff</color>
    <color name = "light_grey">#979797</color>
    <color name = "dark_grey">#666666</color>
    <color name = "black">#979797</color>
    <color name = "greeny_blue">#0d9385</color>
    <color name = "light_blue">#2fcdf4</color>


    <!--Indicadores estatus paylog-->
    <color name = "red">#8c0a0a</color>
    <color name = "green">#0a8c33</color>

    <!--Fondos de layouts transparentes-->
    <color name = "dark_background">#bb312f2f</color>
    <color name = "light_background">#a4464646</color>
    <color name = "transparent">#00ffffff</color>

    <!--Fondos selector Paylog-->
    <color name = "paylog_pressed">#787877</color>
    <color name = "paylog_notpressed">#9f9f9e</color>

    <!--Colores categorias-->
    <color name = "cat1">#787877</color>
    <color name = "cat2">#787877</color>
    <color name = "cat3">#787877</color>
    <color name = "cat4">#787877</color>
    <color name = "cat5">#787877</color>
</resources>
```

Ilustración 25 XML "Paleta de colores"

Podemos observar que tenemos nuestros propios blancos, negros y transparencias de fondos entre otros elementos. Así, a la hora de querer modificar, por ejemplo, nuestro *azul verdoso* solamente debemos cambiar el hexadecimal asociado de "*greeny_blue*" en este documento y automáticamente todos los elementos que utilicen este color cambiarán al nuevo.

```xml
    <ImageView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/status"
        android:background="@color/red" />
</RelativeLayout>

<RelativeLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <TextView
        android:id="@+id/paylog_title"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textColor="@color/white"
        android:textSize="13sp"
        android:textStyle="bold"
        android:paddingStart="5dp"
        android:paddingEnd="5dp"
        android:text="Hotel las Arenas"
        android:layout_marginLeft="15dp"
        android:layout_marginStart="17dp"
        android:layout_centerVertical="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true" />
```

Ilustración 26 Ejemplo asignación de colores

Esto produce un ahorro de tiempo considerable en la fase de mantenimiento y una mayor organización del proyecto.

6.2 Nomenclaturas

Interfaces

Para los elementos de interfaz hemos usado cierta nomenclatura basándonos en categorías para poder distinguir que función tienen:

Fragmentos:

XXXX_fragment: Son como su nombre indica fragmentos (explicados en el apartado 3.3 Patrón Fragmentos)

Ítems:

XXXX_item: Contienen la estructura de ítems dentro de su respectivo RecyclerView. Es decir, cada elemento del listado que se muestra en (Servicios, Near You ó Paylog) deberá estar asociado a una estructura de estas.

Activities:

Activity_XXXX: Son las interfaces que equivalen a una actividad completa en el apartado lógico. Por ello se incluyen login, splash y main. Estas tres interfaces tienen asignada una *activity* para ellas.

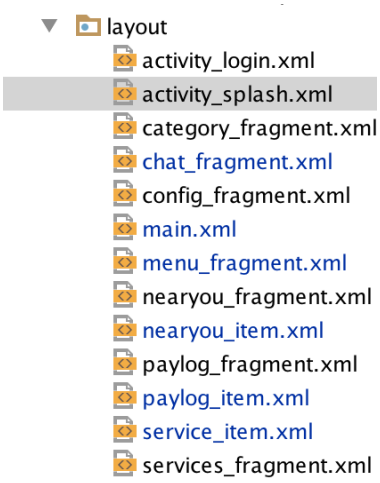


Ilustración 27 Visión del árbol de interfaces

Clases Controlador Java

Objetos

[Servicio / Agente / Pago / Agente / Cliente] En idioma castellano

Elementos de patrón ViewHolder

XXXXAdapter: Son las clases encargadas de unir la parte persistente de datos a la interfaz

XXXXFilterCards: Contiene lo lógica de filtrado del listado en el RecyclerView

XXXX = Nombre del menú al que se desea referir

Recursos (Carpeta Drawable)

Iconos

XXXX_ic : Todo icono usado en la interfaz se compone por su nombre acompañado de una parte estátic "_ic"

Imágenes de cada evento

XXXX_event: Las imágenes que se visualizan en los ítems del apartado Near you

Servicios

XXXX_service: Las imágenes que se visualizan en los ítems del apartado Servicios

6.3 Documentación Javadoc

“Javadoc es una utilidad de Oracle para la generación de documentación de APIs en formato HTML a partir de código fuente Java. Javadoc es el estándar de la industria para documentar clases de Java. La mayoría de los IDEs los generan automáticamente.” – Wikipedia [01/06/2016]

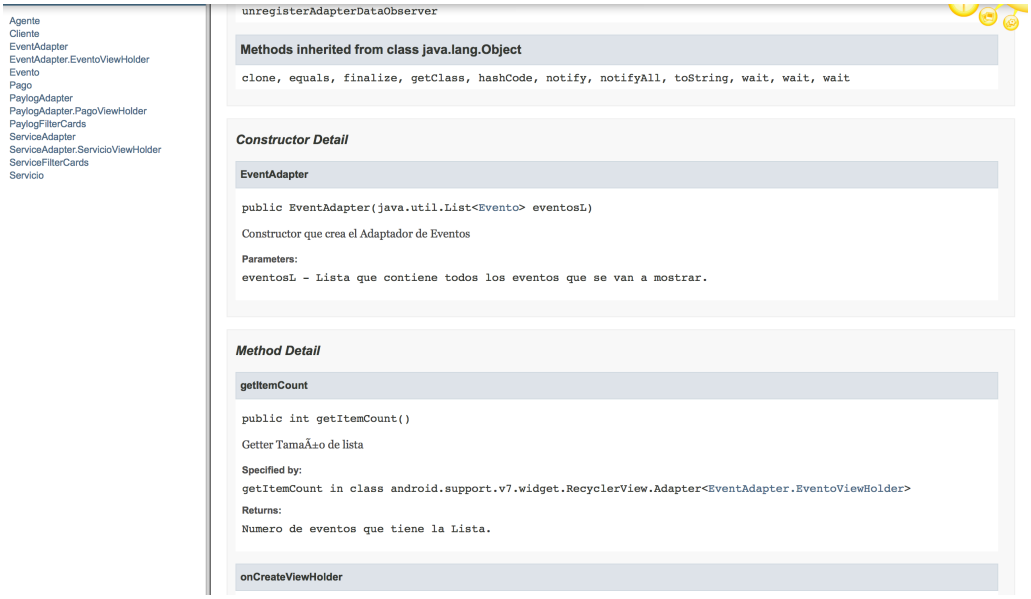


Ilustración 28 Captura Javadoc del Proyecto

Esta documentación ha sido útil para dos procesos destacados. En primer lugar, a la hora de revisar un elemento de código concreto (ya sea un método, un constructor o algo similar) el tiempo de refresco o aprendizaje ha sido notoriamente rápido gracias a las aclaraciones que se han escrito para esta documentación. Por ejemplo, documentando qué significa el valor de retorno de un método, qué son los parámetros que se reciben o una descripción general de qué hace un fragmento de código.
En segundo lugar, a la hora de explicar a otras personas la estructura de la lógica, la interfaz que propone Javadoc, simple, navegación entre clases, referencias entre métodos, ayuda a dar una idea más clara y estructurada del programa.

A pesar del tiempo que conlleva rellenar todas las descripciones de los métodos y las clases en el código, esta técnica (junto a comentarios concretos del código) proporciona, en un periodo de tiempo largo, una mejora en la mantenibilidad de cualquier programa y un incremento de la curva de aprendizaje del mismo.

```
/**
 * Constructor privado para evitar llamadas de clases
*externas
 *        Con        atributos        de        entrada
 *
 * @param city    Ciudad en la que se realiza el servicio.
 *  @param  endDate  Fecha de finalizacion del servicio
 *  @param  key        Clave de acceso a la aplicacion
 */
```

Ejemplo de generación de Javadoc

Nota: En los ficheros adicionales del proyecto, a parte del código del proyecto, se incluyen todos los documentos Javadoc generados.

7. Estado final de la aplicación

Para obtener todo código y documentos generados en la aplicación, se ha dispuesto de un repositorio público en GitHub.

Enlace al repositorio: https://github.com/carquipe/Rivence

Funcionalidades implementadas:

- **Sistema de usuarios**
  - Existen varios usuarios de prueba, cada uno con su clave privada y su información personalizada.
  - Claves de prueba: 487569585, 49588584

- **Interfaz completa**
  - Toda la interfaz está funcional y correctamente implementada, sin ningún problema y adaptada a todo tipo de pantallas

- **Base de datos**
  - La persistencia del proyecto está completamente funcional, y los datos existen en su completitud para los usuarios de prueba. Esto es: Pagos, Servicios, Eventos y Agente asociado a los clientes.-

- **Carga dinámica de datos**
  - Los campos de interfaz son dinámicos y varían en función de la persistencia. Por ejemplo, los campos que indican los títulos de los servicios, las descripciones, imágenes, etc.

- **Lógica funcional**
  - Toda la lógica tales como filtros, conversiones y cálculos que se realizan en el código Java funciona correctamente sin bugs detectados.

- **Documentación adicional**
  - Todos los documentos asociados al mantenimiento (Javadoc, herramientas de soporte y diagramas) están terminados y disponibles para el lector

- **Plataformas adicionales**
  - El panel de administración de la base de datos está funcional sin bugs visibles.

8. Futuras implementaciones

A pesar de que la aplicación funciona correctamente, existen ampliaciones aplicables para hacer de ésta un proyecto profesional.

- **Integración de pasarela de pago con un banco.**
  - En vez de gestionar los pagos de manera manual, se puede hacer que los pagos de cada cliente, se obtengan en función del banco desde el cual se le programan los pagos. Para ello, se necesita tener acceso al servicio de la entidad bancaria y tratar el paso de información. Así, se permite al Cliente pagar desde la aplicación, aportándole mayor comodidad.

- **Integración de chat entre Agente y Cliente.**
  - Debido a la complejidad de este concepto, no se ha introducido en el trabajo, pero, la interfaz de este está perfectamente implementada.

- **Conversión de divisa más real.**
  - Ahora mismo, en el menú configuración existe la opción de cambiar la divisa de la cantidad de los pagos. Ahora mismo el valor de conversión entre Euros y Dólares es estático. Como mejora, se propone acceder al valor de la divisa de una base de datos oficial en todo momento para hacer la conversión más fiable.

- **Mostrado de dispositivos conectados a la cuenta del cliente.**
  - En el apartado de configuración se ha reservado un apartado en el cual se puede ver desde que dispositivos se ha conectado el Cliente al servicio. Éste no está aún funcional, pero en un futuro sería correcto que funcionase para que el Cliente tenga una visión actualizada del servicio.

- **Notificaciones**
  - Hoy en día toda aplicación consistente entrega notificaciones en el dispositivo. En este caso, se debe aplicar en los casos: Cuando se añade un pago, cuando el Agente manda un mensaje al Cliente por el Chat y cuando un evento está próximo a ocurrir.

9. Conclusiones del proyecto

Ha sido un proyecto costoso respecto al gasto en horas, debido al escaso conocimiento de todas las materias explicadas y usadas. Nunca había desarrollado una aplicación para Android y algunas de las herramientas usadas las desconocía por completo.

Si se comparase el conocimiento de programación de aplicaciones para Android antes y después del trabajo, la diferencia sería significativa. Ha habido muchos problemas que se han planteado a lo largo del desarrollo y tras investigaciones propias (por Internet) las he resuelto. Dentro de estos problemas se incluyen la toma de decisiones sobre qué patrones utilizar, qué herramientas usar y bugs a nivel de código tanto estáticos como a nivel de ejecución.

Como aportaciones destacaría la visión del desarrollo de un proyecto, donde cobra importancia la constancia y organización estricta (ya que sin esta organización sería imposible acordarse de qué clase incluye las cosas a modificar, o al paso del tiempo recordar que cambios se habían realizado) a parte de, obviamente el desarrollo en sí de la aplicación.

Se han utilizado los lenguajes XML, Java, SQL, PHP, HTML y CSS, de los cuales, solamente en el Grado se ha visto el segundo mencionado. Por lo que, ha sido todo un reto aprender cada uno de ellos y poder extrapolarlo a nuestro proyecto.

El coste económico ha sido nulo, porque la gran cantidad de herramientas usadas han sido de licencia libre a excepción de Adobe Illustrator (del cual ya disponía licencia) y el servidor ha sido proporcionado por una empresa privada que ha tenido la amabilidad de prestarme un poco de su espacio contratado para almacenar los datos.

Respecto al tema de mantenimiento, a pesar de la constante enunciación del profesorado del Grado sobre la importancia de este aspecto, hasta ahora no me había dado cuenta de lo importante que es mantener un orden y escribir un código con comentarios para que cualquier persona e incluso uno mismo después de un tiempo pueda entender todo el código rápidamente sin dificultades.

10. Librerías usadas y agradecimientos

Librerías:

The Noun Project.com
Fuente de iconos de donde se han extraído todos los usados en la aplicación.

Unsplash.com
Fuente de imágenes de alta calidad usada para los fondos de la aplicación

Volley Android Studio Library
Librería de Android para realizar peticiones http.

AppCompat v7 Library
Librería para que las versiones antiguas de Android soporten el patrón Recycler View implementado.

Circular Progress View Library
Librería de Android que genera el "*spinner*" que aparece en la pantalla de carga.

InVisionApp.com[projects.invisionapp.com]
Herramienta Web para poder hacer simulación de las interfaces creadas.
El enlace corresponde al proyecto usado en este trabajo.

StarUML
Generación de diagramas UML a partir de código Java. En nuestro caso, diagrama de clases.

Agradecimientos:

StackOverflow Forum
Agradecimiento a la comunidad por su ayuda y respuestas respecto a las consultas expuestas en el foro

EuropaParts
Agradecimiento a esta empresa por prestarme espacio en su servidor para colocar la base de datos.

11. Bibliografía

En este apartado se incluyen los enlace más significativos que contienen la documentación aprendida y revisada para poder desarrollar la totalidad de la aplicación.

1. http://academiaandroid.com/android-studio-v1-caracteristicas-comparativa-eclipse/ (Fecha de consulta: 15-05-2016)

2. http://appdesignbook.com/es/contenidos/preparando-los-archivos/ (Fecha de consulta: 07-06-2016)

3. https://developer.android.com/training/material/lists-cards.html?hl=es (Fecha de consulta: 01-05-2016)

4. http://jarroba.com/programar-fragments-fragmentos-en-android/ (Fecha de consulta: 24 - 04 – 2016)

5. http://www.hermosaprogramacion.com/2014/09/android-aplicaciones-fragmento/ (Fecha de consulta: 25- 04 - 2016)

6. http://code.tutsplus.com/es/tutorials/getting-started-with-recyclerview-and-cardview-on-android--cms-23465 (Fecha de consulta: 25-05- 2016)

7. http://inducesmile.com/android/android-cardview-and-toolbar-with-material-design/ (Fecha de consulta 26 – 05 – 2016)

8. http://simpledeveloper.com/how-to-communicate-between-fragments-and-activities/ (Fecha de consulta 06-06-2016)

9. http://www.stackoverflow.com (Consultas periódicas de todo tipo)

10. http://www.travelopenapps.org/turismo-de-lujo-un-sector-en-auge/ (Fecha de consulta: 20-04-2016)