ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Proyecto Fin de Carrera

Zwait
Un gestor de retrasos para citas

AUTOR: Francisco Javier Sánchez Díaz
CODIRECTORES: D. Pedro Sánchez Palma, D. Juan Ángel Pastor Franco
Septiembre/ 2014

Autor	Francisco Javier Sánchez Díaz
E-mail del Autor	Sanchezdiazfranciscojavier@gmail.com
Codirectores	D. Pedro Sánchez Palma D. Juan Ángel Pastor Franco
E-mail del Director	pedro.sanchez@upct.es JuanAngel.Pastor@upct.es
Título de PFC	Zwait. Un gestor de retrasos para citas

Resumen
El proyecto se basará en el desarrollo de una página web y una aplicación para el sistema operativo Android bajo el nombre Zwait, que consistirá en un gestor de retrasos para citas.

Titulación	Ingeniería Técnica de Telecomunicación, especialidad en Telemática
Departamento	Lenguajes y Sistemas Informáticos
Fecha de Presentación	Septiembre 2014

Agradecimientos

A mis padres, Francisco Sánchez y Mª José Díaz que siempre han confiado en mi capacidad, más incluso que yo y nunca han dejado de apoyarme en mis estudios. A toda mi familia en general que siempre me ha dado ánimos para conseguir mis objetivos y siempre me han transmitido valores para intentar ser cada vez mejor persona. A mi novia y por supuesto a su familia por tratarme como a uno más de ellos haciéndome sentir como en casa. A mis tutores de proyecto Pedro Sánchez Palma y Juan Ángel Pastor Franco por el apoyo, la atención prestada y facilidades brindadas. A todos los compañeros y profesores que he tenido durante la carrera de los cuales he aprendido algo y a los cuales guardo un cariño especial. Y como no a mi compañero Francisco Alarcón con el cual he compartido numerosas tardes de estudio y siempre nos hemos intentado ayudar aportando ideas.

Índice

Índice de figuras y tablas

Capítulo 1.

Introducción

La telefonía móvil está cambiando la sociedad actual de una forma tan significativa como lo ha hecho Internet. Esta revolución no ha hecho más que empezar, los nuevos terminales ofrecen unas capacidades similares a un ordenador personal, lo que permite que puedan ser utilizados para leer nuestro correo o navegar por Internet. Pero a diferencia de un ordenador, un teléfono móvil siempre está en el bolsillo del usuario. Esto permite un nuevo abanico de aplicaciones mucho más cercanas al usuario. De hecho, muchos autores coinciden en que el nuevo ordenador personal del siglo veintiuno será un terminal móvil.

El lanzamiento de Android como nueva plataforma para el desarrollo de aplicaciones móviles ha causado una gran expectación y está teniendo una importante aceptación tanto por los usuarios como por la industria. En la actualidad se está convirtiendo en una seria alternativa frente a otras plataformas como Symbian, iPhone o Windows Phone. [1]

1.1 Motivación

¿Cuántas veces hemos acudido a un lugar en el que teníamos cita previa (medico en consulta privada, dentista, ortodoncista,…) y hemos tenido que esperar más de una hora para ser atendidos?

1.2 Descripción básica

Zwait es una idea innovadora de aplicación en movilidad que podría solucionar a corto plazo el problema de la esperas en citas previas, gracias al estado actual de la técnica y la gran disponibilidad de teléfonos inteligentes.

Zwait tiene dos modos de funcionamiento: como cliente/paciente y como empresa. Como cliente siempre es desde el teléfono móvil. Como empresa

puede ser desde ordenador de sobremesa vía Web o desde el teléfono móvil con Zwait.

Cuando el paciente pide cita y es la primera vez que usa Zwait para esa clínica (empresa) recibe un código de entidad junto con el código de la cita (un número de tres cifras puede ser suficiente para identificarla). Introduce ambos en la aplicación del móvil y da de alta a la clínica para catalogarla con una descripción. Las veces sucesivas que pida cita para dicha clínica sólo introducirá el código de la cita.

Cuando se acerque el momento de la cita, desde la clínica y usando Zwait (vía Web o el propio móvil), se anotará el retraso que está sufriendo la cita concertada aportando así una estimación mínima de hora a la que será atendido.

El paciente recibe en el móvil una alerta notificándole el retraso. Haciendo uso de Zwait da acuse de recibo lo que permite a la clínica confirmar que el paciente ha tomado nota del mismo.

Se obtienen estadísticas de retrasos lo que puede redundar en una optimización del tiempo de cita gestionado por una clínica.

1.3 Objetivos del Proyecto

El nivel de la tecnología actual permite perfectamente el desarrollo de la aplicación en movilidad. Se requiere un servicio de almacenamiento en la nube para guardar en una base de datos las clínicas registradas, sus citas y las notificaciones. Esto podría ser proporcionado como un servicio a aquellos que usen Zwait como clínica/empresa obteniéndose de esta forma un retorno de la inversión realizada.

Podría discriminarse entre los usuarios para que en caso de usar la red de Telefónica se ofreciera el producto libre de publicidad y sin coste.

Con Zwait las clínicas y empresas pueden mejorar su imagen frente a sus clientes, ofreciendo el producto como una consideración a su tiempo a la vez que les permitiría ofrecer una imagen tecnológica moderna.

La utilidad de Zwait está garantizada, tal y como podemos analizar considerando únicamente los datos relativos a las clínicas dentales (uno de

los lugares donde los tiempos de espera suelen dispararse). Según los datos disponibles en la web del Consejo General de Dentistas de España en España hay 29730 Dentistas, y 17968 Clínicas Dentales. Obtenemos 2500 habitantes por clínica (1510 habitantes por dentista), para una población de 45.000.000 de habitantes. Considerando que un 55% de la población tiene un smartphone en España (datos del segundo semestre de 2012) las posibilidades de explotación de Zwait son enormes.

Capítulo 2.

Android

Android es un "sistema operativo" relativamente nuevo, elaborado por la compañía **Google**.

Se trata de una plataforma desarrollada para dispositivos móviles como "smartphones", "tables", etc., con la idea de proporcionar a los fabricantes un sistema flexible y actualizado con bajo coste.

Las aplicaciones para este sistema operativo están programadas sobre plataforma **JAVA.**

Una de las características más importantes que diferencia a **Android** de otros sistemas operativos para dispositivos móviles como **IOS** o **Windows Phone**, es que ha sido desarrollado en base a la filosofía de código abierto, dando como resultado que el programa evolucione y mejore con el paso del tiempo y las aportaciones de muchos usuarios. [1] y [2]

Figura 1: Logotipo Android

2.1 Historia de Android

En el año 2005 “Google” adquirió una empresa llamada “Android Inc.” que hacía software para móviles.

Posteriormente, en 2007, se creó un consorcio formado por varias compañías tecnológicas, como fabricantes de hardware, de dispositivos móviles, software, etc.., que recibió el nombre de “OPEN HANDSET ALLIANCE” (OHA), presentándose la plataforma Android para dispositivos móviles basada en la versión 2.6 del kernel de Linux, y comenzando la historia de este exitoso sistema operativo.

Cada nueva versión de Android recibe el nombre de un postre en inglés. Por ejemplo la versión 1.5 presentada en mayo de 2009 se llama “Cupcake”, la versión 1.6 se llama “Donut”, y así sucesivamente.

La evolución tan seguida de las versiones ha creado un problema llamado “defragmentación”, que consiste en que dispositivos antiguos no soporten las nuevas versiones. Por ello desde “Google” se ha solucionado este problema espaciando el lanzamiento de nuevas versiones.

A la fecha de realización de este proyecto, la última versión liberada de Android es la 4.1 Jelly Bean, que traducido al español sería algo como “gominola”.

La próxima versión será la 5 y se llamará “Key Lime Pie” (tarta de lima).

Las aplicaciones de Android se desarrollan en lenguaje “Java”, al que se le añade un paquete especial denominado “Android Software Development Kit”, “Android SDK”, provisto por “Google”, que permite adaptar la programación a las características del sistema operativo Android.

“Google Play” es una tienda virtual de software que permite a los clientes descargar aplicaciones tanto gratuitas como de pago, de igual modo que cualquier persona que desarrolle un programa para Android puede publicarlo en “Google Play”, y hacedlo accesible a millones de usuarios.

2.2 Android en la actualidad

2.2.1 Aplicaciones Android: un caso de éxito

La plataforma Android va ganando terreno y lo hace bastante rápido.

Sin duda, el crecimiento exponencial de Android ha ocurrido, en gran medida, por sus cuestiones comerciales más que de rendimiento o popularidad mediática. Y es que el hecho de que cualquier fabricante pueda tomar Android como Sistema Operativo base para sus equipos, convirtiéndolos así en Smartphones, ha llevado a una verdadera revolución en lo que a dispositivos móviles se refiere, con distintos modelos y equipos variados inundando los mercados internacionales cada semana.

Apple creó la primera plataforma con aplicaciones para descargar en el Smartphone cuando lanzó el iPhone y la App Store. Desde entonces, Google ha mantenido una lucha con Apple desde el principio, tratando de superar el dominio que tenía la compañía de Cupertino, y poco a poco lo ha ido consiguiendo. Ahora, Android ya tiene mayor tráfico publicitario que iOS. Concretamente, el 42,83% del tráfico publicitario total es de usuarios que utilizan Android. IOS queda relegado a una segunda posición con un 38,17%.

Figura 2: trafico publicitario de los SO

Cada día hay más dispositivos, y no sólo eso, sino que cada día hay más fabricantes dispuestos a trabajar con su sistema operativo.

Es la plataforma en la que más fabricantes confían, y en la que más hardware encontramos disponible (y aumentando cada día que pasa…).

El hecho de que cada vez la cuota de mercado que Google le quita a RIM o Nokia sea mayor, hace que el hecho de programar para Android sea sinónimo de programar para un mayor porcentaje del mercado. Y ese hecho supone, además, que muchos desarrolladores hacen un esfuerzo extra para adaptarse a las múltiples configuraciones de hardware disponibles.

2.2.2 Modelos de negocio para las apps

1. La app gratuita: No todo el mundo sabe o puede pagar apps de pago, por lo que la cantidad de descargas de este tipo de apps es muy alta. Además, al ser gratuita permite al usuario darle una oportunidad aunque, a priori, no esté interesado.

La rentabilidad es nula, pero es una forma interesante de darse a conocer a los usuarios para la creación de otras apps con otro modelo de negocio.

2. La app gratuita con anuncios: casi igual a la app gratuita, pero añadiendo banners de publicidad. A priori, puede ser la solución más rentable de todas, aunque muchos usuarios no lo vean con buenos ojos. Inconscientes del trabajo que lleva detrás y a la rentabilidad que debe de tener su creador.

¿Cómo funciona este modelo publicitario?

El programador utiliza un intermediario como, por ejemplo, Google AdMob, el cual sirve los anuncios en las apps haciendo posible la publicidad y cobrando un pequeño porcentaje a cambio cuando se efectúa la acción.

La acción puede ser:

- **Coste por cada mil impresiones:** cuando 1000 usuarios visualizan el anuncio.
- **Coste por click:** cada vez que un usuario hace click en el anuncio.
- **Coste por lead:** el usuario hace click, rellena un formulario o se registra. Los ingresos son mayores, pero es más complicado que se lleve a cabo.
- **Coste por acción:** Se lleva a cabo cuando el usuario realiza una acción determinada, como hacer una compra o realizar una descarga. Los ingresos suelen ser bastante altos, normalmente una comisión por cada producto vendido.

3. la App de pago: el usuario sólo puede instalarla si realiza un pago. Tanto en esta modalidad como en la del modelo publicitario, Google se lleva el 30% de los ingresos, y el resto el programador. Estas apps necesitan mucha suerte para triunfar, pues tienen un alcance bastante bajo y necesitan mucha promoción.

4. freemium: combinación del método gratuito con el de pago. En el que se ofrece una versión gratuita con ciertas limitaciones o desventajas, como los anuncios, los cuales desaparecen si compras la versión de pago existente. 15

5. donación: el usuario puede realizar una donación al programador si ha quedado muy satisfecho o le ha supuesto un ahorro de dinero, mostrando su gratitud al programador por el trabajo realizado. También pueden ser apps únicamente destinadas a las donaciones de organizaciones benéficas.

6. Demostración: es aquella que muestra un ejemplo del producto como un periodo de prueba o el acceso a los primeros niveles de un juego. Y luego invita al usuario a comprar la versión final si le ha gustado.

7. Añadidos in-app: Nos permite la compra de características o funcionalidades dentro de una aplicación, así como ventajas sobre otros usuarios, paquetes de niveles, etc.

2.3 Arquitectura Android

Android es una plataforma para dispositivos móviles que contiene una pila de software, donde se incluye un sistema operativo, **middleware** y aplicaciones básicas para el usuario.

A continuación, daré una versión global de las capas, cada una de estas capas utiliza servicios ofrecidos por las anteriores y ofrece, a su vez, los suyos propios a las capas de niveles superiores.

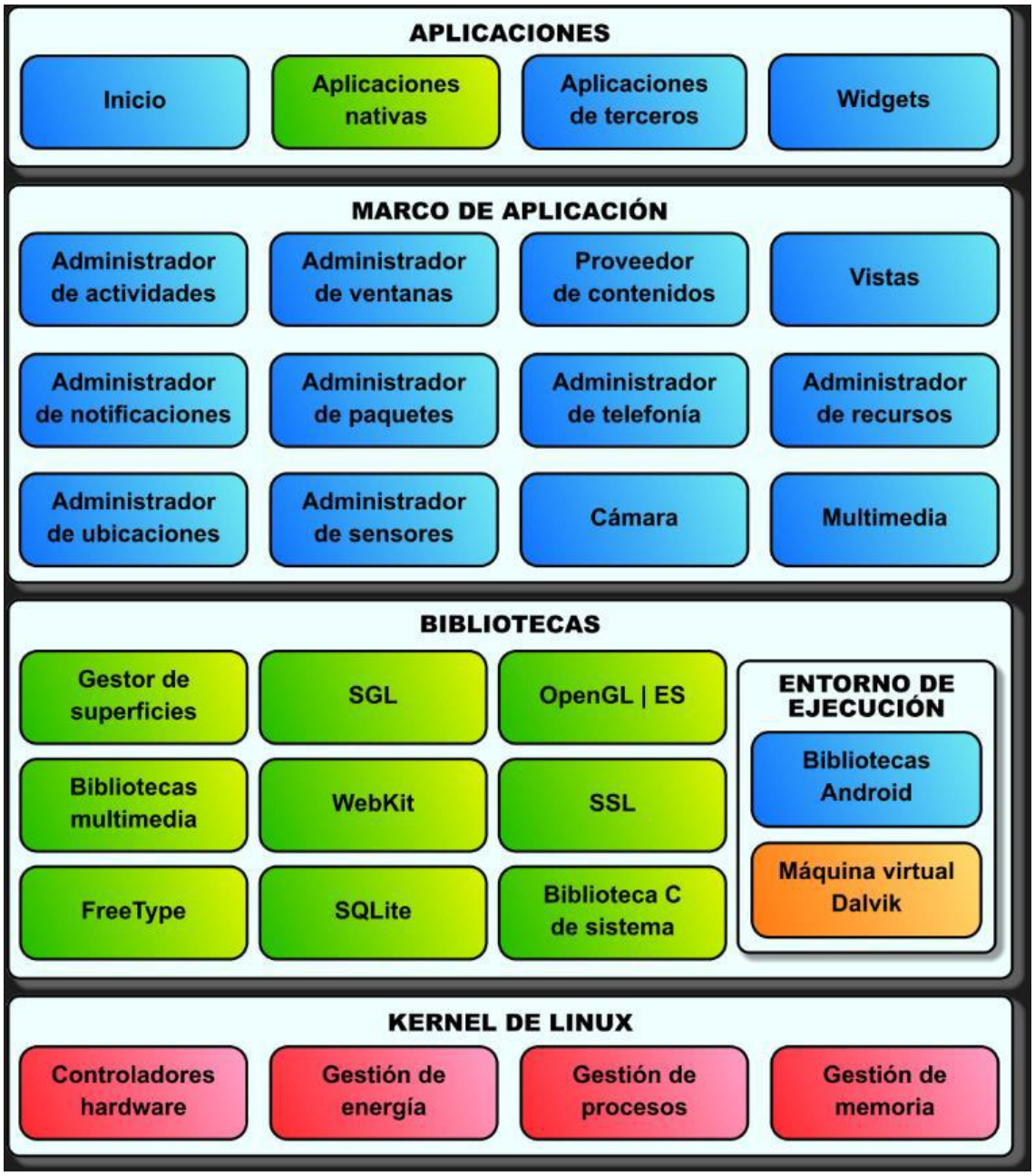

Figura 3: Arquitectura Android

- **Aplicaciones:** En esta última capa se incluyen las aplicaciones tanto nativas (programadas en C o C++) como administradas (programadas en java), las incluidas por defecto de Android como aquellas que el usuario vaya añadiendo posteriormente .Todas estas aplicaciones utilizan los servicios, las API y librerías de los niveles anteriores. En esta capa encontramos también la aplicación principal del sistema: Inicio (Home) o lanzador (launcher), que permite ejecutar otras aplicaciones mediante una lista, mostrando diferentes escritorios donde se pueden colocar accesos directos a aplicaciones y widgets.

- **Framework de Aplicaciones:** Representa fundamentalmente el conjunto de herramientas de desarrollo de cualquier aplicación. Toda aplicación que se desarrolle para Android, ya sean las propias del dispositivo, las desarrolladas por Google o terceras compañías, o incluso las que el propio usuario cree, utilizan el mismo conjunto de API y el mismo "framework", representado por este nivel.

- **Tiempo de ejecución de Android:** Al mismo nivel que las librerías de Android se sitúa el entorno de ejecución. Éste lo constituyen las Core Libraries, que son librerías con multitud de clases Java y la máquina virtual Dalvik.

Las aplicaciones se codifican en Java y son compiladas en un formato específico para que esta máquina virtual las ejecute. La ventaja de esto es que las aplicaciones se compilan una única vez y, de esta forma, estarán listas para distribuirse con la total garantía de que podrán ejecutarse en cualquier dispositivo Android que disponga de la versión mínima del sistema operativo que requiera la aplicación.

- **Núcleo Linux**: Android utiliza el núcleo de Linux 2.6 como una capa de abstracción para el hardware disponible en los dispositivos móviles. Esta capa contiene los drivers necesarios para que cualquier componente hardware pueda ser utilizado mediante las llamadas correspondientes. Siempre que un fabricante incluye un nuevo elemento de hardware, lo primero que se debe realizar para que pueda ser utilizado desde Android es crear las librerías de control o drivers necesarios dentro de este kernel de Linux embebido en el propio Android.

El kernel también se encarga de gestionar los diferentes recursos del teléfono (energía, memoria, etc.) y del sistema operativo en sí: procesos, elementos de comunicación (*networking*), etc.

2.4 Estructura de una aplicación

A la hora de desarrollar una aplicación para Android tendremos múltiples componentes independientes comunicados entre sí a través del sistema operativo, los cuales, implementan una serie de métodos que son invocados por el sistema operativo cada vez que se cumplan las condiciones necesarias para llamarlos. Por lo tanto, las aplicaciones en Android se comportan de forma asíncrona y es el SO el que se encarga de gestionar estas llamadas según las peticiones del usuario.

Componentes que permite declarar el API de Android:

Servicios
Un servicio es una tarea del sistema que se ejecuta en segundo plano o Background sin que el usuario esté interactuando con ella, es decir, sin interfaz gráfica. Los Servicios son usados para realizar operaciones de larga duración, de manera que el usuario no tenga que esperar a que se procese. Un ejemplo de esto es un reproductor de música en el que, una vez que se ha iniciado el usuario, puede realizar otras acciones diferentes.

Actividades
Un Activity es el componente de Android que proporciona una ventana en una aplicación con la que los usuarios pueden interactuar para realizar una acción como, buscar un contacto, realizar una foto, enviar una foto etc.
Entre las diversas cantidades de actividades que pueden formar una aplicación, una de ellas se marcará como la actividad principal o "Main", que inicia la aplicación.
Las actividades pueden iniciar otras actividades con el objetivo de componer la aplicación.

En el siguiente gráfico mostraremos el ciclo de vida de una actividad en Android en la que onCreate() y onDestroy() representan el principio y el fin de la aplicación.

OnStart () y onStop () representan la parte visible de la aplicación, es decir, la actividad será visible para el usuario aunque no tenga el foco de acción.

onResume() y onPause() son las que delimitan la parte útil del ciclo de vida, es decir, además de ser visible tendrá el foco de acción y el usuario podrá interactuar con ella.

El proceso que mantiene a la actividad sólo puede ser eliminado cuando se encuentra en onPause () o en onStop (), cuando no tenga el foco de

aplicación. Aunque el proceso sea eliminado, puede ser restaurado gracias a una copia, volviendo así a estar activo como si no hubiera sido eliminado. Además, la activity puede haber estado en un segundo plano invisible y puede ser despertada pasando por el estado onRestart ().

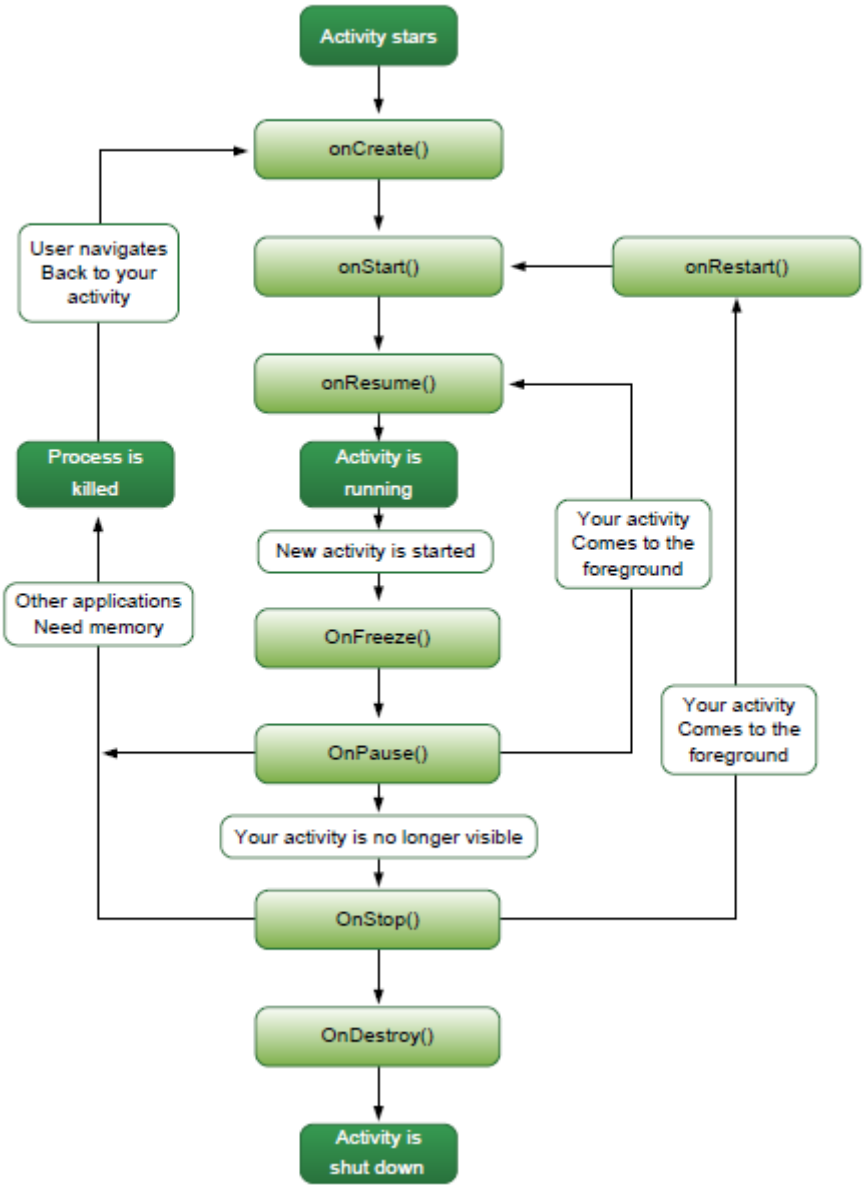

Figura 4: Ciclo de vida de una actividad en Android

Broadcast Receivers

Es el componente que está destinado a recibir y responder ante eventos globales generados por el sistema, como: “batería baja “, “sms recibido”, “tarjeta SD insertada” o por otras aplicaciones. Estos componentes son representados por clases Java, lo que da a entender que cuando la aplicación se ejecuta se crean instancias de éstos, pudiéndose crear más de una instancia del mismo componente.

Intent

Es un mecanismo para describir una acción específica, declarar algo que necesitamos. Son fragmentos de información que describen la acción o servicio deseado. Una clase que permite especificar una activity a ejecutar, llamando a uno de los métodos de la clase Activity con esos intent de parámetro.

Un intent es el elemento básico de comunicación entre los distintos componentes Android que hemos descrito anteriormente. Se pueden entender como los mensajes o peticiones que son enviados entre los distintos componentes de una aplicación o entre distintas aplicaciones. Mediante un intent se puede mostrar una actividad desde cualquier otra, iniciar un servicio, enviar un mensaje broadcast, iniciar otra aplicación, etc.

Content Provider

Es la forma de compartir datos entre aplicaciones que se han definido en Android, sin tener que mostrar detalles de su almacenamiento interno, su estructura o su implementación. De esta manera la aplicación puede tener acceso a los datos de otras aplicaciones que se hayan definido usando el content provider. También puede almacenar datos en los ficheros del sistema, en la base de datos SQLite de la aplicación, una página Web o en cualquier otro lugar de almacenamiento.

2.5 Necesidades

Para instalar un sistema de desarrollo de aplicaciones Android, necesitamos 3 paquetes.

 Un primer paquete es el entorno de desarrollo Eclipse. También podría valer el entorno Netbeans, pero elegimos el primero por ser el más difundido

a la hora de trabajar con Android. Lo podemos obtener en la web http:// www.eclipse.org

 Un segundo paquete que necesitamos es el Java Development Kit (JDK), el cual proporciona el entorno de desarrollo para java, que se acopla perfectamente a Eclipse. Lo podemos obtener en la web http://www.oracle.com

 El tercer paquete es el Android SDK que nos proporcionara las extensiones necesarias para enfocar nuestra programación hacia dispositivos móviles. Lo podemos obtener en la web http://developer.android.com

Tanto las instalaciones como el desarrollo del proyecto se realizarán bajo sistema operativo Windows, pero existen otras versiones que posibilitan la realización de proyectos bajo otra plataforma.

Aunque hemos visto la instalación de los tres paquetes por separado, existe en la actualidad un paquete software que incluye, básicamente, Eclipse con el plugin ADT (Android Developer Tool) y el Android SDK.

2.6 Estructura de carpetas

Los proyectos están formados por una estructura de carpetas que están almacenadas en nuestro espacio de trabajo (workspace). En nuestro caso el proyecto lo hemos denominado como Zwait y corresponde con la carpeta principal. El resto del contenido es el mismo para cualquier proyecto Android.

Vamos a realizar una breve explicación de cada una de las carpetas ya que es fundamental conocer donde debemos colocar cada objeto que vayamos desarrollando.

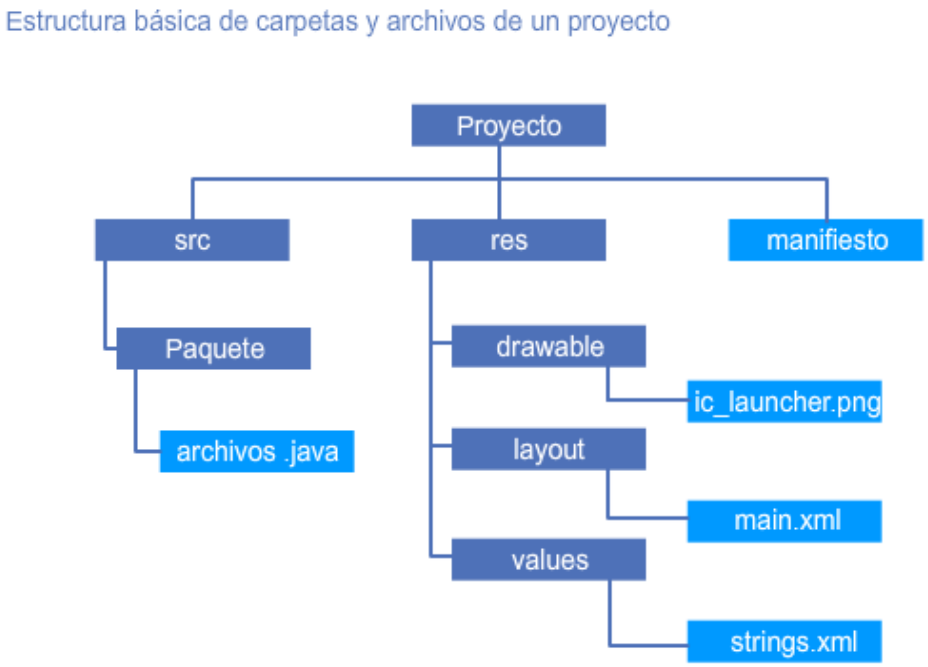

Figura 5: Estructura básica de carpetas y archivos de un proyecto

La carpeta **"src"** contiene el código fuente de las aplicaciones, organizadas en uno o más paquetes que agrupan a una o más clases java, es decir los archivos del tipo .java donde se crean las actividades.

Una **actividad** viene a ser como una pantalla o formulario. Por ejemplo si hablásemos de la libreta de direcciones de nuestro móvil, la actividad principal sería mostrar el listado de direcciones. Si pulsásemos el botón para dar de alta una nueva dirección se abriría otra actividad distinta. Si pulsásemos el botón para editar una dirección se abriría otra actividad distinta.

La carpeta **"gen"** no debe tocarse nunca, ya que aquí se encuentran los archivos de compilación generados automáticamente a partir del código fuente.

En la carpeta **"assets"** se guardan archivos auxiliares que se pueden necesitar en nuestra aplicación, como por ejemplo ficheros de sonido.

La carpeta **“res”** es otra carpeta de recursos para la aplicación, pero más importante ya que contiene aquellos recursos fundamentales para ésta, como son las pantallas que se le muestran al usuario, los iconos con los que se representa la aplicación, etc.

Esta carpeta se subdivide en otras para organizar los archivos por tipos.

- Tipo drawable: contiene ficheros de imagen jpg, gif, png; por ejemplo iconos, imágenes que se muestran en la interface del usuario, fondos de aplicación, etc.
- Tipo Layout: contiene ficheros XML que definen el aspecto de la interfaz del usuario.
- Tipo values: entre otros contiene el fichero string.xml, donde se definen las cadenas de texto usadas en la aplicación.

  Además existen otras carpetas que contienen otro tipo de elementos, habitualmente en proyectos complejos, que veremos según sean necesarias.

La gran variedad de carpetas que ofrece Android permite que se creen proyectos versátiles que se adapten a gran cantidad de escenarios. Por ejemplo, mediante la organización de imágenes en diferentes tipos de carpetas drawable, se podrán cargar imágenes a diferentes resoluciones, según el móvil en el que tenga que funcionar la aplicación. Mediante el uso de diferentes carpetas del tipo value se conseguirá tener aplicaciones con textos en varios idiomas.

2.7 Fichero Manifest

El fichero “AndroidManifest.xml” está presente en todos los proyectos Android, y sirve para indicar cuál es la actividad principal del proyecto, es decir, que actividad debe ejecutarse primero al arrancar el programa, si aparece un icono cuando se instala la aplicación en un móvil, etc.

La información que contiene es leída por el sistema operativo justo al empezar a arrancar la aplicación.

El fichero de manifiesto realiza las siguientes acciones:

1. Describe los componentes de la aplicación, tales como actividades, servicios, proveedores de contenido, y bajo qué condiciones debe lanzarse.

2. Declara los permisos que debe tener la aplicación para acceder a determinados recursos del dispositivo móvil o cómo debe interactuar con otras aplicaciones.

3. Establece la versión de Android mínima sobre la cual la aplicación puede funcionar.

Este fichero sigue una estructura XML, donde se definen etiquetas para detallar la actividad principal, los iconos, los permisos, etc.

Es muy importante recordar que todos los componentes de un proyecto deben indicarse en el AndroidManifest. Las actividades son uno de los tipos de componentes del proyecto. Existen otros tipos que no se verán de momento.

Tener la información del Manifest actualizada, correcta y ordenada, ayudará mucho a la hora de mantener nuestra aplicación, pero sobre todo ayudará a que sea visible en el Google Play, el cual muestra a los móviles las aplicaciones sólo si estos cumplen los requisitos de hardware y software especificados por la aplicación en su manifest.

Es decir, si tenemos mal la información del manifest nuestra aplicación, en el mejor de los casos, podrá ser descargada por dispositivos que no la pueden ejecutar correctamente dando como resultado puntuaciones y comentarios negativos, o directamente no aparecerá para ser descargada.

Algunas de los componentes definidos en el fichero de manifestó son:

manifest: Dentro de este tag encontramos los siguientes atributos:

1.- package: Es el nombre del paquete de nuestro programa.

2.- android:versionCode: Hace referencia al número de versión de desarrollo de nuestro programa, cada versión final que se desea publicar debe tener un versionCode distinto.

3.- android:versionName: Este es el número de versión de nuestro programa.

Application: Aquí van todas las Activities, Services, Providers, Receivers y las bibliotecas que se usan en nuestra aplicación.

1.- activity: Se tendrá por cada pantalla de nuestra aplicación y dentro de él van sus atributos y los distintos intents para comunicarse. Algunos de los atributos que nos podemos encontrar son:

a.- android:name = .UnActivitiy: Este es el nombre de la clase de nuestra Activity.

b.- android:label = @string/app_name: Es el texto que aparecerá en la barra superior de nuestra activity.

c.- android:theme = @style/Theme.NoBackground: En Android uno puede crear Themes (conjuntos de estilos) para reutilizar en una aplicación y de esa manera poder mantener la coherencia en el diseño de la aplicación.

d.- android:configChanges = orientation|keyboardHidden: Este atributo establece que la Activity no detectará los cambios de orientación y por ocultación del teclado físico.

e.- android:screenOrientation = portrait: Determinando esta propiedad le decimos a nuestra Activity que sólo tendrá disposición en modo en modo portrait (el teléfono puesto verticalmente. Recordemos que landscape significa el teléfono puesto horizontalmente).

Supports-screens: Este tag es utilizado para describir las pantallas soportadas por nuestra aplicación. Entre los atributos podemos encontrar:

1.- android:largeScreens = false: Especifica si se van a soportar pantallas del tipo large (tabs, netbooks, etc.) con nuestra aplicación.

2.- android:normalScreens = true: Realiza lo mismo que el largeScreens pero haciendo referencia a las pantallas normal, más o menos del entorno de 3,5 pulgadas.

3.- android:smallScreens = false: lo mismo que los anteriores pero para dispositivos con pantalla pequeña.

4.- android:anyDensity = true: Este atributo sirve para determinar que nosotros nos encargaremos de escalar las pantallas para distintas densidades y que el teléfono no tendrá que hacerse cargo de adaptarlas automáticamente.

uses-permissions: Mediante este Tag especificamos los permisos que va a necesitar nuestra aplicación para poder ejecutarse, además son los que deberá aceptar el usuario antes de instalarla. Por ejemplo, si se desea utilizar funcionalidades con Internet o el vibrador del teléfono, hay que indicar que nuestra aplicación requiere esos permisos.

uses-sdk: En este tag determinamos las distintas versiones Android que va a utilizar nuestra aplicación, tanto sobre qué versiones va a correr como qué versión fue utilizada para realizar nuestras pruebas. Mediante el atributo android:minSdkVersion establecemos a partir de que versión de Android nuestra aplicación podrá correr.

Capítulo 3.

Entorno de desarrollo y componentes

Durante el desarrollo del proyecto he intentado usar el software y las herramientas más adecuadas para realizarlo, las cuales detallaré a continuación.

3.1 El SDK de Android (Software Development Kit)

El SDK de Android incluye un conjunto de herramientas de desarrollo. Comprende un depurador de código, biblioteca, un simulador de teléfono basado en QEMU, documentación, ejemplos de código y tutoriales.

Las plataformas de desarrollo soportadas incluyen Linux, Mac OS X 10.4.9 o posterior, y Windows XP o posterior. La plataforma integral de desarrollo (IDE, Integrated Development Environment) soportada oficialmente es Eclipse, junto con el complemento ADT (Android Development Tools plugin). Aunque también puede utilizarse un editor de texto para escribir ficheros Java y Xml, y utilizar comandos en un terminal (se necesitan los paquetes JDK, Java Development Kit y Apache Ant) para crear y depurar aplicaciones. Además, pueden controlarse dispositivos Android que estén conectados.

Las Actualizaciones del SDK están coordinadas con el desarrollo general de Android. El SDK soporta también versiones antiguas de Android, por si los programadores necesitan instalar aplicaciones en dispositivos ya obsoletos o más antiguos. Las herramientas de desarrollo son componentes descargables, de modo que una vez instalada la última versión, pueden instalarse versiones anteriores y hacer pruebas de compatibilidad.

3.2 Eclipse Versión: 4.3.1

Eclipse es un programa informático compuesto por un conjunto de herramientas de programación de código abierto multiplataforma para desarrollar, lo que el proyecto llama, "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano", basadas en navegadores.
Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados, como el IDE de Java, llamado *Java Development Toolkit* (JDT); y el compilador (ECJ), que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse).
Eclipse fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas para VisualAge. Eclipse está siendo ahora desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios.

El entorno de desarrollo integrado (IDE) de Eclipse emplea módulos (en inglés *plug-in*) para proporcionar toda su funcionalidad al frente de la plataforma de cliente enriquecido, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no. Este mecanismo de módulos es una plataforma ligera para componentes de software.

Ventajas en la utilización de Eclipse:

1- El entorno de desarrollo integrado (IDE) de Eclipse emplea módulos (plug-in) para proporcionar toda su funcionalidad al frente de la Plataforma de Cliente rico, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no.

2- Este mecanismo de módulos es una plataforma ligera para componentes de software. Adicionalmente permite a Eclipse extenderse usando otros lenguajes de programación como son C/C++ y Python, los cuales permiten a Eclipse trabajar con lenguajes para procesado de texto como LaTeX, aplicaciones en red como Telnet y Sistema de gestión de base de datos.

3- La arquitectura plug-in permite escribir cualquier extensión deseada en el ambiente, como sería Gestión de la configuración. Se provee soporte para Java y CVS en el SDK de Eclipse. Y no tiene por qué ser usado únicamente para soportar otros Lenguajes de programación.

En cuanto a la utilización de eclipse para la creación de aplicaciones clientes se puede decir que:

1- Eclipse provee al programador con Frameworks muy ricos para el desarrollo de aplicaciones gráficas, definición y manipulación de modelos de Software, Aplicaciones web, etc. Por ejemplo, GEF (Graphic Editing Framework - Framework para la edición gráfica) es un plug-in de Eclipse para el desarrollo de editores visuales, que pueden ir desde procesadores de texto wysiwyg hasta editores de diagramas UML, interfaces gráficas para el usuario (GUI), etc. Dado que los editores realizados con GEF "viven" dentro de Eclipse, además de poder ser usados conjuntamente con otros plugins, hacen uso de su interfaz gráfica personalizable y profesional.

2- El SDK de Eclipse incluye las herramientas de desarrollo de Java, ofreciendo un IDE con un compilador de Java interno y un modelo completo de los archivos fuente de Java. Esto permite técnicas avanzadas de refactorización y análisis de código.

3.3 EasyPHP 14.1

"EasyPHP" es un programa que instala en un solo paso el servidor Apache, junto con el modulo para programación en PHP y la base de datos MySQL.

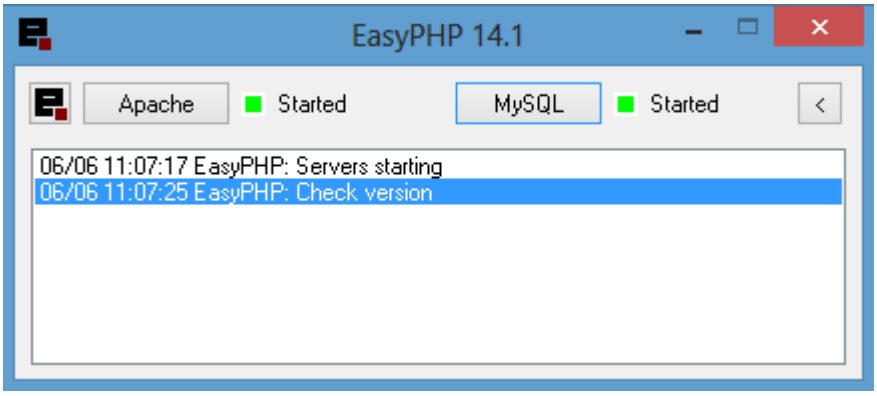

Figura 6: pantalla principal EasyPHP

Debemos comprobar que todo está correcto. De un modo visual, podemos observar dos luces verdes correspondientes al servidor web y a la base de datos, que deben de estar en verde.

Configuración básica

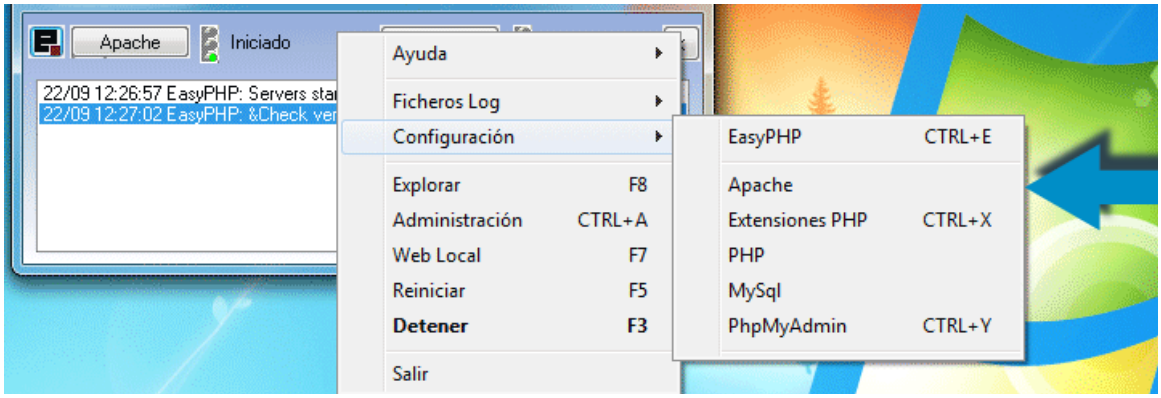

Figura 7: Configuración básica EasyPHP

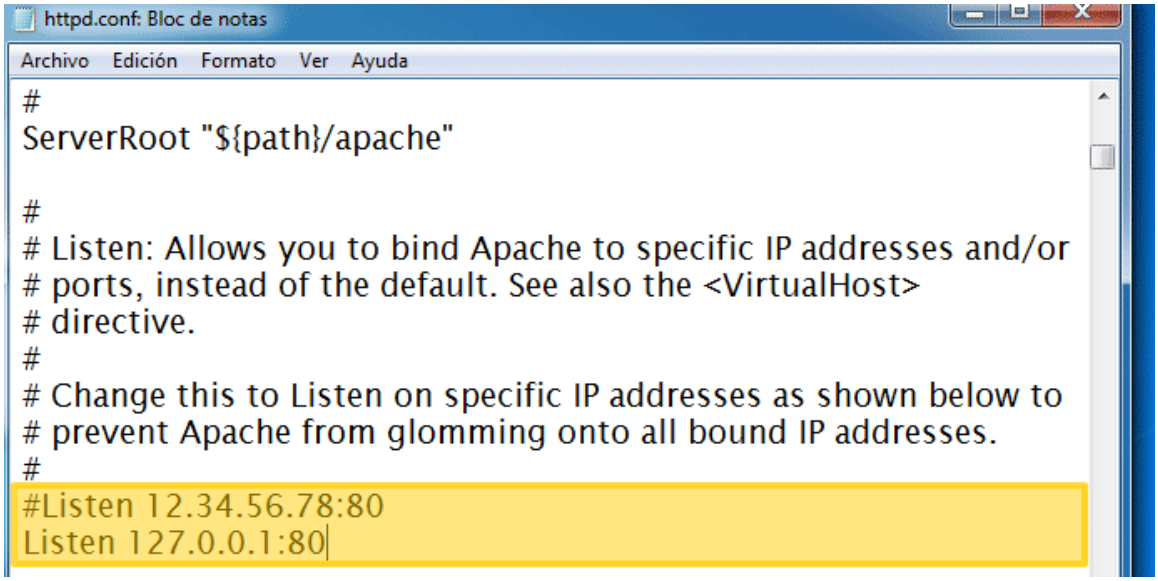

Figura 8: listen loopback

La instrucción “listen” le indica al servidor que escuche todas las llamadas que le entran por la dirección IP 127.0.0.1 (loopback), a la cual conoceremos como “localhost”. El puerto 80 es por defecto el puerto de comunicaciones para http://

En informática, en el contexto de redes TCP/IP,”localhost” es un nombre reservado que tienen todas las computadoras, router o dispositivos independientemente de que disponga o no de una tarjeta de red Ethernet. El nombre “localhost” es traducido como la dirección IP de loopback 127.0.0.1.

3.4 El Servidor Web Apache

Un Servidor Web es un programa que se ejecuta continuamente en un servidor, manteniéndose a la espera de peticiones de ejecución que le hará un cliente o un usuario de Internet. El servidor web se encarga de contestar a esta petición de forma adecuada entregando como resultado una página web o información relativa a los comandos solicitados.

Gracias a los avances en conectividad y la gran disponibilidad de banda ancha, hoy en día es muy común establecer los servidores web dentro de la propia empresa, sin tener que recurrir a caros alojamientos en proveedores externos. Esto es posible gracias a Apache, uno de los mejores y el más utilizado entre los servidores web gracias a su gran estabilidad y confiabilidad.

Entre las ventajas que presenta un servidor como Apache se encuentran las siguientes:

- Es personalizable, la arquitectura modular de Apache permite construir un servidor hecho a la medida. Además permite la implementación de los últimos y más nuevos protocolos.

- En cuanto a la administración, los archivos de configuración de Apache están en ASCII, por lo que tiene un formato simple, y pueden ser editados tan solo con un editor de texto. El servidor puede ser administrado vía línea de comandos, lo que hace la administración remota muy conveniente.

- Por otra parte se trata de un servidor muy eficiente, mucho esfuerzo se ha puesto en optimizar el rendimiento de código C de Apache. Como resultado, éste corre rápido y consume menos recursos de sistema en comparación con otros servidores. Además, Apache corre en una amplia variedad de sistemas operativos, incluyendo varias versiones de UNIX, Windows o MacOS (Sobre Power PC).

3.5 PHP

PHP es un lenguaje de programación del lado del servidor, es decir, la ejecución la hace el servidor y no el cliente, de este modo lo que le llega al

cliente cuando accede a una página creada en código HTML es código HTML, el cual lo interpreta nuestro navegador.
PHP es una alternativa a las tecnologías de Microsoft ASP y ASP.NET (que utiliza C# y Visual Basic .NET como lenguajes), a ColdFusion de la empresa Adobe, a JSP/Java, CGI/Perl y a Node.js/JavaScript.
Aunque su creación y desarrollo se da en el ámbito de los sistemas libres, bajo la licencia GNU, existe además un entorno de desarrollo integrado comercial llamado Zend Studio.

PHP es comúnmente utilizado para:

- Acceder a bases de datos.
- Procesar datos de formularios.
- Enviar correos desde la interfaz web.
- Manipular información como textos, imágenes, campos o formularios de forma dinámica.
- Crear sitios web y aplicaciones dinámicas.
- Crear foros, portales de noticias, tablones, etc.

3.6 Base de datos MySQL

Una base de datos es un conjunto de datos que están organizados para un uso determinado. Los programas que permiten gestionar estos datos se llaman Sistemas Gestores de Bases de Datos. Estos tratan la información utilizando el modelo de gestión de bases de datos relacional, en el que los datos se organizan en tablas.
Las tablas almacenan información sobre un tema, como pueden ser los clientes de una empresa, o los pedidos realizados por cada uno de ellos.
Para el presente proyecto he empleado la base de datos MySQL porque es una de las más populares, pues es de código abierto y ofrece soporte para más de 20 plataformas como Linux, Windows, NetWare, etc.
Pueden utilizarse infinidad de lenguajes de programación para acceder a MySQL. Existen interfaces para C, C++, PHP, C#, y Java, entre otras.
Es multihilo, por lo que es capaz de trabajar con más de un procesador simultáneamente. Ofrece un alto rendimiento, fiabilidad y facilidad de uso. Proporcionando además una amplia gama de herramientas de bases de datos, por lo que es la mejor opción para las muchas empresas y para mí en este caso.

3.7 Notepad++ v6.6.1

Es un editor básico para diversos lenguajes de programación, disponible en varios idiomas, incluido el español. Es de resaltar su sistema de búsqueda y coincidencia de texto con capacidad de filtros de expresiones regulares.
Con este pequeño y poderoso "blog de notas "podemos abrir y editar archivos HTML, PHP y otros lenguajes fácilmente.
El programa reconoce el tipo de formato con el que estoy trabajando y modifica sus características acorde a él. Así un archivo de tipo PHP, HTML o CSS tendrá resaltadas variables y valores con distintos colores, para facilitar la edición.

Capítulo 4.

Tareas en segundo plano Android y AsyncTask

Introducción

A veces tenemos aplicaciones que tardan mucho tiempo en realizar una tarea pesada y queremos que esta no bloquee la aplicación mientras se está ejecutando. Si estamos realizando aplicaciones complejas como tener que realizar una conexión con el servidor o enviar y recibir datos, como es el caso de mi app, es obligatorio en las últimas versiones del sistema operativo Android, que esto se haga sin interrumpir el hilo principal, ya que no será permitido, produciéndose múltiples errores de código en la app. El hilo principal del usuario es el principal en mi aplicación. Normalmente es el único que se utiliza en app sencillas, sin embargo, existen aplicaciones complejas en las que tenemos múltiples hilos que se ejecutan a la vez. Como por ejemplo, una en la que utilizamos el hilo principal para movernos por la interfaz de pantalla mientras que, en segundo plano, se están recibiendo datos de un servidor.

4.1 ¿Cómo se soluciona y que elecciones tenemos?

Para resolver esto normalmente se hace uso de los Threads (hilos). Para ello, creamos un hilo adicional para nuestra tarea y continuamos con el hilo normal para el resto de la aplicación. El problema es que la interfaz gráfica (*UI*) de Android no permite llamadas desde otros hilos que no sea el suyo, por lo que si realizamos alguna actualización, como añadir información nuestra aplicación se cerrará.

4.2 Usando threads (hilos) de java.

Mediante el uso de hilos, se pueden crear aplicaciones que responden cuando se está trabajando con procesos que requieren mucho tiempo.
En Java se puede crear una clase que extiende de la clase Thread y anula el método public void run (), después se llama al método start () para ejecutar el proceso que consume más tiempo. Si la clase ya extiende de otra clase, se puede implementar la interfaz Runnable.
Cuando empieza la actividad en el método **onCreate ()** creamos un objeto de tipo **Thread** que está construido con un objeto **Runnable**.
El método de la clase **Runnable run ()** se ejecutará cuando llamemos al método **start ()** de la clase **Thread**. Desde aquí se pude llamar a otro método o varios métodos y operaciones que requieren mucho tiempo y que, de otra forma, harían que la aplicación no respondiese.
Normalmente, cuando nosotros trabajamos con hilos obtenemos unos resultados que queremos enseñar al usuario. Si intentamos actualizar la interfaz gráfica de usuario desde el **Thread** (no desde el **Thread** principal) la aplicación se bloqueará.

Los **Thereads** creados e inicializados seguirán funcionando incluso si el usuario sale de la aplicación. Se puede hacer un seguimiento de los **Threads** y decirles que se detengan, esto normalmente se hace con un valor booleano. Para estar seguros de que el **Thread** se detiene cuando el usuario sale de la aplicación, antes de llamar al método **start ()** el objeto **Thread**, lo estableceremos como un **Thread** demonio.

4.3 Utilizando la clase AsyncTask

Esta segunda solución es más que suficiente, y además es más sencilla y más limpia de implementar. Considero que ésta es mejor que las demás, utilizando la clase AsyncTask.
Uno de los problemas de los sistemas operativos Android es que no es posible realizar una conexión al servidor en una única tarea principal, por lo que es necesario crear tareas adicionales en el servidor para poder hacer esta conexión.
Debido a los problemas que me han aparecido a la hora de solventar esta cuestión, tuve que buscar una manera de realizar este tipo de conexión. Finalmente encontré que se pueden realizar estas tareas en segundo plano con la clase AsyncTask.

Una estructura típica de una clase de AsyncTask es la siguiente:

```
private class miTarea extends AsyncTask<x, y, z>

protected void onPreExecute(){

}
```

- **X:** Datos que pasaremos al comenzar la tarea.
- **Y:** Parámetros que necesitaremos para actualizar la *UI*.
- **Z:** Dato que devolveremos una vez terminada la tarea.

OnPreExecute (). Se ejecutará antes de la tarea en segundo plano. Se suele utilizar para preparar la ejecución de la tarea, inicializar la interfaz, etc.

```
EJEMPLO DE MI APP:
      class CargaCita extends AsyncTask<String, String, String> {

      }
```

Quiero crear una aplicación que descargue un archivo de internet. A la tarea le pasaré la dirección del fichero (*String*). Conforme vaya descargándolo irá mostrando un mensaje informándonos de qué está pasando (*String*), y una vez terminado devolverá una cadena de caracteres (*String*).

Si alguno de los parámetros no lo necesitamos, se puede sustituir por void.

```
protected Z doInBackground(X...x) {
}
```

Al crear una clase que extiende de AsyncTask tendremos que implementar un método abstracto (ya que AsyncTask es una clase abstracta), llamado **doInBackground(),** el cual contendrá el código de la tarea que queremos ejecutar en segundo plano. Podemos decir que es el más importante de la clase AsyncTask. Éste método será el encargado de realizar la tarea en segundo plano de la aplicación, en un hilo diferente al principal. Aquí la entrada es un conjunto de objetos x, donde, cómo podemos ver en la cabecera, son los que le entran a la extends de AsyncTask, devolviendo un objeto Z.

```
protected void onProgressUpdate(Y y){

}
```

OnProgressUpdate (). Se ejecutará cada vez que llamemos al método **publishProgress (y)** desde el método **doInBackground ()**. Normalmente, se utiliza para actualizar la interfaz de usuario cuando queremos mostrar algún progreso o información en pantalla principal, mostrando la operación que se está haciendo en el fondo.

```
protected void onPostExecute(Z z){

}
```

onPostExecute(). Se ejecutará cuando finalice nuestra tarea en segundo plano, es decir, cuando termine el método **doInBackground()**. Este método es utilizado cuando, después de toda la operación que se hizo en background, lo manda llamar y le entra como parámetro de entrada, el parámetro de salida del método de doInBackground.

Entonces podemos decir que los valores que entran a la AsyncTask son X, que es el tipo de variable de entrada que tenemos para el proceso del fondo. La Y es el tipo de objetos que se van a introducir en el método **onProgressUpdate**, y la Z es el tipo de objetos que tendremos del resultado de las operaciones hechas en doIbBackground.

Para hacer llamar esta tarea lo que tenemos que hacer es:

```
miTarea tarea1 = new miTarea ();

tarea1.execute(x);
```

Donde x es la entrada del parámetro del tipo X.
Podemos, una vez que se esté ejecutando, saber el estatus con el siguiente método.

```
tarea1.getStatus();
```

resultados:

Running: Indicando que la tarea se está ejecutando.
Pending: Indicando que la tarea no se está ejecutando.
Finished: Indicando que onPostExecute(); ha terminado.

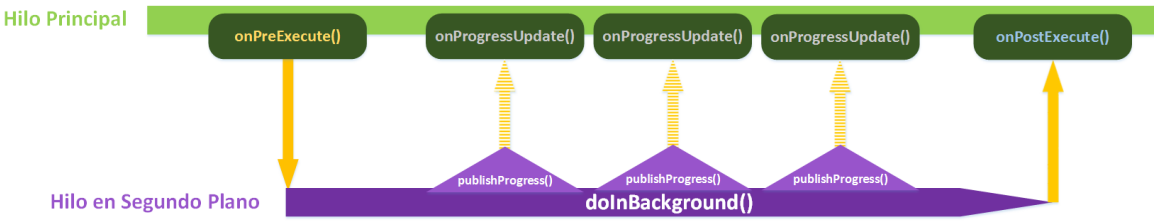

Figura 9: proceso completo AsyncTask

Capítulo 5.

JSON como método de intercambio de datos con el servidor.

JSON (JavaScript Object Notation - Notación de Objetos de JavaScript) es un formato ligero de intercambio de datos. Leerlo y escribirlo es simple para humanos, mientras que para las máquinas es simple interpretarlo y generarlo. Está basado en un subconjunto del Lenguaje de Programación JavaScript, Standard ECMA-262 3rd Edition - Diciembre 1999. JSON es un formato de texto que es completamente independiente del lenguaje pero utiliza convenciones que son ampliamente conocidos por los programadores de la familia de lenguajes C, incluyendo C, C++, C#, Java, JavaScript, Perl, Python, y muchos otros. Estas propiedades hacen que JSON sea un lenguaje ideal para el intercambio de datos.

JSON está constituido por dos estructuras:

- Una colección de pares de nombre/valor. En varios lenguajes esto es conocido como un objeto, registro, estructura, diccionario, tabla hash, lista de claves o un arreglo asociativo.
- Una lista ordenada de valores. En la mayoría de los lenguajes esto se implementa como arreglos, vectores, listas o secuencias.

En JSON, se presentan de estas formas:

Un objeto es un conjunto desordenado de pares nombre/valor. Un objeto comienza con {(llave de apertura) y termine con} (llave de cierre). Cada nombre es seguido por: (dos puntos) y los pares nombre/valor están separados por, (coma).

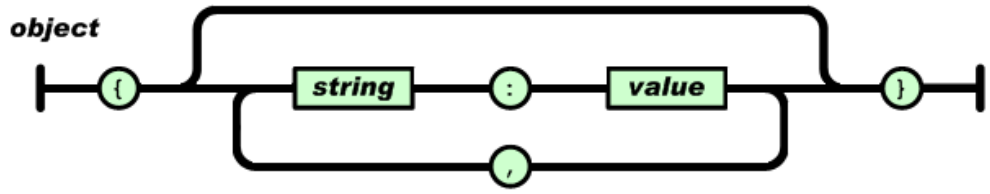

Figura 10: representación JSON objeto

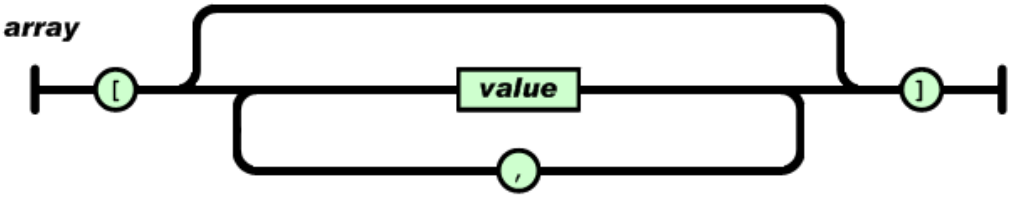

Figura 11: representación JSON array

En resumen JSON es una forma de codificar objetos, arrays o cualquier otra serie de datos en un string y, posteriormente, poder descodificarlo sin muchos problemas. Es especialmente útil para enviar información entre cliente y servidor de una forma muy sencilla, puesto que cada componente descodifica la información según le convenga, indistintamente del resto del sistema.

5.1 ¿Porque utilizar esta opción y no utilizar otras opciones posibles como XML?

Durante un buen tiempo, XML ha sido la única opción para compartir datos libremente. No había otros formatos abiertos disponibles y XML fue considerado como la solución para todos los problemas de intercambio de datos. Este único formato podía manejar datos clásicos como números y texto, pero también podía manejar documentos, formatos, imágenes, audio, vídeo, y mucho más. Ahora que otras opciones están disponibles, XML puede ser una exageración en muchos casos.

Estabilidad

Con JSON, se está limitado a almacenar sólo datos clásicos como texto y números. Sin embargo, XML le permite almacenar cualquier tipo de datos que se le pueda ocurrir. La capacidad para extender los atributos de los datos almacenados en los ficheros XML es lo que le permite ser más flexible que JSON. Sin embargo, también lo hace más difícil de leer. Esto hace a XML más extensible, pero puede que no sea algo bueno. Esto depende del tipo de

la información que trata de transferir. Los documentos requieren extensibilidad para gestionar imágenes, tablas, gráficos, y otros elementos de formato. Sin embargo, los datos clásicos no requieren esta extensibilidad y pueden beneficiarse de la simplicidad de JSON.

Mayor legibilidad

Los ficheros JSON son más restrictivos y, por lo tanto, ligeramente más legibles. Esto se debe a que el número de formatos de datos permitidos por JSON es mucho menor que XML. Además, la estructura de los datos está más estandarizada con los ficheros JSON debido al hecho de que existen menos opciones cuando se compara con el formato XML.

Integración completa con todos los formatos

Con XML es posible adjuntar cualquier fichero de cualquier formato. Por otro lado, JSON sólo soporta formatos de datos tradicionales. Esto significa que es posible incluir fotos, audio, vídeo, y otros ficheros dentro de un fichero XML. Mientras que esto puede parecer algo bueno al principio, también puede ser peligroso. Eso es porque podría incluir un fichero ejecutable que podría tener peligrosas consecuencias para la seguridad. La simplicidad de las estructuras de datos que JSON soporta hace imposible los ataques usando este formato.

Compartir datos tradicionales

JSON es la mejor herramienta para compartir datos. Esto se debe a que los datos están almacenados en vectores y registros, mientras que XML almacena los datos en árboles. Ambos tienen sus ventajas, pero las transferencias de datos son mucho más fáciles cuando los datos se almacenan en una estructura que está familiarizada a los lenguajes orientados a objetos. Esto hace que sea muy sencillo importar datos desde un fichero JSON a Perl, Ruby, JavaScript, Python, y otros muchos lenguajes. Para hacer lo mismo con XML, necesitaría primero transformar los datos antes de que puedan ser importados. Por este motivo, JSON es un formato de fichero superior para las APIs web.

La opción XML pasa por:

- Convertir el array en un XML.
- Pasarle el XML al servidor.
- Parsear el XML en el servidor.
- Convertir los datos en un array.

Además de ser un pesado sistema cuando cambian las especificaciones de datos, puedes llevarte la sorpresa de que el cliente ahora quiera los datos en un array multidimensional.

Compartir documentos

Cuando se quiera compartir documentos, XML es la herramienta adecuada para ello. Esto se debe a que permite incluir tipos de datos como imágenes, tablas, y gráficos. Además, XML ofrece opciones para transferir la estructura, o formato de los datos, junto con los verdaderos datos. JSON sólo ofrece opciones para la transferencia de los datos sin formato, y sólo utiliza formatos de datos tradicionales. Esto hace a XML el formato superior para documentos.

5.2 XML y JSON comparación de análisis de datos

El cuadro comparativo muestra el tiempo más rápido de análisis de JSON para los datos más pequeños. Sin embargo, aumentando el tamaño de los datos XML se realiza un análisis más rápido. El tiempo de carga es más rápido con JSON y también el tamaño de los datos de JSON es menor.

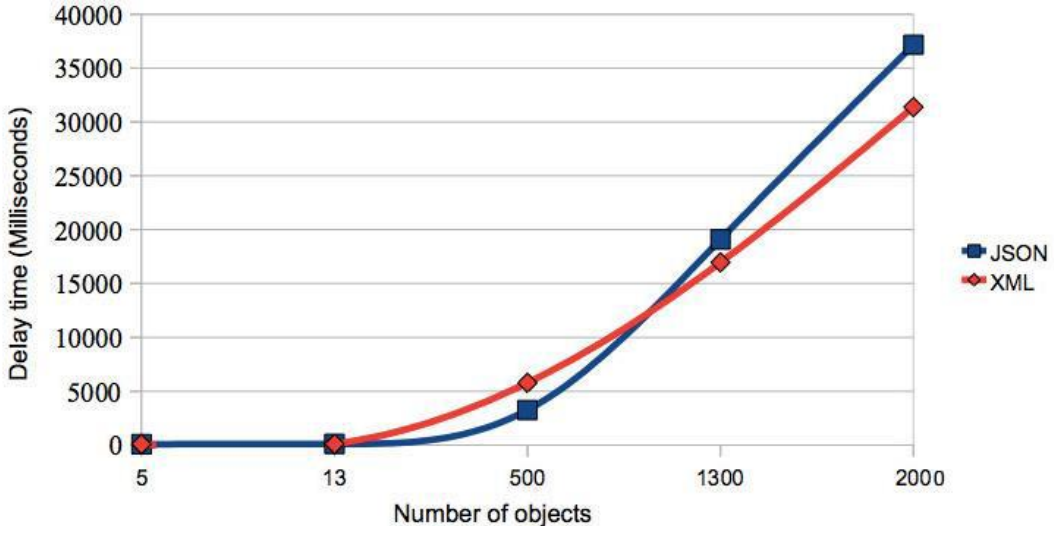

Figura 12: comparación XML vs JSON análisis de datos

5.3 Conclusión ¿Por qué JSON?

JSON es el mejor método para compartir datos tradicionales, ya que los datos están almacenados en vectores y en registros. Este tipo de datos es el que voy a usar en este proyecto, además de que con XML se deberían de transformar los datos.

La eficiencia en el uso de los recursos a la hora de desarrollar aplicaciones móviles es una piedra angular que no podemos obviar, sobre todo, en el caso del consumo de datos de internet, que deberemos optimizar lo máximo posible para ahorrar batería y aprovechar las tarifas de datos existentes actualmente. En este sentido JSON, gracias a su estructura, ahorra considerablemente respecto a XML.

Capítulo 6.

Interacción con Base de Datos

En este capítulo explicaremos el flujo de trabajo de la página web de la clínica, su interacción con el usuario mediante formularios para la manipulación de datos, y el tratamiento de los mismos mediante "scripts" PHP interactuando de este modo con la base de datos. En definitiva estudiaremos los siguientes puntos. [3]

- Controlar el flujo de información.
- Recoger información desde la web.
- Manipular dicha información.
- Interactuar con bases de datos.
- Trabajar con sesiones.
-

6.1. Listado de registros de una tabla.

6.1.1. Creación de archivo de conexión

Para interactuar con la base de datos **MySQL** desde nuestro desarrollo **PHP**, necesitamos realizar una conexión con ella que nos permita extraer e introducir datos en la misma. Para ello **PHP** tiene una serie de funciones de conexión a **MySQL**.

Para la conexión utilizaremos dos funciones: **mysql_connect**, que nos conecta al servidor donde está la base de datos; y **mysql_select_db**, que nos permite definir a qué base de datos del servidor deseamos conectarnos.

En nuestro caso hemos creado un único archivo con el script referente a la conexión, de modo que cuando necesitemos acceder a la base de datos, bastará con hacer una referencia a dicho archivo.

conexion.php

```
<?php
$servidor="localhost";
$usuario="root";
$clave="";
$basedatos="zwait";

$descriptor=mysql_connect($servidor,$usuario,$clave);
mysql_select_db($basedatos);
?>
```

6.1.2. Creación de listado

A continuación se muestra paso a paso como realizar, en **PHP**, un listado de registros de una tabla de la base de datos, concretamente de la tabla **gestorcitas**.

El primer paso es establecer la conexión a nuestra base de datos obteniendo el descriptor de la misma. Para ello debemos hacer referencia al fichero **conexion.php**, que contiene todos los datos de la conexión, en nuestra página **PHP** para lo que utilizamos la instrucción "include".

Una vez establecida la conexión, creamos una sentencia **SQL** que realice una lectura de todos los registros de la tabla **gestorcitas**. Dicha sentencia la guardaremos en una variable.

A continuación, para ejecutar la sentencia **SQL** creada, utilizamos la función **mysql_query**, a la cual le pasamos dos parámetros, uno es la sentencia **SQL** y el otro es el descriptor de conexión conseguido anteriormente.

El resultado de la ejecución, que normalmente será un array de registros, lo guardaremos en una variable llamada **resultado**.

Ahora vamos a recorrer el array con el resultado de la ejecución, y obtener cada una de las filas del mismo.

Para ello hacemos uso de la función **mysql_fetch_array**, que nos devuelve cada una de las filas del array. Cuando llega a la última, es decir cuando no quedan más filas, nos devuelve un valor false.

Asignamos el array devuelto por la función a la variable **$row** y con una instrucción **WHILE** recorreremos el contenido del mismo hasta que nos devuelva "false", es decir, hasta llegar a la última fila.

Dentro del bucle, para enviar por pantalla cada uno de los valores del "array", empleamos una instrucción **echo.**

listado.php

```
<html><head></head><body>
<?php
// Estableciendo la conexión
include("conexion.php");

// Creamos la sentencia SQL
$ssql="select * from gestorcitas";
$resultado=mysql_query($ssql,$descriptor);


while($row=mysql_fetch_array($resultado)){

      echo "Id_entidad = ".$row['id_entidad']."<BR>";
      echo "Id_cita = ".$row['id_cita']."<BR>";
      echo "Nombre_Clinica = ".$row['nombre_clinica']."<BR>";
      echo "Fecha_cita = ".$row['fecha_cita']."<BR>";
      echo "Descripcion = ".$row['descripcion']."<BR>"."<BR>";
}

?>
</body>
</html>
```

Comprobamos la página web creada:

Figura 13: Listado de citas

6.2. Altas de registros en una tabla

6.2.1 Formulario de altas

En el siguiente script mostramos el modo de crear un formulario que nos permita dar de alta nuevos registros en la tabla.

Para ello hemos creado un formulario con los datos, organizados en una tabla, que queremos que introduzca el usuario, y que no son otros que los campos de nuestra tabla gestorcitas.

El formulario, que enviará los datos a otra página llamada altas.php utilizando el método POST, contiene los campos id_entidad, id_cita, nombre_clinica, fecha_cita, descripcion, todos de tipo texto, y un campo de tipo archivo llamado imagen. También hemos incluido los botones de enviar y de borrar.

formulario_altas.php

```
<html><head></head>

      <body>

      <form action="altas.php" method="post"
enctype="multipart/form-data">
```

```
                <table width="400" border="0">

                <tr><td colspan="4"> ALTA DE CITAS</td></tr>

                <tr>

                      <td> Id_entidad </td><td><input type="text"
name="id_entidad"/></td>

                      <td> Id_cita </td><td><input type="text"
name="id_cita"/></td>

                </tr>



                <tr>

                      <td> Nombre_clinica </td><td colspan="3">
<input type="text" name="nombre_clinica"/></td>

                </tr>



                <tr>

                      <td> Fecha_cita </td><td><input type="text"
name="fecha_cita"/></td>

                      <td> Descripcion </td><td><input type="text"
name="descripcion"/></td>

                </tr>

                <tr>

                      <td colspan="2"><input type="submit"
name="button" value="Enviar"/></td>

                      <td colspan="2"><input type="reset"
name="button2" value="Restablecer"/></td>

                </tr>



                </table>

                </form>

                </body>

                </html>
```

Comprobamos la página web creada:

Figura 14: Formulario de altas

6.2.2. Ejecución de altas

Mostramos ahora el archivo **altas.php** que recibe los datos del formulario anterior.

Lo primero es establecer la conexión a la base de datos, por lo que incorporamos la instrucción **include** con el archivo **conexion.php**.

A continuación recogemos los datos que nos llegan del formulario.

Hemos creado unas variables cuyos nombres coinciden con los nombres de los campos del formulario y las recogemos mediante la instrucción **$POST**.

A continuación construimos la sentencia SQL que insertará en los campos de la tabla gestorcitas, id_entidad, id_cita, nombre_clinica, fecha_cita y descripcion los valores de las variables que hemos obtenido del formulario. Observa que estas variables han de ir entre comillas simples.

Sólo nos queda escribir el "script" que ejecuta la sentencia **SQL**.

Como hemos hecho anteriormente, utilizamos la función **mysql_query** para ejecutar nuestra sentencia **SQL**.

Para finalizar, emplearemos la instrucción **HEADER**, que remite al usuario a otra página web. En nuestro caso, la idea es redireccionar el navegador a la página **listado.php**, de modo que podamos comprobar que el resultado de la inserción en la base de datos ha sido correcto.

altas.php

```
<?php
// Estableciendo la conexión
include("conexion.php");

// Recogida de datos
$id_entidad=$_POST['id_entidad'];
$id_cita=$_POST['id_cita'];
$nombre_clinica=$_POST['nombre_clinica'];
$fecha_cita=$_POST['fecha_cita'];
$descripcion=$_POST['descripcion'];


// creamos la sentencia SQL
$ssql="insert into gestorcitas(id_entidad, id_cita, nombre_clinica,
fecha_cita,
descripcion)values('$id_entidad','$id_cita','$nombre_clinica','$fech
a_cita','$descripcion')";

// ejecutamos la sentencia SQL
mysql_query($ssql,$descriptor);

// redireccionar a la web listados
header("LOCATION:listado.php");

?>
```

6.3. Bajas de registros en una tabla

6.3.1. Formulario de bajas

Ya se ha visto como mostrar listados de una base de datos y proceder a dar de alta nuevos registros. Ahora el objetivo es dar de baja, o lo que es lo mismo a eliminar, determinados registros.

Para ello basta con un formulario que recoja el dato o datos por los que se desea dar la baja de un registro de la tabla, y una página que recoja ese dato y efectúe dicha baja.

En nuestro caso se ha creado un formulario que envía los datos a la página **bajas.php** mediante el método **POST**. Se recogerá del formulario dos datos que corresponderán con el campo **id_entidad** y el campo **id_cita**, para posteriormente dar de baja el registro cuyo campo contenga dichos datos.

Observa la visualización del formulario.

Figura 14: Formulario de bajas

6.3.2. Ejecución de bajas

La página **bajas.php** recibirá el id_entidad y el id_cita, efectuará la baja del registro asociado a dichos campos y devolverá el listado de citas.

Lo primero se establece la conexión a la base de datos.

Al igual que se hizo antes, se efectúa la recogida de datos del formulario, pero esta vez únicamente con dos campos.

A continuación se crea la sentencia SQL, que borrará todos aquellos registros de la tabla **gestorcitas** que cumplan la condición. En este caso sólo uno.

Se realiza la ejecución mediante la función **mysql_query**.

Para finalizar, se emplea de nuevo la instrucción **HEADER** para redireccionar el navegador a la página **listado.php**, de modo que se pueda comprobar que el resultado del borrado en la base de datos ha sido correcto.

6.4 Modificación de registros

6.4.1 Formulario de selección de registro

A continuación se muestra cómo se modifican los registros de la tabla gestorcitas. Para ello se construye un primer formulario que permita seleccionar un registro de la tabla, y después un segundo formulario que permita modificar los datos de ese registro.

En el siguiente código se puede ver la construcción del primer formulario, **formulario_seleccion_registro.php**, el cual se limita a recoger un código y enviarlo a una página llamada **modificacion.php**.

Se puede ver cómo queda el formulario de selección de registro.

Figura 16: Formulario de selección de registro

6.4.2. Formulario de modificación

El segundo formulario recoge los campos introducidos en el formulario anterior, y permite al usuario introducir los valores del resto de campos a modificar.

Este nuevo formulario contiene los campos sobre los que se quiera permitir la modificación. En nuestro caso, se usa el id_entidad e id_cita, que estarán desactivados, el nombre clínica, fecha_cita y descripcion.

También se incorpora en este formulario el "script" necesario para que envíe los datos, vía **POST**, a la página **modificación.php**.

Se puede ver en esta vista previa como queda el formulario de modificación.

Figura 17: Formulario de modificación

Se añade la línea necesaria para la conexión a la base de datos.

Se crea el script para la recogida de datos. En este caso tenemos el **id_entidad** y el **id_cita** recogido en la variable global del método POST

Ahora se crea una sentencia SQL que lea de la base de datos, aquel registro que coincida con nuestro **id_entidad** e **id_cita** y se realiza la ejecución de la sentencia SQL asignando el resultado a una variable.

Por último queda la asignación de los valores leídos de cada campo del registro, en la propiedad **value** de cada campo del formulario.

6.4.3 Ejecución de modificación

El formulario anterior recogía los datos desde **formulario_seleccion_registros.php**, permitía modificar los campos y los enviaba a **modificacion.php**.

Ahora se va a explicar paso a paso esta última página web, **modificacion.php**, que es la que finalmente grabará la modificación en la tabla.

Lo primero es establecer la conexión a la base de datos.

El segundo paso es recoger los datos como hemos hecho en anteriores ocasiones.

El tercer paso es crear la sentencia **SQL**, utilizando la instrucción **UPDATE** que actualizará todos los valores de los campos cuyo id_entidad e id_cita coincida con el nuestro.

EL cuarto paso es la ejecución de esa sentencia **SQL**.

El quinto paso será redireccionar la página a listado.php para comprobar si los cambios han tenido efecto.

6.4.4 Prueba de modificación

Se empieza ejecutando en nuestro navegador la página con el formulario de selección del registro.

El formulario de modificación recoge el id_entidad e id_cita introducidos y rellena el resto de campos asociados a ese código.

Por ejemplo se puede modificar la descripción cuyo valor actual es “limpieza bucal”.

Figura 18: Prueba de modificación 1

Se realiza la modificación de la descripción, ahora el nuevo valor es “Empaste”.

A continuación se pulsa el botón “Enviar”

Figura 19: Prueba de modificación 2

AL pulsar en **Enviar**, se ha grabado la modificación en la base de datos, y se redirecciona a la página web **listado.php**, donde se puede observar dicha modificación.

Figura 20: Listado

6.5 Actuaciones recursivas

6.5.1 Idea

Se trata de una técnica que se puede emplear en páginas web, minimizando de este modo el número de páginas de un sitio web.

La idea es que el formulario permite definir los datos de búsqueda y los envía a la misma página donde se encuentra para resolver el listado, produciendo una retroalimentación de la página.

En este caso, en la misma página donde se dan de alta las citas, se comprueba el listado de las mismas que se van insertando en la tabla.

6.5.2 Sección de listado

La página formulario.recusivo.php contiene dos secciones, un listado y un formulario.

El primer paso es crear la conexión a la base de datos.

El segundo paso es crear la sentencia **SQL** y su ejecución.

El tercer paso es hacer un recorrido por el "array" para mostrar todos los datos.

6.5.3 Sección de formulario

La segunda sección de la página web contiene un formulario para introducir datos y actualizar de este modo, el listado de citas.

Este formulario es similar al de altas de citas pero en este hay que destacar dos detalles. El primero, el parámetro **ACTION**, que en este caso, en lugar de enviar los datos del formulario a otra página, los envía a sí misma. El

segundo es el botón **Enviar**, al cual tiene el nombre de **botonenviar**, y que actuará como llave para el siguiente paso.

6.5.4 Condicional

Únicamente falta escribir el código para la inserción de los datos introducidos en el formulario en la tabla **gestorcitas**. Como el tratamiento de estos datos se hace en la misma página web, es necesario establecer la condición para su ejecución.

Esta condición se hace al principio de la página, mediante la función **ISSET**, la cual comprueba si existe la variable que llega vía **POST**, llamada **botonenviar**.

En caso afirmativo se ejecuta todo el código encerrado entre llaves que produce la inserción de los datos en la tabla **gestorcitas**. En caso contrario seguirá con la ejecución del resto del código escrito en la página.

Para finalizar, se puede ver en la ejecución de la página web como en la parte superior aparece el listado de citas, y en la parte inferior se encuentra el formulario para la inserción de nuevas citas, de manera que cualquier nueva cita que insertemos, se irá incorporando al listado.

Figura 20: Formulario recursivo

6.6. Creación de Menú

6.6.1 Menú

Para organizar las páginas que hemos desarrollado, vamos a crear un menú que será la raíz de nuestro sitio web, y que permitirá acceder de forma directa a los distintos formularios y listados. Para ello crearemos una página llamada **menu.php**.

menu.php

```
<html>
      <head></head>
      <body>
            <p><a href="listado.php">Listado de Citas</a></p>
            <p><a href="formulario_altas.php">Altas de Citas</a></p>
            <p><a href="formulario_bajas.php">Bajas de Citas</a></p>
            <p><a
href="formulario_seleccion_registro.php">Modificacion de
Citas</a></p>
            <p><a href="formulario_recursivo.php">Formulario
Altas(Recursivo)</a></p>

</body>
</html>
```

Comprobamos la página web creada:

Figura 22: Menú

6.7 Control de acceso

6.7.1 Incorporación de sesión

Una sesión en el entorno de desarrollo web consiste en controlar el acceso individual a una página o crear sistemas de autentificación en páginas restringidas mediante usuario y contraseña.

En nuestro caso se ha creado un sencillo sistema de acceso a una página web mediante usuario y contraseña, en la que se ha programado un control de entrada que validará las credenciales del usuario. Si no son correctas se le denegará el acceso; si son correctas lo autentificará en el sistema y se le asignará una variable global.

En cada página se comprobará si la variable global de sesión es correcta. En caso contrario lo remitiría nuevamente a la página de introducción del usuario y contraseña.

Comenzamos abriendo el gestor de base de datos **phpMyAdmin**. Se ha creado una nueva tabla para usuarios con dos campos a los que se ha llamado usuario y clave, los dos de tipo **VARCHAR** y con una longitud de 10 caracteres.

El campo usuario se ha marcado como "clave primaria".

Se pueden dar de alta tantos usuarios como se quiera, una vez terminada el alta se debe guardar el registro.

Seguidamente se ha creado nuestra página web de entrada. Se trata de un formulario para introducir el usuario y la contraseña, y que envía, vía GET, dichos datos a otra página llamada **activacion.php** que recogerá los campos usuario y clave.

Se puede observar como en la primera línea de cada página donde vayamos a controlar sesión, como en este caso se debe incluir la instrucción **sesión_start().**

El control de la sesión se realiza mediante una instrucción **IF** que comprueba si no existe la variable global **ACCESO** o su valor es **FALSE**, en cuyo caso nos muestra el formulario de introducción de usuario y contraseña. En el

caso de que la sesión sea correcta, estaremos autentificados y se mostrará un mensaje de bienvenida y un vínculo para que nos podamos desconectar.

entrada.php

```
<html><head></head><body>
<?php
      session_start();

      if(!isset($_SESSION['acceso']) or $_SESSION['acceso']==false){

?>
      <form action="activacion.php" method="get">
            <p>Usuario<input name="usuario" type="text"/><br>
                  Clave<input name="clave" type="password"/><br>
                  <input type="submit" name="button" value="Enviar"/>
                  <input type="reset" name="button2" value="Borrar"/>
            </p>
      </form>

<?php
      }elseif($_SESSION['acceso']==true){
            echo "Bienvenido al sistema ";
?>
<a href="desactivacion.php"> Desconectar</a>

<?php } ?>

</body>
</html>
```

Al ejecutar la página web en la web en el navegador, se puede ver el formulario de introducción de usuario y contraseña porque la condición inicial establecida sobre la variable global se cumple, es decir, el usuario todavía no ha sido autentificado.

Figura 23: Entrada

La página **activacion.php** recibirá los datos de la página de entrada que contiene el formulario de introducción de usuario y contraseña, y los comprobará.

Los primeros pasos a realizar son establecer la conexión a la base de datos, iniciar una sesión, y poner la variable **ACCESO** a **FALSE**.

Después se recogen los datos con el modo habitual, y mediante una sentencia **SQL** se busca el usuario y la clave en la tabla usuarios.

Si hay coincidencia, se pone la variable **ACCESO** a **TRUE**; si no la hubiese se pondría a **FALSE**, aunque no sería necesario porque inicialmente ya se puso a ese valor. Por último se redirecciona la web a la página de entrada.

activacion.php

```
<html><head></head><body>
<?php
include("conexion.php"); // Establece la conexión
session_start();
$_SESSION['acceso']=false; // pone la variable acceso a FALSE

$v01=$_GET['usuario']; // Recogemos los datos
$v02=$_GET['clave'];

// Buscamos el usuario y la clave en la tabla usuarios
$ssql="select * from usuarios where usuario='$v01' and
clave='$v02'";
```

```
mysql_query($ssql,$descriptor);
$resultado=mysql_query($ssql,$descriptor);
$row=mysql_fetch_array($resultado);

if($row['usuario']<>null){ // Si hay coincidencia ponemos la
variable ACCESO a TRUE
      $_SESSION['acceso']=true;
      }else{
            $_SESSION['acceso']=false;}
            header("LOCATION:entrada.php");   // redireccionamos la
web a la página de entrada
?>

</body>
</html>
```

En cuanto a la página de desactivación, lo único que hace es que la variable global **ACCESO** la pone a **FALSE** y nos redirecciona de nuevo a **entrada.php**.

desactivacion.php

```
<html><head></head><body>
<?php
include("conexion.php"); // Establece la conexión
session_start();
$_SESSION['acceso']=false; // pone la variable acceso a FALSE

$v01=$_GET['usuario']; // Recogemos los datos
$v02=$_GET['clave'];

// Buscamos el usuario y la clave en la tabla usuarios
$ssql="select * from usuarios where usuario='$v01' and
clave='$v02'";

mysql_query($ssql,$descriptor);
$resultado=mysql_query($ssql,$descriptor);
$row=mysql_fetch_array($resultado);

if($row['usuario']<>null){ // Si hay coincidencia ponemos la
variable ACCESO a TRUE
      $_SESSION['acceso']=true;
      }else{
            $_SESSION['acceso']=false;}
            header("LOCATION:entrada.php");   // redireccionamos la
web a la página de entrada
?>

</body>
</html>
```

A continuación se muestra un trozo de código que debe ir al inicio de todas aquellas páginas del sitio web en la que se quiera controlar la sesión.

Actuaría como puerta de entrada comprobando si el usuario está autentificado, en cuyo caso continuaría en la página. En caso contrario, lo devolvería a la entrada para que introdujese su nombre de usuario y contraseña.

acceso.php

```
<?php
session_start();
if(!isset($_SESSION['acceso'])or $_SESSION['acceso']==false){
      header("LOCATION:entrada.php");
      }
      else{
            echo"Bienvenido al sistema";
            ?>
            <a href="desactivacion.php">Desconectar</a>
            <?php
            }
            ?>
<html><head></head><body>
</body>
</html>
```

Por último se puede observar la estructura completa del sitio web. Ayuda a entender el flujo de información que se está produciendo entre el usuario y la base datos **MySQL**.

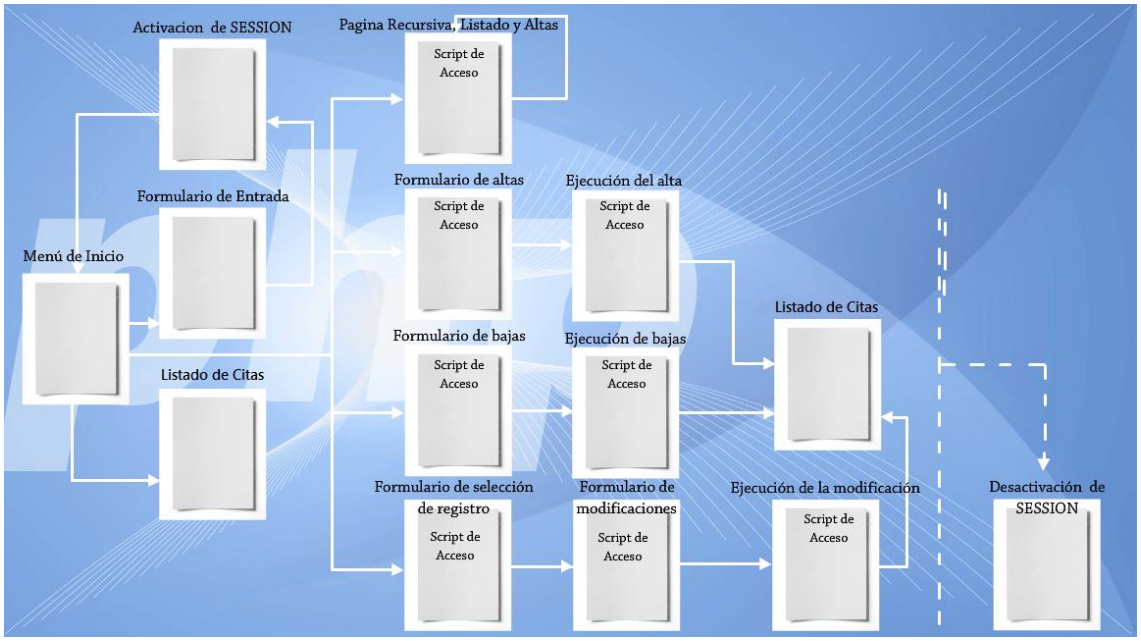

Figura 24: Estructura completa del sitio web

Capítulo 7.

Implementación de la aplicación Zwait arquitectura y desarrollo

7.1 Requisitos de la aplicación

1. Plataforma

Los requisitos para una correcta instalación y uso de la aplicación Zwait serán:

-La aplicación desarrollada deber ser compatible con el mayor número posible de dispositivos, por lo que he elegido como plataforma un Smartphone o Tablet con sistema operativo Android 2.2 (Froyo) o superior (API 8). Para que la aplicación sea compatible con prácticamente todos los dispositivos Android del mercado, no siendo determinante el tener un móvil medianamente moderno.

-El espacio disponible de al menos 1,4MB.

2. El Servidor Web

Quiero que el servidor web procese la aplicación del lado del servidor, realizando conexiones bidireccionales o unidireccionales y asíncronas con el cliente. En esta ocasión estoy utilizando EasyPHP, el cual contiene el servidor web Apache *2.4.4*. Se deberá configurar el archivo httpd.conf (figura 6) con la instrucción "listen", la cual le indica al servidor que escuche todas las llamadas que entran por la dirección IP 127.0.0.1 a la que denominaré como "localhost".
Dentro de la carpeta localweb, alojada dentro del programa EasyPHP, crearé las carpetas referentes al servidor web; y en el interior de éstas, los archivos PHP, que permiten acceder a la base de dados.

3. La Base de Datos

La base de datos que utilizo es MySQL *5.6.11* que, al igual que el servidor web, viene instalado con EasyPHP. Accederé a ella de manera local a través de localhost. Para ello, utilizaré como nombre “root” y ninguna contraseña. En MySQL crearé dos bases de datos; una llamada “android_api”, en la que se almacenará el registro de los usuarios; y otra llamada zwait, donde almacenaré el contenido de los mensajes creados por los usuarios. La función que espero de esta base de datos es que me almacene los datos de manera sistemática para su posterior uso.

7.2 Arquitectura de la Aplicación

Me conectaría a una base de datos MySQL (3) remota desde dentro de mi aplicación (1). Pero por desgracia, una base de datos MySQL y una aplicación para Android no se pueden comunicar directamente, por lo que crearé un sencillo servicio web Apache basado en PHP, que permitirá a mi aplicación de Android insertar datos en la base de datos y acceder a ellos cuando quiera. El sistema constará de una parte Android, que será el cliente que se conectará a un servidor. En este caso, será con un servidor Apache (2) con PHP.

Por ejemplo, si quiero crear un nuevo usuario en la aplicación desde mi dispositivo Android (1), tendré que introducir algunos valores de entrada, como el nombre de usuario y la contraseña. Cuando el usuario rellena esta información, se pasa la información a nuestro servicio web PHP. El servicio web se conectará a la base de datos MySQL, y buscará la tabla "Usuarios", por ejemplo, e insertará una nueva fila en la base de datos MySQL con la información que se envió desde nuestro dispositivo Android. Después, si queremos utilizarlos, manipularemos los datos que nos envía el servicio web en JSON para poder mostrarlos en Android, en un ListView, por ejemplo.

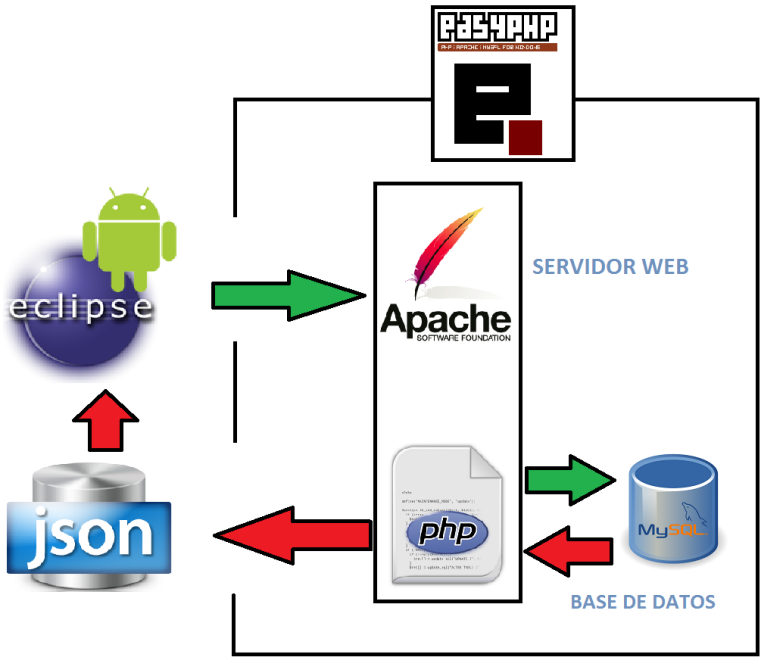

Figura 25: Arquitectura de la aplicación

7.3 Desarrollo del servidor

Voy a hablar del servidor web Apache. Como he mencionado en el punto anterior, lo haremos sobre php. Los archivos php con los que vamos a trabajar son los siguientes:

Archivos de inicio de sesión y registro. Carpeta:"android_api":

config.php.: Aquí pondremos nuestra configuración para poder conectar a la base de datos.

```
<?php
define("DB_HOST", "localhost");
define("DB_USER", "root");
define("DB_PASSWORD", "");
define("DB_DATABASE", "android_api");
?>
```

DB_Connect.php: Es el encargado de conectar a la base de datos o desconectarnos.

DB_Functions.php: contiene métodos para insertar o leer datos de la base de datos, como por ejemplo, para insertar el usuario que al crearlo devuelve

un array con los nuevos datos; o buscarlo por email y contraseña, y comprobar si el usuario ya existe. También genera un identificador de usuario único y encriptará la contraseña utilizando el método **base64encode.**

index.php, él aceptará todas las peticiones GET y POST. Lo utilizaremos para gestionar todas las peticiones con la BD y dar respuesta en formato JSON.

La otra parte de los archivos php de la aplicación estará situada en la carpeta "Android_connect":

db_Config.php y db_connect.php : mismas funciones que las anteriores.

crear_cita_clinica.php: creará o insertará una nueva cita (fila) en su tabla de la base de datos con todos sus campos (id_entidad, id_cita, nombre_clínica, fecha, y descripción).

crear_cita_cliente.php: creará o insertará una nueva cita (fila) en su tabla de la base de datos con solo dos campos (id_entidad, id_cita). La idea es que el paciente pueda darse de alta y sea la clínica la que rellene el resto de campos.

get_all_citas_clinica.php: devuelve a la clínica una lista con todas sus citas.

get_all_citas_cliente.php: devuelve una lista al cliente con todas las citas.

get_detalles_citas.php: vale para obtener los detalles individuales de las citas, las citas se identifican por su ID.

update_cita.php: actualizará La información de una cita, las citas se identifican por su ID.

delete_cita: borrara una cita de la tabla de citas, las citas se identifican por

7.4 Desarrollo del cliente

7.4.1 Desarrollo del cliente/paciente

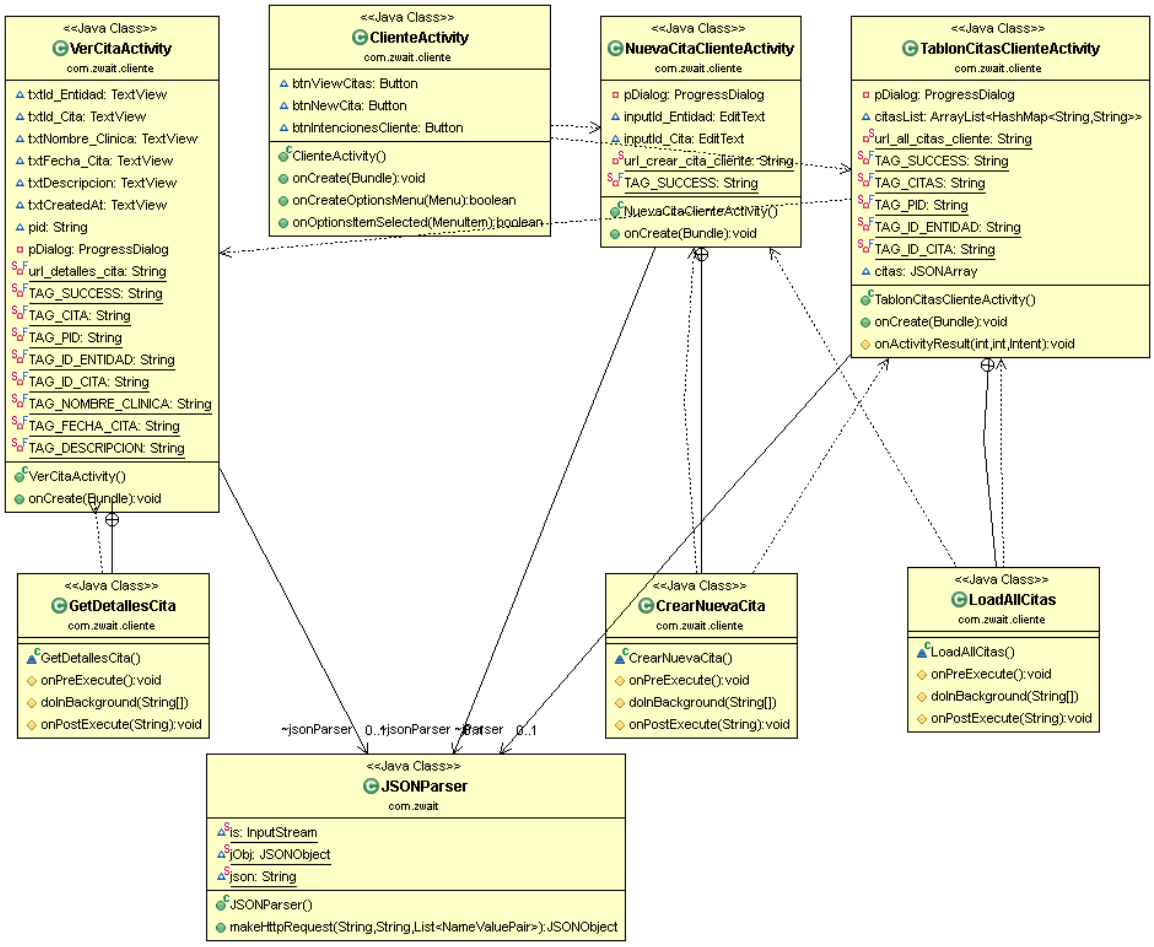

Figura 26: Diagrama UML de clases del cliente

7.4.2 Desarrollo de la clínica (cliente de la aplicación)

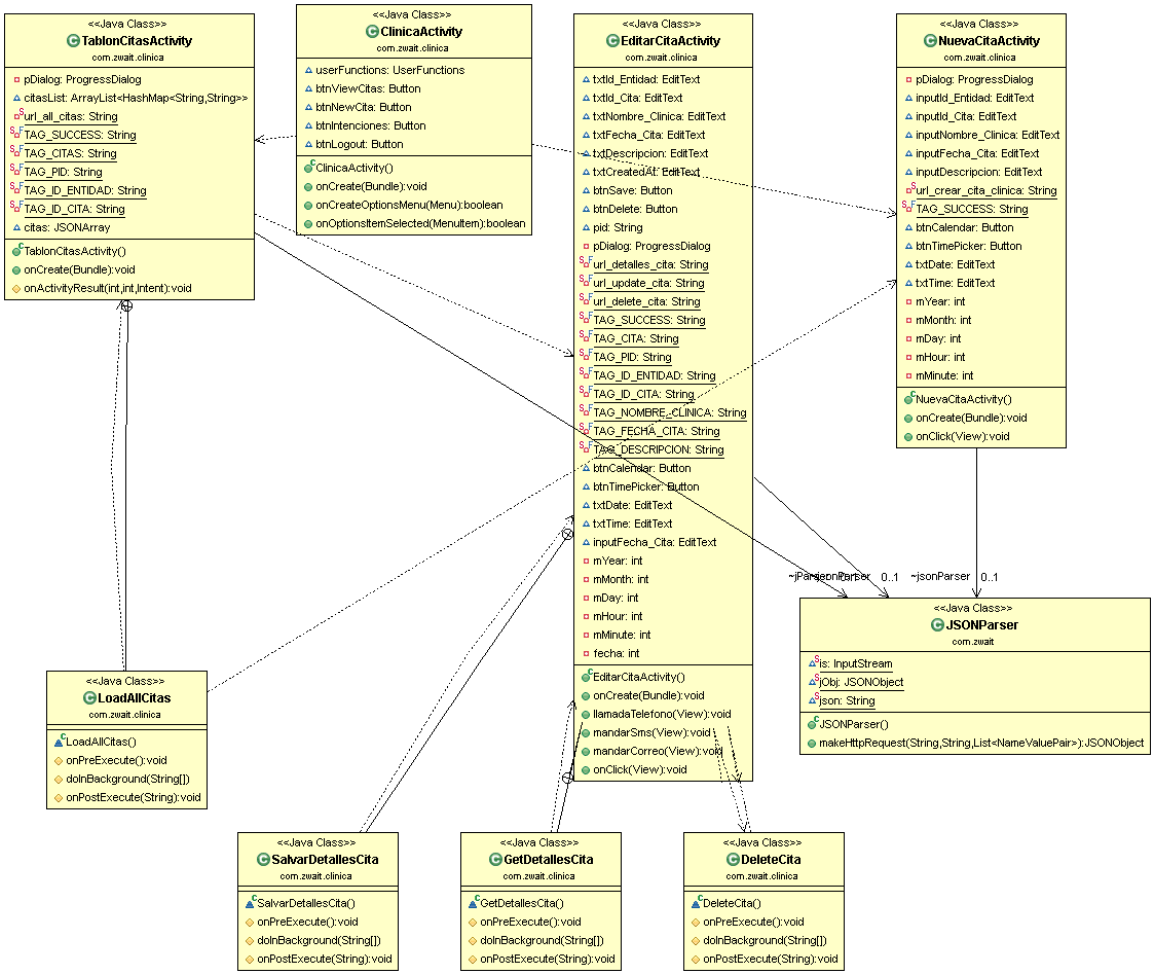

Figura 27: Diagrama UML de clases de la clínica

Antes de entrar en detalles, daré un vistazo global de los diagramas UML (figuras 25 y 26) y del desarrollo del cliente de la aplicación.

Además comentaré las clases JSON Parser, las cuales utilizo para analizar las respuestas api JSON. Y con esa respuesta, la aplicación y el usuario conocerán si todo ha transcurrido correctamente o si, por el contrario, algún campo está vacío, o si el email y el password son incorrectos.

La otra clase admite dos métodos de solicitud HTTP GET y POST utilizados para obtener los datos json de las url. Éstos nos ayudarán a enviar y recibir información de la base de datos.

También comentaré el uso de la clase AsyncTask, ampliamente explicada en un punto anterior (pág. 38), con la que crearemos tareas en segundo plano para poder conectarnos al servidor, y realizar tareas como crear una cita, editarla o eliminarla.
A continuación pasaré a explicar en detalle las actividades y el funcionamiento.
Lo primero que nos encontraremos, tras instalar la aplicación, será el icono de Zwait. Después de pulsarlo pasaremos a la siguiente actividad.

Figura 28: ic_launcher.png

Figura 29: activity_splash.xml

Figura 30: menu.xml

Figura 31: cliente.xml

7.4.3 Actividades de login y registro de cuenta.

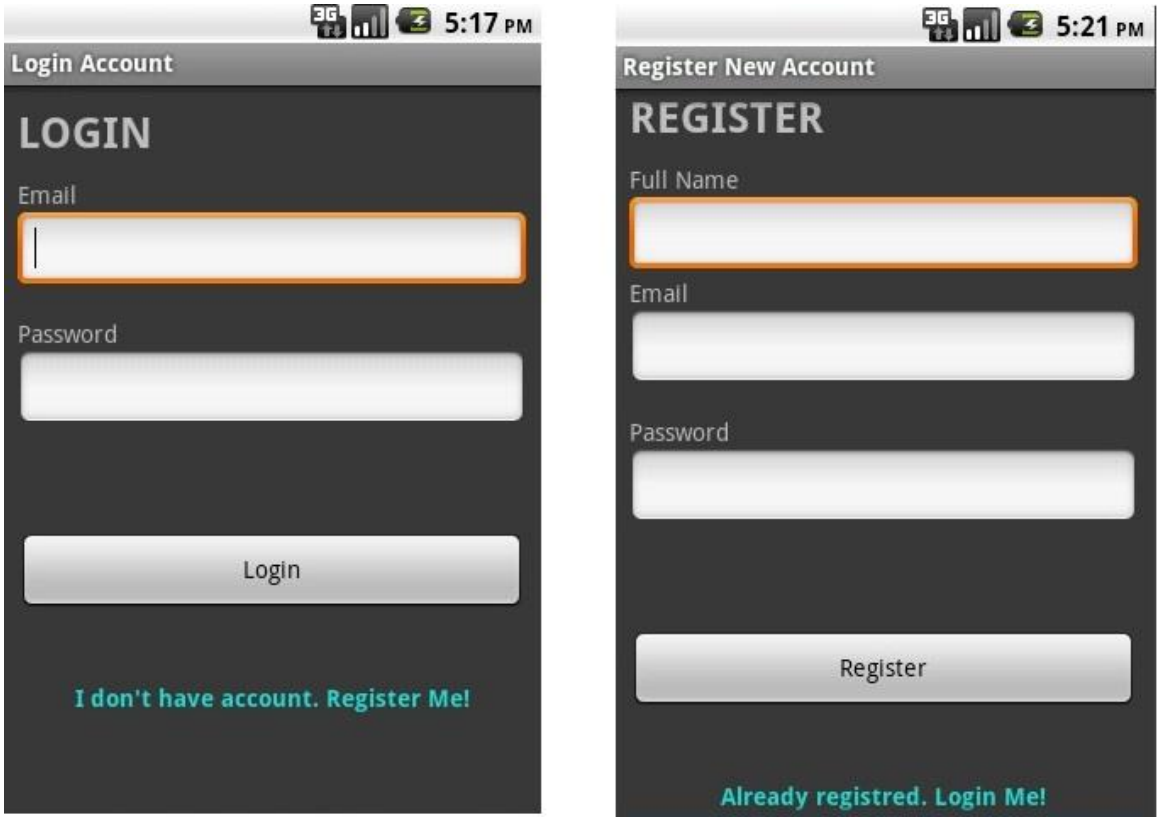

Figura 32: login.xml y register.xml

LoginActivity.java - Actividad para manejar eventos de inicio de sesión. Con el botón Login (login.xml) pasamos el contexto de la activity, un email y una contraseña. Esta actividad se encargará de enviarlos al servidor para validar el proceso y almacenaremos los detalles del usuario en una base de datos SQlite.

Una vez logueados, y si todo se ha producido correctamente, nos llevará a la pantalla principal de la aplicación.

RegisterActivity.java - En el caso de que no estemos registrados pulsaremos otro botón (¿No tiene cuenta? registrarse) que nos llevará a la pantalla de registro para crear un nuevo usuario. Introduciremos el nombre, el email y la contraseña, la cual será encriptada, y nos mandará directamente a la pantalla principal de la aplicación.

Clases utilizadas que se encuentran en la librería:

UserFunctions.java
Funciones de la clase UserFunctions.java

loginUser(): solicitud de logueo de usuario, parámetros (email y password).

RegisterUser (): solicitud de registro de usuario, parámetros (name, email y password).

IsUserLoggedIn (): comprueba en nuestra aplicación si ya hay un usuario registrado.

LogoutUser (): elimina la información almacenada de un usuario en la BD del dispositivo.

DatabaseHandler.java: en la aplicación se almacena la información del usuario que se registra usando la base de datos no remota SQlite.

JSONParser.java: utilizado para analizar las respuestas api JSON, las cuales nos indican, por ejemplo, si un usuario se ha almacenado correctamente o no; si el usuario ya existía; o si se ha equivocado al introducir el email y el password.

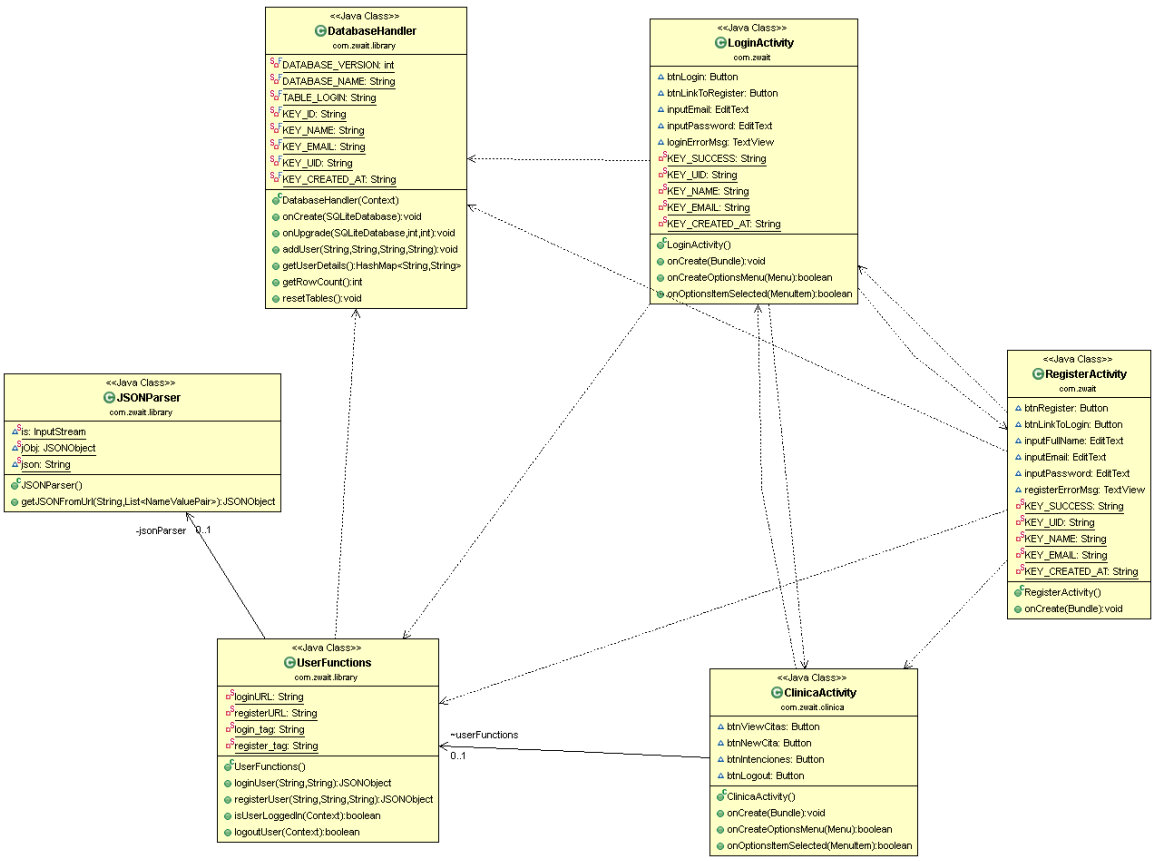

Figura 33: UML login y registro

En la figura anterior (figura 32) represento las clases y funciones explicadas en el punto anterior, las cuales utilizo durante el registro, el logueo de la aplicación y la relación que existe entre ellas.

ClinicaActivity.java

Tras iniciar sesión o registrarnos de manera correcta pasamos a la siguiente actividad **ClinicaActivity.java** clase principal, que da función a los botones a través de eventos para movernos por otras actividades de la aplicación, como **TablonCitasActivity.java, NuevaCitaActivity.java e IntencionesAct.java** o para cerrar la sesión. Si cerramos la sesión, al volver a abrir la aplicación habrá que volver a introducir el correo y el password.

Figura 34: clinica.xml y UML

7.4.4 TablonCitasActivity.java

Ahora vamos a crear una actividad que muestre todas las citas que tanto la clínica como los pacientes han enviado en forma de lista o tablón. Para hacer cualquier tipo de lista en Android necesitaremos dos archivos XML; uno que contenga un listview; y otro que sea un elemento de esa lista. Ese elemento o fila de esa lista contendrá la información que queremos que aparezca de cada cita creada por el usuario.

En mi caso, dispongo de 2 campos: id_entida e id_cita que identifican unívocamente cada cita.

Figura 35: tablon_citas.xml

Desde Android accederemos al servidor web enviando una solicitud o petición en segundo plano AsyncTask, explicada en otro punto de la memoria al fichero php get_all_citas.php. Solicitando la lista con todas las citas y sus respectivos datos.

Estos datos que recibiré de mi petición php los “parsearé” a un objeto JSONObjects y los adaptaré al listview para poder mostrarlos.

En el caso de que no exista ninguna cita en el tablón lanzaremos la actividad NuevaCitaActivity.java

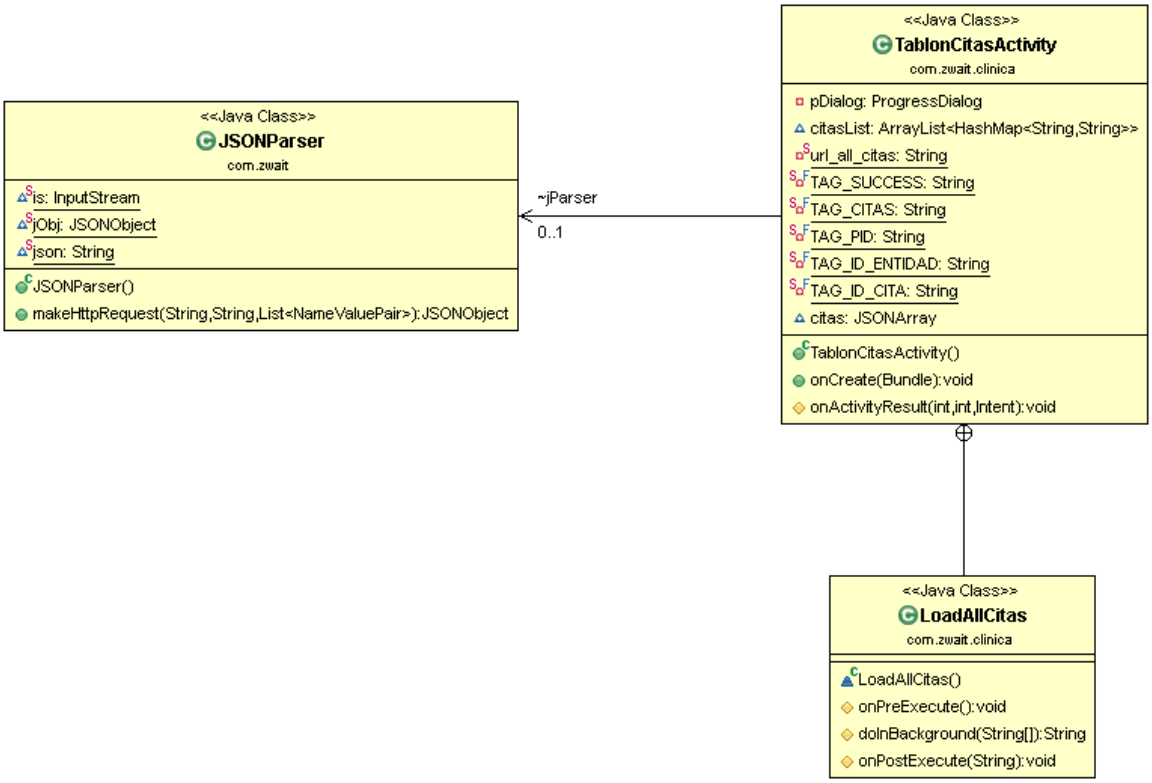

Figura 36: UML TablonCitasActivity.java

7.4.5 VerCitaActivity.java

Cuando pulsemos un elemento de la lista se nos pasará a otra actividad en la que se nos mostrará la cita con todos los detalles y campos que nos faltaban. Los cuales nos describen más detalladamente el nombre de la clínica, la fecha y hora de la cita y una descripción de la misma. El servidor web y el archivo php get_detalles_cita.php me devolverá en formato JSON los detalles de la cita para poder utilizarla en mi app.

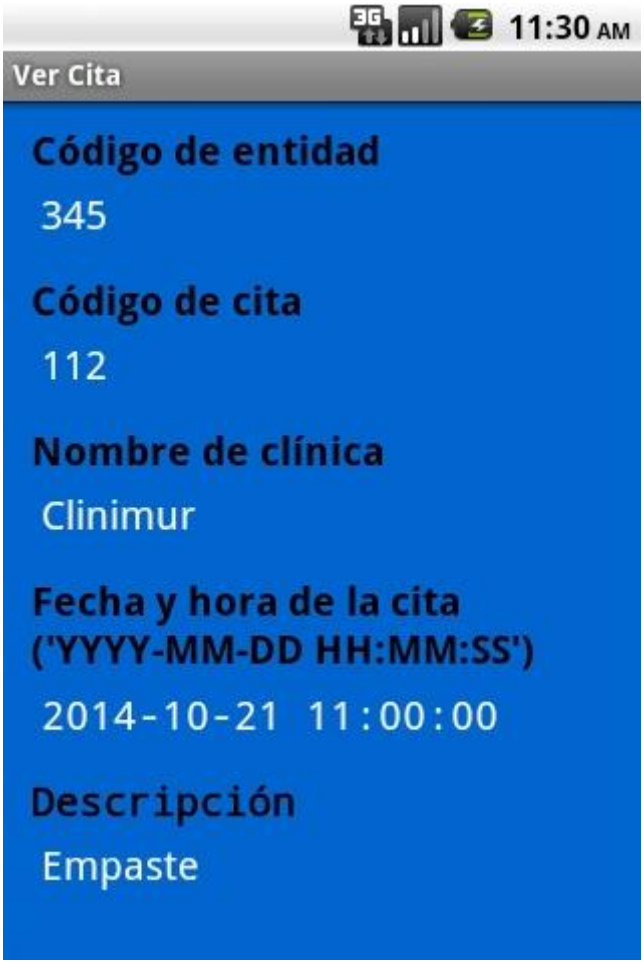

Figura 37: ver_detalles_cita.xml

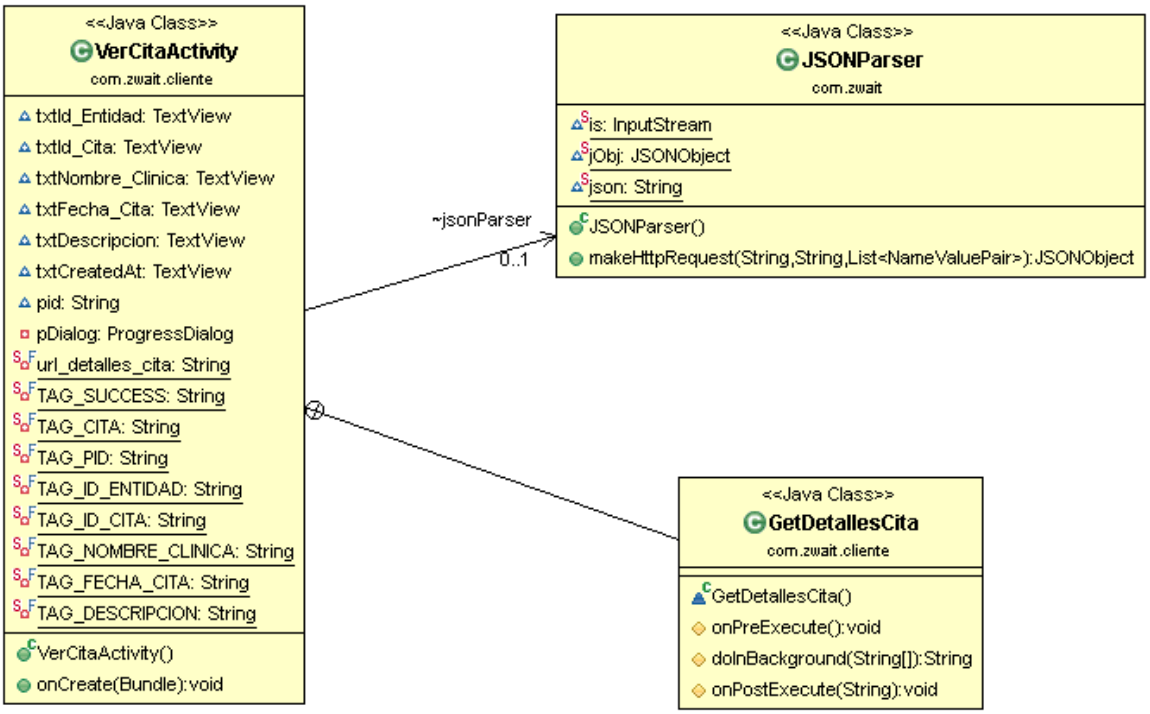

Figura 38: UML VerCitaActivity.java

7.4.6 NuevaCitaActivity.java

En el caso de que apretemos el botón "Añadir Nueva Cita", de la actividad ClinicaActivity.java, pasaremos a la actividad NuevaCitaActivity.java, donde podremos crear una nueva cita que agregar al tablón de citas, donde la clínica rellenará todos los campos.

Cuando se pulse el botón crear nueva cita, llamará al método CrearNuevaCita ().

Nuestros datos se leerán en forma EditText y realizaremos una solicitud a add_cita.php para crear una nueva cita con todos esos parámetros a través de HTTP POST. Cuando recibamos una respuesta json de add_cita.php, si se realiza con éxito (el bit es 1), pasaremos a la actividad TablonCitasActivity.java que se actualizará con la nueva cita que acabamos de crear.

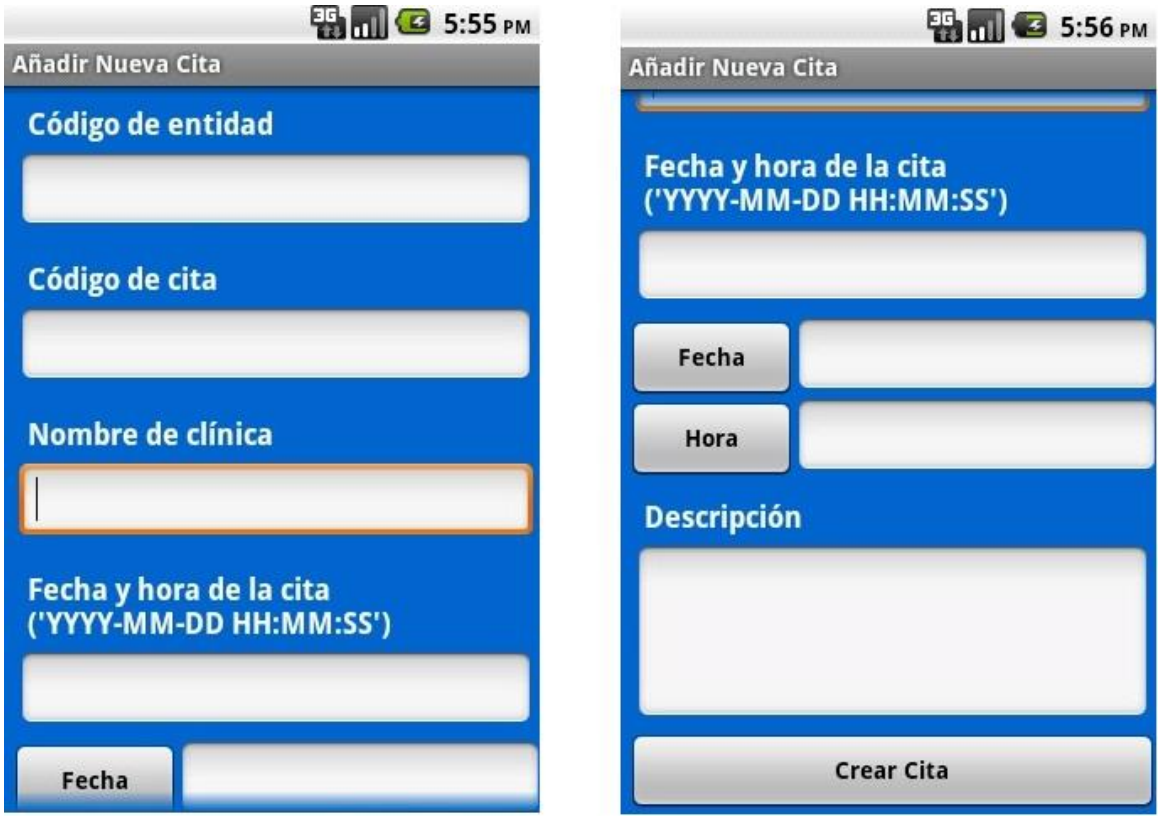

Figura 39: add_cita.xml

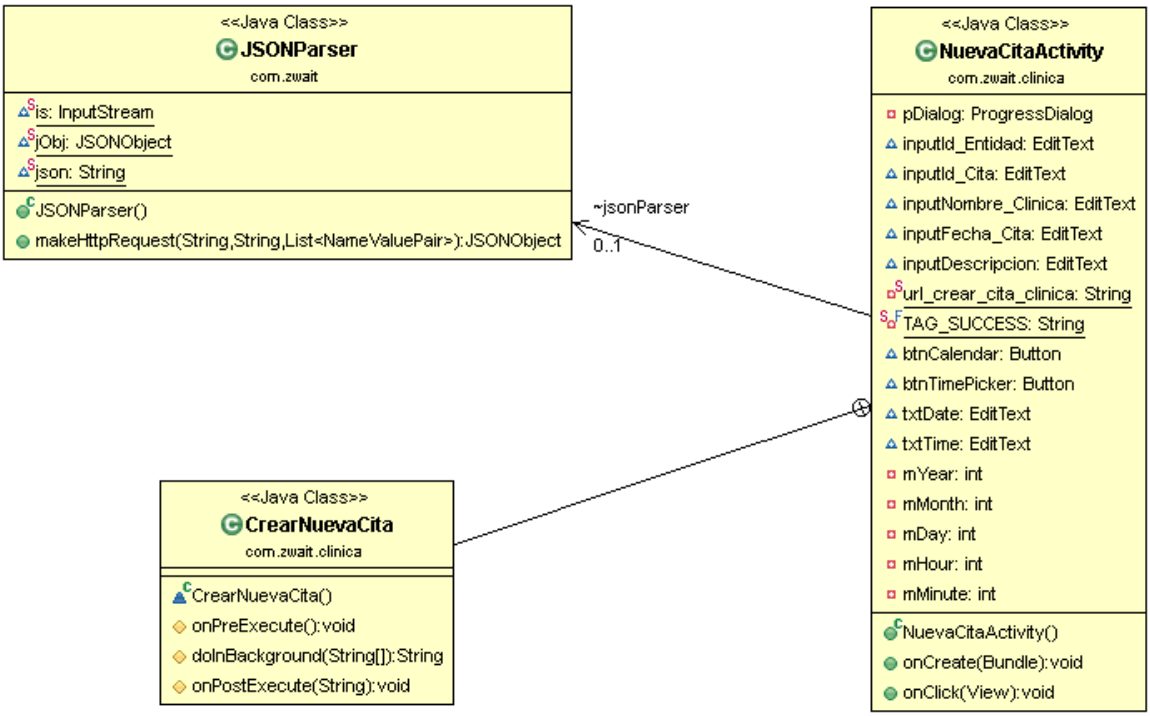

Figura 40: UML NuevaCitaActivity.java

7.4.7 EditarCitaActivity.java

Esta última actividad será la encargada de modificar las citas almacenadas en la base de datos, actualizarlas en el caso de que se nos haya pasado añadir algo, o eliminarla.

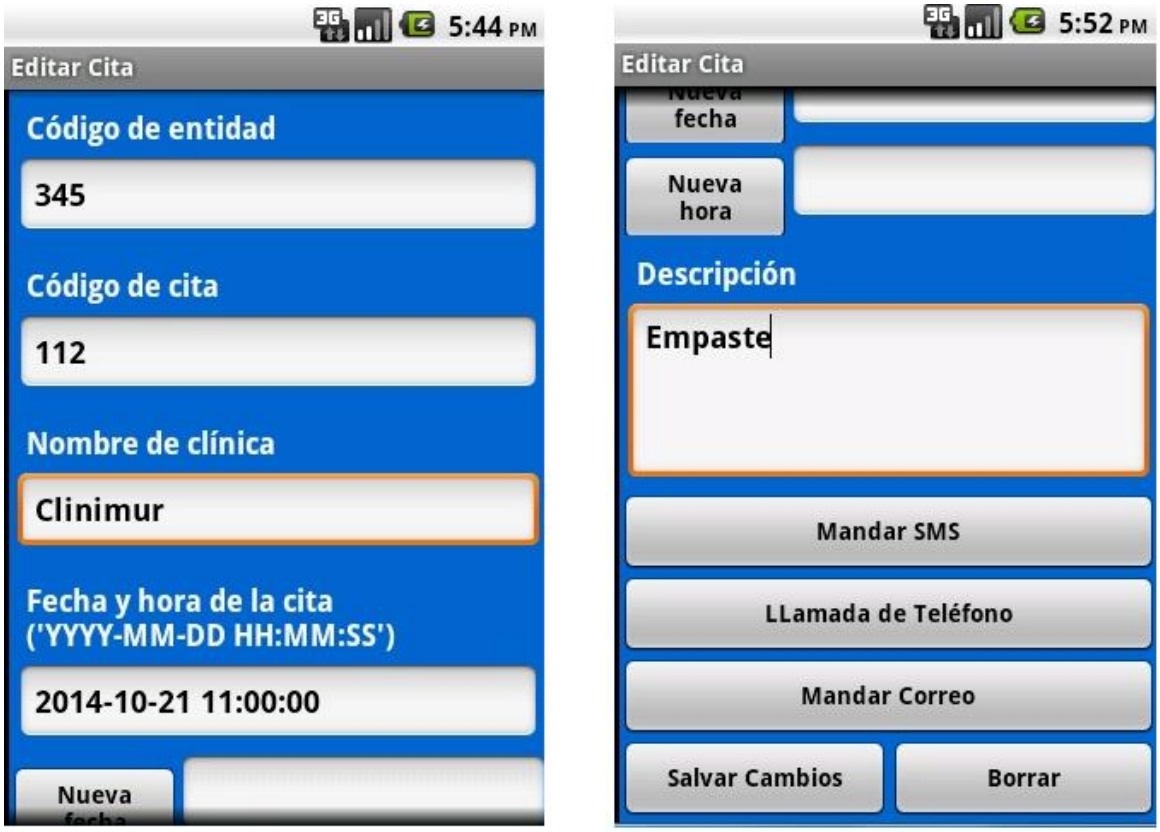

Figura 41: editar_cita.xml

Obtenemos la identificación de la cita (PID) de la intención que ha comenzado la actividad, la cual se envía desde el listview donde estaba la cita.

Comenzaremos realizando una solicitud a get_detalles_cita.php con la obtención de los detalles de la cita en segundo plano y en formato json Mostrándose en varios EditText.

Después de mostrar los datos de la cita, se esperará a que el usuario pulse alguno de los dos botones:

Si el usuario hace click en el botón guardar, se hace una solicitud HTTP a update_cita.php para almacenar los datos de las citas actualizadas. Todo esto se realizará en un segundo plano AsyncTask.

Si el usuario hace click en el botón eliminar, se hace una solicitud HTTP a delete_cita.php, comenzar en segundo plano la eliminación de la cita de la base de datos MySQL y la actualización del tablón de citas.

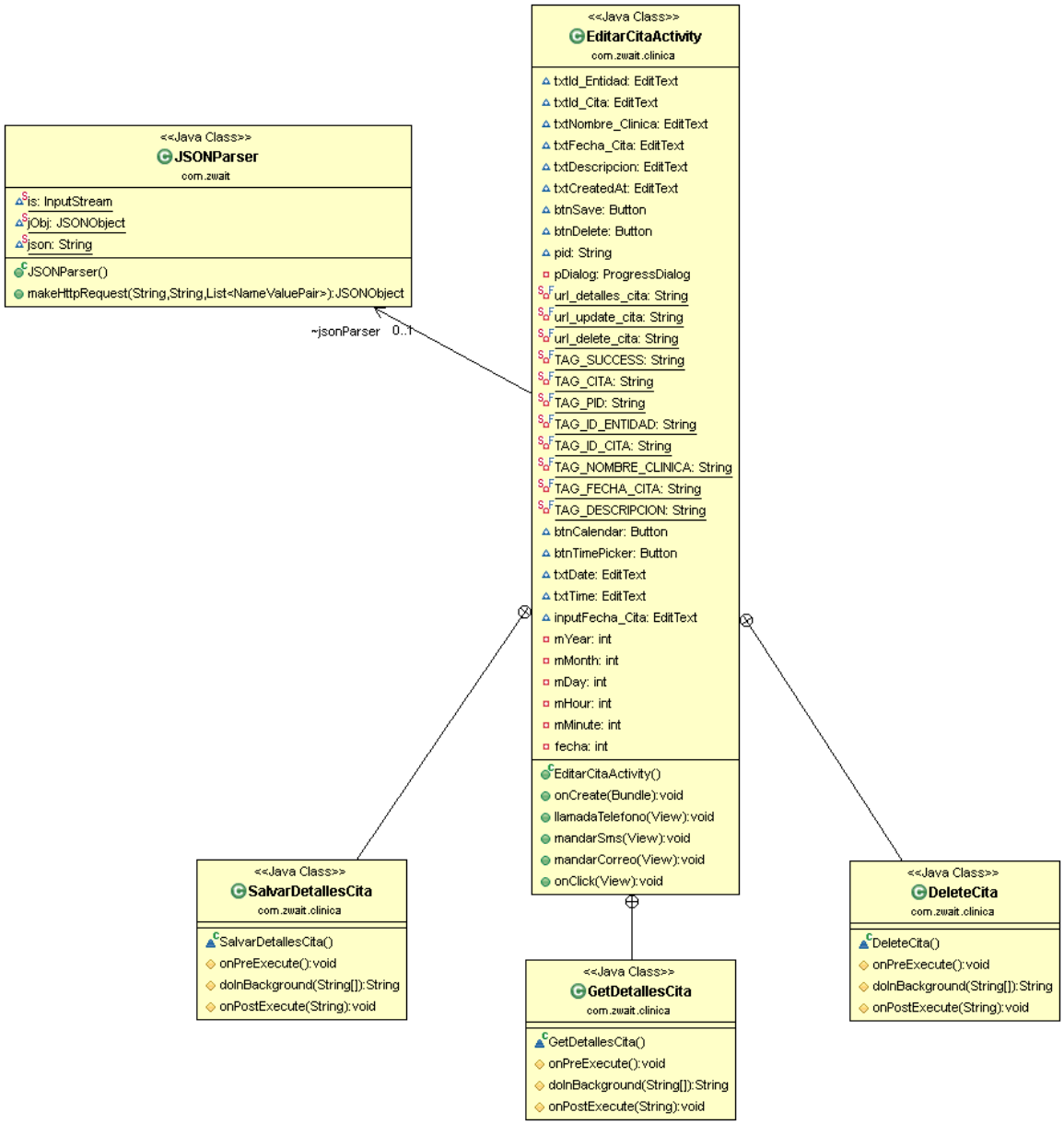

Figura 42: UML EditarCitaActivity.java

7.4.8 JSON Parser class: esta clase admite dos métodos de solicitud HTTP GET y POST para obtener los datos json de las url.

7.5 AndroidManifest.xml

Cada aplicación Android que se desarrolle debe de tener un archivo con el nombre AndroidManifest.xml. Este fichero contiene información sensible sobre la aplicación que el sistema operativo debe conocer y está situado en la raíz de cada aplicación. Esta información es obligatoria para que la aplicación pueda ejecutarse. El icono de la aplicación que aparecerá posteriormente al instalarse en los menús y nombre de la misma se mostrará definida aquí.

Añadiremos los siguientes permisos para que nuestra aplicación pueda conectarse a internet y llamar por teléfono.

```
<!--  Internet Permissions -->
    <uses-permission android:name="android.permission.INTERNET" />

<!--  Call_phone Permissions -->
    <uses-permission android:name="android.permission.CALL_PHONE"/>
```

Entre las informaciones que maneja este archivo podemos encontrar:

- Contiene el nombre del paquete de desarrollo de la aplicación, el cual se utiliza como identificador único de la aplicación dentro del sistema.
- Define cada uno de los componentes de la aplicación, las activities, services, brodcast receivers y los content providers, los cuales componen una aplicación.
- Se declara el nivel mínimo y la versión para la que fue creada de la API, que el terminal Android debe tener instalado.
- Determina qué procesos son los que se encargan de acoger a los componentes implementados por la aplicación.
- Indica los tipos de permisos que debe tener la aplicación a la hora de querer acceder a partes protegidas de la API e interactuar con otras aplicaciones.

Capítulo 8.

Las intenciones

8.1 Definición y características

Una *intención* representa la voluntad de realizar alguna acción o tarea; como realizar una llamada de teléfono o visualizar una página web. Una intención nos permite lanzar una actividad o servicio de nuestra aplicación o de una aplicación diferente.

Existen dos tipos de intenciones:

- **Intenciones explícitas:** se indica exactamente el componente a lanzar. Su utilización típica es la de ir ejecutando los diferentes componentes internos de una aplicación. Por ejemplo, desde la actividad Menu.java lanzamos AcercaDe por medio de una intención explicita.
- **Intenciones implícitas:** pueden solicitar tareas abstractas, como "quiero llamar por teléfono" o "quiero enviar un mensaje". Además las intenciones se resuelven en tiempo de ejecución, de forma que el sistema mirará cuantos componentes han registrado la posibilidad de ejecutar ese tipo de intención. Si encuentra varias el sistema puede preguntar al usuario el componente que prefiere utilizar.

Además, las intenciones ofrecen un servicio de paso de mensajes que permite interconectar datos entre componentes.

En concreto se utilizan intenciones cada vez que queramos:

- lanzar una actividad (startActivity() y startActivityForResult())
- lanzar un servicio (startService())
- lanzar un anuncio de tipo broadcast (sendBroadcast())
- conectarnos con un servicio (bindService())

En muchas ocasiones una *intención* no será inicializada por la aplicación, si no por el sistema, por ejemplo, cuando pedimos visualizar una página Web. En otras ocasiones será necesario que la aplicación inicialice su propia *intención*. Para ello se creará un objeto de la clase ***Intent*.**

Cuando se crea una Intención (es decir, se instancia un objeto de tipo Intent) esta contiene información de interés para que el sistema trate adecuadamente la intención o para el componente que recibe la intención. Puede incluir la siguiente información:

- **Nombre del componente:** Identificamos el componente que queremos lanzar con la intención. Podemos utilizar el nombre de clase totalmente cualificado (com.zwait.AcercaDe) que queremos lanzar. El nombre del componente es opcional. En caso de no indicarse se utilizará otra información de la intención para obtener el componente a lanzar. A este tipo de intenciones se les conocía como intenciones explícitas.

- **Acción:** Una cadena de caracteres donde indicamos la acción a realizar (o en caso de un Receptor de anuncios (Broadcast receiver) la acción que tuvo lugar y queremos reportar). La clase Intent define una serie de constantes para acciones genéricas que son listadas a continuación. No obstante, además de estas podemos definir nuevas acciones:

Constante	componente a lanzar	Acción
ACTION_CALL	Actividad	Inicializa una llamada de teléfono.
ACTION_EDIT	Actividad	Visualiza datos para que el usuario los edite.
ACTION_MAIN	Actividad	Arranca como actividad principal de una tarea. (sin datos de entrada y sin devolver datos)
ACTION_SYNC	Actividad	Sincroniza datos en un servidor con los datos de un dispositivo móvil.
ACTION_BATTERY_LOW	receptor de anuncios	Advertencia de bateria baja.
ACTION_HEADSET_PLUG	receptor de anuncios	Los auriculares han sido conectados o desconectados.
ACTION_SCREEN_ON	receptor de anuncios	La pantalla es activada.
ACTION_TIMEZONE_CHANGED	receptor de anuncios	Se cambia la selección de zona horaria.

Tabla 1: *Algunas acciones estándar de las Intenciones*

También puedes definir tus propias acciones. En este caso has de indicar el paquete de tu aplicación como prefijo. Por ejemplo:

```
com.zwait.MUESTRA_MENSAJES.
```

Categoría: Complementa a la acción. Indica información adicional sobre el tipo de componente que ha de ser lanzado. El número de categorías puede ser arbitrariamente ampliado. No obstante, en la clase Intent se definen una serie de categorías genéricas que podemos utilizar.

Constante	Significado
CATEGORY_BROWSABLE	La actividad lanzada puede ser con seguridad invocada por el navegador para mostrar los datos referenciados por un enlace - por ejemplo, una imagen o un mensaje de correo electrónico.
CATEGORY_HOME	La actividad muestra la pantalla de inicio, la primera pantalla que ve el usuario cuando el dispositivo está encendido o cuando la tecla HOME es presionada .
CATEGORY_LAUNCHER	La actividad puede ser la actividad inicial de una tarea y se muestra en el lanzador de aplicaciones de nivel superior .
CATEGORY_PREFERENCE	La actividad a lanzar es un panel de preferencias.

Tabla 2: Categorías estándar de las Intenciones

Una categoría suele utilizarse junto con una acción para aportar información adicional. Por ejemplo, indicaremos ACTION_MAIN a las actividades que pueden utilizarse como puntos de entrada de una aplicación. Indicaremos además CATEGORY_LAUNCHER para que la actividad sea mostrada en la pantalla de inicio.

Datos: Referencia a los datos con los que trabajaremos. Hay que expresar estos datos por medio de una URI (el mismo concepto ampliamente utilizado en Internet). Ejemplos de URIs son: tel:968123456, http://www.zwait.com, content://call_log/calls… En muchos casos resulta importante saber el tipo de datos con el que se trabaja. Con este propósito se indica el tipo MIME asociado a la URI, es decir, se utiliza el mismo mecanismo que en Internet. Ejemplos de tipos MIME son text/xml, image/jpeg, audio/mp3…

Extras: Información adicional que será recibida por el componente lanzado. Está formada por un conjunto de pares variable/valor. Estas colecciones de valores se almacenan en un objeto de la clase Bundle (pasar datos entre actividades). Estos valores se introducen en un Intent.

```
intent.putExtra("usuario", "Pepito Perez");
intent.putExtra("edad", 27);
```

Para lanzar una actividad de forma explícita podemos usar el constructor Intent(Context contexto, Class<?> clase). Por ejemplo, para lanzar la actividad explícita AcercaDe escribíamos:

```
Intent intent = new Intent(this, AcercaDe.class);
startActivity(intent);
```

Para lanzar una actividad de forma implícita podemos usar el constructor Intent(String action, Uri uri). Por ejemplo:

```
Intent intent = new Intent(Intent.ACTION_DIAL,
            URI.parse("tel:968123456");
startActivity(intent);
```

También se puede utilizar startActivityForResult() si esperamos que la actividad nos devuelva datos.

8.2 El uso de intenciones implícitas en Zwait

En este apartado explicaré brevemente los diferentes servicios que Zwait ofrece a sus usuarios, mediante el uso de intenciones implícitas.

Figura 43: intenciones.xml

Cómo se observa en el layout anterior correspondiente a la actividad intencionesAct.java, está formado por seis botones, los cinco primeros de ellos asociados mediante el atributo onClick a cada uno de los métodos encargados de implementar diferentes intenciones implícitas.

- **"abrir página Web":** tanto la clínica como el cliente podrán abrir una ventana de navegador por ejemplo con la url de la página web de la clínica.

- **“llamada de Teléfono”:** Se introducirá un número en la aplicación Teléfono, sin llegar a realizar la llamada. No necesita permiso. Esto facilitará la comunicación entre clínica y cliente y viceversa.

Figura 44: dial del teléfono

- **“Google Maps”:** Abre la aplicación Google Maps para una localización determinada. Aquí la clínica puede introducir sus coordenadas de longitud y latitud para que el cliente no tuviera problemas de encontrar la clínica en caso de ser su primera visita.

- **“mandar Correo”:** En este apartado se abrirá la aplicación correo y se rellenarán de manera predefinida el asunto, el texto y el destinatario del mensaje o email. Estas colecciones de valores se almacenan en un objeto de la clase **Bundle** la cual sirve para contener tipos primitivos

y objetos de otras clases. Con esta clase puedes pasar datos entre distintas activities.

- Si ejecutas la aplicación en un terminal real. Observa como el botón "mandar Correo" te permite seleccionar entre diferentes aplicaciones con esta funcionalidad.

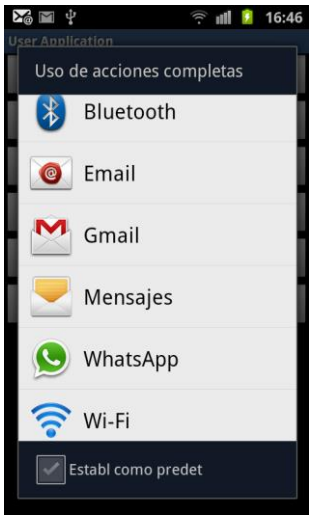

Figura 44: mandar Correo

En Zwait existen tres mensajes predefinidos

1º Clínica → Paciente: "Su cita ha sido modificada. Recuerde consultar su nueva cita en el apartado "Cliente" de Zwait. Rogamos disculpe las molestias. 1 saludo!"

2º Clínica → Paciente: En este mensaje se incluyen los datos correspondientes al código de entidad, código de cita, nombre de la clínica, fecha de la cita y una descripción de la cita.

3º Paciente → Clínica: "Mensaje recibido. Conforme con la cita. 1saludo!"

Figura 46: tipos de mensajes

"Añadir recordatorio": En este apartado se abrirá la aplicación Calendario y se rellenarán de manera predefinida un recordatorio de la cita.

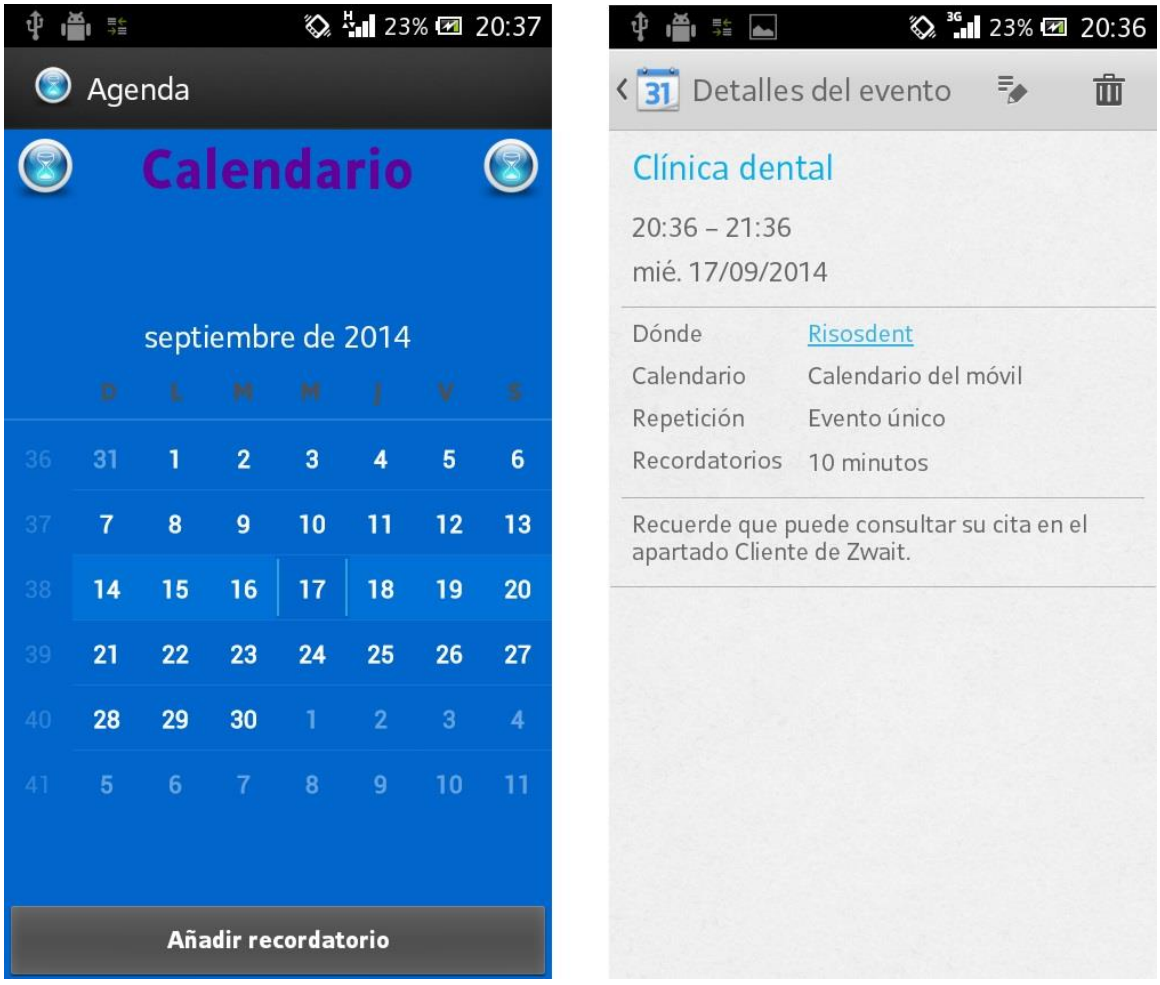

Figura 47: calendario.xml y alerta del calendario

Capítulo 9.

Conclusiones

Considerando que el número de smartphones sigue creciendo, ya existen en el mundo casi tantos teléfonos móviles como personas, la idea de implementar una aplicación android como pfc me pareció bastante atractiva.

Gracias a la gran comunidad de desarrolladores de aplicaciones para Android y la buena documentación que ofrece la comunidad oficial de Android, desarrollar este proyecto con conocimiento prácticamente nulos sobre el tema pudo ser realidad.

El objetivo implícito de este proyecto fue aprender a desarrollar aplicaciones para Android y recordar e incrementar los conocimientos adquiridos durante el transcurso de la carrera en el lenguaje de programación Java, XML, php, MySQL, hmtl y servidores web.

Pienso además, que el tema de las notificaciones predefinidas por medio de sms, correo o wassaps implementado en el proyecto para notificar el retraso de las citas, es muy útil ya que hay muchos establecimientos de diferente índole que gustan de informar a sus clientes por medio de estas herramientas.

También me gustó la idea de hacer una página web para la clínica y aunque el diseño en esta ocasión ha sido sencillo, me ha despertado la curiosidad de profundizar más en la programación web HTML/CSS.

9.1 Dificultades encontradas

Uno de los problemas de los sistemas operativos Android es que no es posible realizar una conexión al servidor en una única tarea principal, por lo que es necesario crear tareas adicionales en el servidor para poder hacer

esta conexión. Por esta razón, tuve que buscar una manera, que desconocía, de solucionarlo.

Aquí muestro alguno de los fallos más frecuentes de su utilización:

1. No se puede llamar a un AsyncTask anidado o dentro de otro AsyncTask.
2. La llamada a un método de ejecución debe de realizarse desde el hilo de la interfaz de usuario.

También surgieron algunos problemas con la configuración del entorno de aplicación, aunque el entorno de desarrollo es bastante intuitivo. Si no se configura debidamente, o no tenemos las APIs adecuadas y la versión de java adecuada, puede dar muchos problemas.

9.2 Mejoras y posibles aplicaciones

- Implementar un servicio de almacenamiento en la nube para guardar en una base de datos las clínicas registradas, sus citas y notificaciones.
- Podría discriminarse entre los usuarios para que en caso de usar la red de Telefónica se ofreciera el producto libre de publicidad y sin coste.
- Inserción de un sistema de búsquedas que mejore la búsqueda al usuario final.
- Aplicación o utilización de este modelo para el envío de notificaciones para otro tipo de establecimientos como por ejemplo Ópticas, las cuales suelen informar a sus clientes cuando tienen listos sus productos tales como gafas, lentillas, etc.
- Mejora del interfaz gráfico.

Bibliografía

Cursos realizados por el autor del trabajo:

Programación en PHP
Impartido por la Plataforma de eFormación de la CARM (form@carm)
De 35 horas de duración.

Desarrollo de aplicaciones para Android
Impartido por la Plataforma de eFormación de la CARM (form@carm)
De 35 horas de duración.

Curso de la Universitat Politècnica de València:
Android: Programación de Aplicaciones
120 horas de duración.

[0] Android Application Development for For Dummies
Wiley Publishing, Inc, 111 River Street Hoboken, NJ 07030-5774

[1] Android: Programación de aplicaciones para móviles
<http://www.androidcurso.com/index.php/tutoriales-android/31-unidad-1-vision-general-y-entorno-de-desarrollo/96-introduccion>

[2] Desarrollo de aplicaciones para Android
<http://www.formacarm.es/moodle/course/view.php?id=69>

[3] Programación en PHP
<http://www.formacarm.es/moodle/course/view.php?id=30>

[4] Página oficial de Android. Se encuentra toda la documentación de Android.
<http://www.android.com/>

[5] Página oficial de MySQL. Documentación de la base de datos MySQL.
<http://www.mysql.com/>

[6] El gran libro de Android.
Jesús Tomas Girones 2ªedicion

[7] MySQL guía de aprendizaje Ullman, Larry

[8] Evolución de Android
<http://androidzone.org/2013/05/historia-de-android-la-evolucion-a-lo-largo-de-sus-versiones/>

[9] Arquitectura de Android
<http://androideity.com/2011/07/04/arquitectura-de-android/>

[10] Stack Overflow: 'android' questions.
<http://stackoverflow.com/questions/tagged/android>

[11] Conferencia: Ganar dinero con Android
<http://www.emezeta.com/articulos/conferencia-ganar-dinero-con-android>

Anexo

App

/Zwait_P16/src/com/zwait/AcercaDe.java

```
package com.zwait;

import android.app.Activity;
import android.os.Bundle;

public class AcercaDe extends Activity {
    /** Called when the activity is first created. */
    @Override public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.acercade);
    }
}
```

/Zwait_P16/src/com/zwait/AgendaAct.java

```
package com.zwait;

import com.zwait.cliente.ClienteActivity;
import com.zwait.clinica.ClinicaActivity;

import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;

public class AgendaAct extends ZwaitAct{


      @Override
      public void onCreate(Bundle savedInstanceState) {
                  super.onCreate(savedInstanceState);
                  setContentView(R.layout.calendario);
      }


      public void calendario(View view){

            Intent intent = new Intent(Intent.ACTION_EDIT);
            intent.setType("vnd.android.cursor.item/event");
            intent.putExtra("title", "Clínica dental");
            intent.putExtra("description", "Recuerde que puede consultar
su cita en el apartado Cliente de Zwait.");
            intent.putExtra("eventLocation", "Risosdent");
            //intent.putExtra("beginTime", eventStartInMillis);
            //intent.putExtra("endTime", eventEndInMillis);
```

```
            startActivity(intent);

      }


      @Override
    public boolean  onCreateOptionsMenu(Menu menu) {
      // si se pulsa el botón MENU

        super.onCreateOptionsMenu(menu);

        // se crea el menú a partir del recurso menu
        getMenuInflater().inflate(R.menu.menu, menu);

        // a cada elemento del menú se asocia un objeto Intent
        menu.findItem(R.id.item1).setIntent(new Intent(this,
ClienteActivity.class));
        menu.findItem(R.id.item2).setIntent(new Intent(this,
ClinicaActivity.class));
        menu.findItem(R.id.item3).setIntent(new Intent(this,
AgendaAct.class));
        menu.findItem(R.id.item4).setIntent(new Intent(this,
AyudaAct.class));

        // el método debe devolver un booleano
        return true;
    }

      @Override
      public boolean onOptionsItemSelected(MenuItem item) {
            // inicia la actividad solicitada

            super.onOptionsItemSelected(item);
            startActivity(item.getIntent());
            return true;
      }

}
```

/Zwait_P16/src/com/zwait/AyudaAct.java

```
package com.zwait;

import java.io.IOException;

import com.zwait.cliente.ClienteActivity;
import com.zwait.clinica.ClinicaActivity;

import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.TextView;


public class AyudaAct extends ZwaitAct{
    /** onCreate se ejecuta cuando se crea la actividad */

      private static final String TAG = "AyudaAct";
```

```
    @Override
      public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // se asigna el layout de la actividad
        setContentView(R.layout.ayuda);

        // se obtiene el objeto TextView de la interfaz de la actividad
      TextView txtAyuda = (TextView) findViewById(R.id.TextView_Ayuda);

      // se asigna el texto al control TextView se usa una función
auxiliar
      try {
            txtAyuda.setText(leerArchivo(TAG, R.raw.ayuda));}
      catch (IOException e) {
             // generar un mensaje de excepción
             Log.e(TAG, "excepción al procesar archivo", e);
         }
    }


    @Override
    public boolean  onCreateOptionsMenu(Menu menu) {
      // si se pulsa el botón MENU

        super.onCreateOptionsMenu(menu);

        // se crea el menú a partir del recurso menu
        getMenuInflater().inflate(R.menu.menu, menu);

        // a cada elemento del menú se asocia un objeto Intent
        menu.findItem(R.id.item1).setIntent(new Intent(this,
ClienteActivity.class));
        menu.findItem(R.id.item2).setIntent(new Intent(this,
ClinicaActivity.class));
        menu.findItem(R.id.item3).setIntent(new Intent(this,
AgendaAct.class));
        menu.findItem(R.id.item4).setIntent(new Intent(this,
AyudaAct.class));

        // el método debe devolver un booleano
        return true;
    }

      @Override
      public boolean onOptionsItemSelected(MenuItem item) {
            // inicia la actividad solicitada

            super.onOptionsItemSelected(item);
            startActivity(item.getIntent());
            return true;
      }


} // fin class AyudaAct
```

/Zwait_P16/src/com/zwait/IntencionesAct.java

```
package com.zwait;

import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;


public class IntencionesAct extends Activity {
       /** Called when the activity is first created. */
    @Override public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.intenciones);


        Button bSalir =(Button) findViewById(R.id.Button07);
        bSalir.setOnClickListener(new OnClickListener() {

          public void onClick(View view) {
           finish();
          }
        });


    }


    public void salir(View view){

       finish();

    }


public void pgWeb(View view) {
       Intent intent = new Intent(Intent.ACTION_VIEW,
                            Uri.parse("http://www.teleco.upct.es/"));
       startActivity(intent);
}

public void llamadaTelefono(View view) {
      // Intent intent = new Intent(Intent.ACTION_CALL,
        //                    Uri.parse("tel:968262435"));

       Intent intent = new Intent(Intent.ACTION_DIAL,
               Uri.parse("tel:5556"));


       startActivity(intent);
}

public void googleMaps(View view) {
       Intent intent = new Intent(Intent.ACTION_VIEW,
                            Uri.parse("geo:41.656313,-0.877351"));
```

```
        startActivity(intent);
}



public void mandarCorreo(View view) {
        Intent intent = new Intent(Intent.ACTION_SEND);
        intent.setType("text/plain");
        intent.putExtra(Intent.EXTRA_SUBJECT, "Notificación Clínica");
        intent.putExtra(Intent.EXTRA_TEXT, "Su cita ha sido modificada.
Recuerde consultar su nueva cita en el apartado \"Cliente\" de Zwait.
Rogamos disculpe las molestias. 1 saludo!");
        intent.putExtra(Intent.EXTRA_EMAIL,
                                    new String[]
{"sanchezdiazfranciscojavier@gmail.com" });
        startActivity(intent);
}



public void calendario(View view){

      Intent intent = new Intent(Intent.ACTION_EDIT);
      intent.setType("vnd.android.cursor.item/event");
      intent.putExtra("title", "Clínica dental");
      intent.putExtra("description", "Recuerde que puede consultar su
cita en el apartado Cliente de Zwait.");
      intent.putExtra("eventLocation", "Risosdent");
      //intent.putExtra("beginTime", eventStartInMillis);
      //intent.putExtra("endTime", eventEndInMillis);
      startActivity(intent);

}

}
```

/Zwait_P16/src/com/zwait/IntencionesClienteAct.java

```
package com.zwait;

import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;


public class IntencionesClienteAct extends Activity {
       /** Called when the activity is first created. */
    @Override public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.intenciones_cliente);


        Button bSalir =(Button) findViewById(R.id.Button07);
        bSalir.setOnClickListener(new OnClickListener() {

          public void onClick(View view) {
```

```
            finish();
          }
        });


    }


    public void salir(View view){

       finish();

    }


public void pgWeb(View view) {
       Intent intent = new Intent(Intent.ACTION_VIEW,
                           Uri.parse("http://www.teleco.upct.es/"));
       startActivity(intent);
}

public void llamadaTelefono(View view) {
      // Intent intent = new Intent(Intent.ACTION_CALL,
        //                     Uri.parse("tel:968262435"));

       Intent intent = new Intent(Intent.ACTION_DIAL,
               Uri.parse("tel:5556"));


       startActivity(intent);
}

public void googleMaps(View view) {
       Intent intent = new Intent(Intent.ACTION_VIEW,
                            Uri.parse("geo:41.656313,-0.877351"));
       startActivity(intent);
}



public void mandarCorreo(View view) {
       Intent intent = new Intent(Intent.ACTION_SEND);
       intent.setType("text/plain");
       intent.putExtra(Intent.EXTRA_SUBJECT, "Notificación Clínica");
       intent.putExtra(Intent.EXTRA_TEXT, "Mensaje recibido. Conforme
con la cita. 1 saludo!");
       intent.putExtra(Intent.EXTRA_EMAIL,
                                    new String[]
{"sanchezdiazfranciscojavier@gmail.com" });
       startActivity(intent);
}



public void calendario(View view){

      Intent intent = new Intent(Intent.ACTION_EDIT);
      intent.setType("vnd.android.cursor.item/event");
      intent.putExtra("title", "Clínica dental");
      intent.putExtra("description", "Recuerde que puede consultar su
cita en el apartado Cliente de Zwait.");
```

```
        intent.putExtra("eventLocation", "Risosdent");
        //intent.putExtra("beginTime", eventStartInMillis);
        //intent.putExtra("endTime", eventEndInMillis);
        startActivity(intent);

}



}
```

/Zwait_P16/src/com/zwait/JSONParser.java

```
package com.zwait;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.UnsupportedEncodingException;
import java.util.List;

import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.NameValuePair;
//import org.apache.http.NameValuePair;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.client.utils.URLEncodedUtils;
import org.apache.http.impl.client.DefaultHttpClient;
import org.json.JSONException;
import org.json.JSONObject;

import android.util.Log;

public class JSONParser {

        static InputStream is = null;
        static JSONObject jObj = null;
        static String json = "";

        // constructor
        public JSONParser() {

        }

        // function get json from url
        // by making HTTP POST or GET mehtod
        public JSONObject makeHttpRequest(String url, String method,
                        List<NameValuePair> params) {

                // Making HTTP request
                try {

                        // check for request method
                        if(method == "POST"){
                                // request method is POST
                                // defaultHttpClient
```

```
                        DefaultHttpClient httpClient = new
DefaultHttpClient();
                        HttpPost httpPost = new HttpPost(url);
                        httpPost.setEntity(new
UrlEncodedFormEntity(params));

                        HttpResponse httpResponse =
httpClient.execute(httpPost);
                        HttpEntity httpEntity =
httpResponse.getEntity();
                        is = httpEntity.getContent();

                  }else if(method == "GET"){
                        // request method is GET
                        DefaultHttpClient httpClient = new
DefaultHttpClient();
                        String paramString =
URLEncodedUtils.format(params, "utf-8");
                        url += "?" + paramString;
                        HttpGet httpGet = new HttpGet(url);

                        HttpResponse httpResponse =
httpClient.execute(httpGet);
                        HttpEntity httpEntity =
httpResponse.getEntity();
                        is = httpEntity.getContent();
                  }


            } catch (UnsupportedEncodingException e) {
                  e.printStackTrace();
            } catch (ClientProtocolException e) {
                  e.printStackTrace();
            } catch (IOException e) {
                  e.printStackTrace();
            }

            try {
                  BufferedReader reader = new BufferedReader(new
InputStreamReader(
                              is, "iso-8859-1"), 8);
                  StringBuilder sb = new StringBuilder();
                  String line = null;
                  while ((line = reader.readLine()) != null) {
                        sb.append(line + "\n");
                  }
                  is.close();
                  json = sb.toString();
            } catch (Exception e) {
                  Log.e("Buffer Error", "Error converting result " +
e.toString());
            }

            // try parse the string to a JSON object
            try {
                  jObj = new JSONObject(json);
            } catch (JSONException e) {
                  Log.e("JSON Parser", "Error parsing data " +
e.toString());
            }
```

```
            // return JSON String
            return jObj;

        }
}
```

/Zwait_P16/src/com/zwait/LoginActivity.java

```
package com.zwait;



import org.json.JSONException;
import org.json.JSONObject;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

import com.zwait.cliente.ClienteActivity;
import com.zwait.clinica.*;
import com.zwait.R;
import com.zwait.library.DatabaseHandler;
import com.zwait.library.UserFunctions;

public class LoginActivity extends Activity {
    Button btnLogin;
    Button btnLinkToRegister;
    EditText inputEmail;
    EditText inputPassword;
    TextView loginErrorMsg;

    // JSON Response node names
    private static String KEY_SUCCESS = "success";
    //private static String KEY_ERROR = "error";
    //private static String KEY_ERROR_MSG = "error_msg";
    private static String KEY_UID = "uid";
    private static String KEY_NAME = "name";
    private static String KEY_EMAIL = "email";
    private static String KEY_CREATED_AT = "created_at";

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.login);

        // Importing all assets like buttons, text fields
        inputEmail = (EditText) findViewById(R.id.loginEmail);
        inputPassword = (EditText) findViewById(R.id.loginPassword);
        btnLogin = (Button) findViewById(R.id.btnLogin);
        btnLinkToRegister = (Button)
findViewById(R.id.btnLinkToRegisterScreen);
        loginErrorMsg = (TextView) findViewById(R.id.login_error);
```

```
        // Login button Click Event
        btnLogin.setOnClickListener(new View.OnClickListener() {

            public void onClick(View view) {
                String email = inputEmail.getText().toString();
                String password = inputPassword.getText().toString();
                UserFunctions userFunction = new UserFunctions();
                Log.d("Button", "Login");
                JSONObject json = userFunction.loginUser(email,
password);

                // check for login response
                try {
                    if (json.getString(KEY_SUCCESS) != null) {
                        loginErrorMsg.setText("");
                        String res = json.getString(KEY_SUCCESS);
                        if(Integer.parseInt(res) == 1){
                            // user successfully logged in
                            // Store user details in SQLite Database
                            DatabaseHandler db = new
DatabaseHandler(getApplicationContext());
                            JSONObject json_user =
json.getJSONObject("user");

                            // Clear all previous data in database

userFunction.logoutUser(getApplicationContext());
                            db.addUser(json_user.getString(KEY_NAME),
json_user.getString(KEY_EMAIL), json.getString(KEY_UID),
json_user.getString(KEY_CREATED_AT));

                            // Launch Dashboard Screen
                            Intent dashboard = new
Intent(getApplicationContext(), ClinicaActivity.class);

                            // Close all views before launching
Dashboard

dashboard.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
                            startActivity(dashboard);

                            // Close Login Screen
                            finish();
                        }else{
                            // Error in login
                            loginErrorMsg.setText("Incorrect
username/password");
                        }
                    }
                } catch (JSONException e) {
                    e.printStackTrace();
                }
            }
        });

        // Link to Register Screen
        btnLinkToRegister.setOnClickListener(new View.OnClickListener()
{

            public void onClick(View view) {
```

```
                Intent i = new Intent(getApplicationContext(),
                        RegisterActivity.class);
                startActivity(i);
                finish();
            }
        });
    }
    @Override
    public boolean  onCreateOptionsMenu(Menu menu) {
        // si se pulsa el botón MENU

        super.onCreateOptionsMenu(menu);

        // se crea el menú a partir del recurso menu
        getMenuInflater().inflate(R.menu.menu, menu);

        // a cada elemento del menú se asocia un objeto Intent
        menu.findItem(R.id.item1).setIntent(new Intent(this,
ClienteActivity.class));
        menu.findItem(R.id.item2).setIntent(new Intent(this,
ClinicaActivity.class));
        menu.findItem(R.id.item3).setIntent(new Intent(this,
AgendaAct.class));
        menu.findItem(R.id.item4).setIntent(new Intent(this,
AyudaAct.class));

        // el método debe devolver un booleano
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // inicia la actividad solicitada

        super.onOptionsItemSelected(item);
        startActivity(item.getIntent());
        return true;
    }


}
```

/Zwait_P16/src/com/zwait/MenuAct.java

```
package com.zwait;

import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.TextView;

import com.zwait.cliente.ClienteActivity;
```

```
import com.zwait.clinica.ClinicaActivity;


public class MenuAct extends ZwaitAct{

       @Override
          public void onCreate(Bundle savedInstanceState) {
              super.onCreate(savedInstanceState);
              setContentView(R.layout.menu);
                  //crea un array de string a partir de un recurso
                        //        String[] elementos ={
                        //
      getResources().getString(R.string.menu_cliente),
                        //
      getResources().getString(R.string.menu_clinica),
                        //
      getResources().getString(R.string.menu_agenda),
                        //
      getResources().getString(R.string.menu_ayuda) };

                    //otro modo de realizar el paso anterior
                      String[] elementos =
getResources().getStringArray(R.array.entradas);

                    //se crea un adaptador para asociar el array de
entrada.
                      ArrayAdapter<String> adaptador = new
ArrayAdapter<String>
                      (this,
android.R.layout.simple_list_item_1,elementos);

                    // se obtiene el objeto ListView
                      ListView listaMenu = (ListView)
findViewById(R.id.ListView_Menu);

                    // se selecciona por defecto el segundo elemento
                      listaMenu.setSelection(1);

                    // se asigna el adaptador al objeto ListView
                      listaMenu.setAdapter(adaptador);

                    // se atienden los eventos click que puedan suceder
dentro del objeto ListView
                      listaMenu.setOnItemClickListener(new
AdapterView.OnItemClickListener() {

                    // si hay un clic en un elemento se analiza  mediante
la comparación del texto de
                    // cada entrada respecto al que recibió el clic
                      public void onItemClick(AdapterView<?> parent,
                                View itemClicked,
                                int position, long id) {

                    // se obtiene el objeto TextView que recibe el clic
                          TextView objClicked = (TextView) itemClicked;

                    // se obtiene la cadena de texto de ese elemento
                          String strTexto =
objClicked.getText().toString();
```

```
                        if
(strTexto.equalsIgnoreCase(getResources().getString(R.string.menu_client
e))) {
                            // Inicia la actividad del cliente
                            startActivity(new Intent(MenuAct.this,
ClienteActivity.class));

                        } else if
(strTexto.equalsIgnoreCase(getResources().getString(R.string.menu_clinic
a))) {
                            // inicia la actividad de la clinica
                            startActivity(new Intent(MenuAct.this,
ClinicaActivity.class));
                          //finaliza la actividad vigente
                             //MenuAct.this.finish();

                        } else if
(strTexto.equalsIgnoreCase(getResources().getString(R.string.menu_agenda
))) {
                            // inicia la actividad de configuración del
usuario
                            startActivity(new Intent(MenuAct.this,
AgendaAct.class));

                        } else if
(strTexto.equalsIgnoreCase(getResources().getString(R.string.menu_ayuda)
)) {
                            // Inicia la actividad de la ayuda
                            startActivity(new Intent(MenuAct.this,
AyudaAct.class));
                        }}
                    });

                    comenzarAnimacion();

          }

      private void comenzarAnimacion() {


            // animación del título
                  // se crea un objeto TextView a partir del objeto
TextView del layout
            TextView txtTitulo = (TextView)
findViewById(R.id.TextViewNombrePantalla);

            // se crea un objeto Animation a partir del objeto recurso
anim03

              Animation objAnim01 = AnimationUtils.loadAnimation(this,
R.anim.anim03);

              // se inicia la animación
              txtTitulo.startAnimation(objAnim01);
      }



       public void lanzarAcercaDe(View view){
            Intent i = new Intent(this, AcercaDe.class);
```

```
                startActivity(i);
        }

        @Override public boolean onCreateOptionsMenu(Menu menu) {
            super.onCreateOptionsMenu(menu);
            MenuInflater inflater = getMenuInflater();
            inflater.inflate(R.menu.zwait, menu);
            return true; /** true -> el menú ya está visible */
         }

         @Override public boolean onOptionsItemSelected(MenuItem item)
{
                switch (item.getItemId()) {
                case R.id.acercaDe:
                       lanzarAcercaDe(null);
                       break;
                }
                return true; /** true -> consumimos el item, no se
propaga*/
         }



    }
```

/Zwait_P16/src/com/zwait/RegisterActivity.java

```
package com.zwait;

import org.json.JSONException;
import org.json.JSONObject;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

import com.zwait.clinica.*;
import com.zwait.R;
import com.zwait.library.DatabaseHandler;
import com.zwait.library.UserFunctions;

public class RegisterActivity extends Activity{
    Button btnRegister;
    Button btnLinkToLogin;
    EditText inputFullName;
    EditText inputEmail;
    EditText inputPassword;
    TextView registerErrorMsg;

    // JSON Response node names
    private static String KEY_SUCCESS = "success";
    //private static String KEY_ERROR = "error";
    //private static String KEY_ERROR_MSG = "error_msg";
    private static String KEY_UID = "uid";
    private static String KEY_NAME = "name";
```

```
      private static String KEY_EMAIL = "email";
      private static String KEY_CREATED_AT = "created_at";

      @Override
      public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.register);

            // Importing all assets like buttons, text fields
            inputFullName = (EditText) findViewById(R.id.registerName);
            inputEmail = (EditText) findViewById(R.id.registerEmail);
            inputPassword = (EditText)
findViewById(R.id.registerPassword);
            btnRegister = (Button) findViewById(R.id.btnRegister);
            btnLinkToLogin = (Button)
findViewById(R.id.btnLinkToLoginScreen);
            registerErrorMsg = (TextView)
findViewById(R.id.register_error);

            // Register Button Click event
            btnRegister.setOnClickListener(new View.OnClickListener() {

                  public void onClick(View view) {
                        String name =
inputFullName.getText().toString();
                        String email = inputEmail.getText().toString();
                        String password =
inputPassword.getText().toString();
                        UserFunctions userFunction = new
UserFunctions();
                        JSONObject json =
userFunction.registerUser(name, email, password);

                        // check for login response
                        try {
                              if (json.getString(KEY_SUCCESS) != null) {
                                    registerErrorMsg.setText("");
                                    String res =
json.getString(KEY_SUCCESS);
                                    if(Integer.parseInt(res) == 1){
                                          // user successfully registred
                                          // Store user details in
SQLite Database
                                          DatabaseHandler db = new
DatabaseHandler(getApplicationContext());
                                          JSONObject json_user =
json.getJSONObject("user");

                                          // Clear all previous data in
database

      userFunction.logoutUser(getApplicationContext());

      db.addUser(json_user.getString(KEY_NAME),
json_user.getString(KEY_EMAIL), json.getString(KEY_UID),
json_user.getString(KEY_CREATED_AT));
                                          // Launch Dashboard Screen
                                          Intent dashboard = new
Intent(getApplicationContext(), ClinicaActivity.class);
                                          // Close all views before
launching Dashboard
```

```
                dashboard.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
                                                startActivity(dashboard);
                                                // Close Registration Screen
                                                finish();
                                          }else{
                                                // Error in registration

                registerErrorMsg.setText("Error occured in registration");
                                          }
                                    }
                              } catch (JSONException e) {
                                    e.printStackTrace();
                              }
                        }
                  });

                  // Link to Login Screen
                  btnLinkToLogin.setOnClickListener(new View.OnClickListener()
{

                        public void onClick(View view) {
                              Intent i = new Intent(getApplicationContext(),
                                          LoginActivity.class);
                              startActivity(i);
                              // Close Registration View
                              finish();
                        }
                  });
            }
}
```

/Zwait_P16/src/com/zwait/Zwait.java

```
package com.zwait;


import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.animation.Animation;
import android.view.animation.Animation.AnimationListener;
import android.view.animation.AnimationUtils;
import android.view.animation.LayoutAnimationController;
import android.widget.TableLayout;
import android.widget.TableRow;
import android.widget.TextView;



public class Zwait extends ZwaitAct{

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_splash);
```

```
      //hacemos la llamada a la función auxiliar para iniciar la
animación
        comenzarAnimacion();
    }

    private void comenzarAnimacion(){
      //creamos un objeto textView a partir del TextView del layout
      TextView txtTitulo = (TextView)
findViewById(R.id.TextViewNombreApp);

      //creamos un objeto Animation a partir del recurso anim01
      Animation objAnim01 =
AnimationUtils.loadAnimation(this,R.anim.anim01);

      // se inicia la animación
      txtTitulo.startAnimation(objAnim01);

      //carga animación para todas las imagenes
      Animation objAnim02 =
AnimationUtils.loadAnimation(this,R.anim.anim02);

      //se crea un objeto layoutAnimationControler
      LayoutAnimationController controller = new
LayoutAnimationController(objAnim02);

      //se crea objeto TabletLayout a partir del layout
      TableLayout tabla =(TableLayout) findViewById(R.id.tableLayout1);

      // se accede a los hijos del objeto
      for (int i = 0;i < tabla.getChildCount(); i++){
            TableRow fila =(TableRow) tabla.getChildAt(i);

            //asigna animación
            fila.setLayoutAnimation(controller);}

      //gestion de la transicion de una actividad a otra
      objAnim02.setAnimationListener(new AnimationListener(){

            public void onAnimationEnd(Animation animation){
                  //se inicia una nueva actividad
                  startActivity(new Intent(Zwait.this,MenuAct.class));
                  //finaliza la actividad vigente
                  Zwait.this.finish();
            }
            public void onAnimationRepeat(Animation animation){
            }
            public void onAnimationStart(Animation animation){
            }
      });
            }
      @Override
      protected void onPause(){
            super.onPause();
            TextView objAnim01 =(TextView)
            findViewById(R.id.TextViewNombreApp);
            objAnim01.clearAnimation();
            TableLayout tabla =(TableLayout)
                        findViewById(R.id.tableLayout1);
            for (int i=0;i< tabla.getChildCount();i++){
                  TableRow fila = (TableRow) tabla.getChildAt(i);
                  fila.clearAnimation();}
```

```
        }
              @Override
       protected void onResume(){
              super.onResume();
              comenzarAnimacion();
       }

    @Override
     public boolean onCreateOptionsMenu(Menu menu) {
         getMenuInflater().inflate(R.menu.activity_splash, menu);
         return true;
     }



}
```

/Zwait_P16/src/com/zwait/ZwaitAct.java

```
package com.zwait;

import java.io.DataInputStream;
import java.io.IOException;
import java.io.InputStream;
import android.app.Activity;

public class ZwaitAct extends Activity {

    // métodos accesible desde las clases derivadas
   public String leerArchivo(String tag, int id)  throws IOException {
         // Lectura del archivo de texto definido como recurso raw el
contenido pasa al control TextView
         InputStream objIs = getResources().openRawResource(id);
         // cadena para guardar cada línea leída
         String sLin = null;
         // se crea el bufer de string
         StringBuffer objSb =  new StringBuffer();
         // objeto que permite la lectura línea a línea
         DataInputStream objDis =new DataInputStream(objIs);
         // Bucle de lectura
         while ((sLin = objDis.readLine()) != null) {
               // se añade la línea leída al bufer
               objSb.append(sLin + "\n"); }
         // cierre de los dos objetos
         objDis.close();
         objIs.close();
         // se devuelve todo el bufer como una cadena
         return objSb.toString();
         } }
```

/Zwait_P16/src/com/zwait/cliente/ClienteActivity.java

```
package com.zwait.cliente;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;

import com.zwait.AgendaAct;
import com.zwait.AyudaAct;
import com.zwait.IntencionesAct;

import com.zwait.R;
import com.zwait.clinica.ClinicaActivity;



public class ClienteActivity extends Activity{

	Button btnViewCitas;
	Button btnNewCita;
	Button btnIntencionesCliente;

	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		//setContentView(R.layout.main_screen);
		setContentView(R.layout.cliente);

		// Buttons
		btnViewCitas = (Button) findViewById(R.id.btnViewCitas);
		btnNewCita = (Button) findViewById(R.id.btnCreateCita);
		btnIntencionesCliente = (Button)
findViewById(R.id.btnIntencionesCliente);

		// view citas click event
		btnViewCitas.setOnClickListener(new View.OnClickListener() {

			@Override
			public void onClick(View view) {
				// Launching All citas Activity

				Intent i = new Intent(getApplicationContext(),
com.zwait.cliente.TablonCitasClienteActivity.class);
				startActivity(i);

			}
		});

		// view citas click event
		btnNewCita.setOnClickListener(new View.OnClickListener() {

			@Override
			public void onClick(View view) {
				// Launching create new cita activity
```

```
                        Intent i = new Intent(getApplicationContext(),
com.zwait.cliente.NuevaCitaClienteActivity.class);
                        startActivity(i);

                  }
            });



            // view intenciones click event
            btnIntencionesCliente.setOnClickListener(new
View.OnClickListener() {

                  @Override
                  public void onClick(View view) {
                        // Launching create new intención activity
                        Intent i = new Intent(getApplicationContext(),
com.zwait.IntencionesClienteAct.class);
                        startActivity(i);

                  }
            });

      }

      @Override
    public boolean  onCreateOptionsMenu(Menu menu) {
      // si se pulsa el botón MENU

        super.onCreateOptionsMenu(menu);

        // se crea el menú a partir del recurso menu
        getMenuInflater().inflate(R.menu.menu, menu);

        // a cada elemento del menú se asocia un objeto Intent
        menu.findItem(R.id.item1).setIntent(new Intent(this,
ClienteActivity.class));
        menu.findItem(R.id.item2).setIntent(new Intent(this,
ClinicaActivity.class));
        menu.findItem(R.id.item3).setIntent(new Intent(this,
AgendaAct.class));
        menu.findItem(R.id.item4).setIntent(new Intent(this,
AyudaAct.class));

        // el método debe devolver un booleano
        return true;
    }
      @Override
      public boolean onOptionsItemSelected(MenuItem item) {
            // inicia la actividad solicitada

            super.onOptionsItemSelected(item);
            startActivity(item.getIntent());
            return true;
      }



}
```

/Zwait_P16/src/com/zwait/cliente/NuevaCitaClienteActivity.java

```
package com.zwait.cliente;

import java.util.ArrayList;
import com.zwait.*;
import java.util.List;

import org.apache.http.NameValuePair;
import org.apache.http.message.BasicNameValuePair;
import org.json.JSONException;
import org.json.JSONObject;

import android.app.Activity;
import android.app.ProgressDialog;
import android.content.Intent;
import android.os.AsyncTask;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;

public class NuevaCitaClienteActivity extends Activity {

	// Progress Dialog
	private ProgressDialog pDialog;

	JSONParser jsonParser = new JSONParser();

	EditText inputId_Entidad;
	EditText inputId_Cita;
	/*EditText inputNombre_Clinica;
	EditText inputFecha_Cita;
	EditText inputDescripcion;

	*/

	// url to create new cita
	private static String url_crear_cita_cliente =
"http://10.0.2.2/android_connect/crear_cita_cliente.php";
	// JSON Node names
	private static final String TAG_SUCCESS = "success";

	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.add_cita_cliente);

		// Edit Text
		inputId_Entidad = (EditText)
findViewById(R.id.inputId_Entidad);
		inputId_Cita = (EditText) findViewById(R.id.inputId_Cita);
		/*inputNombre_Clinica = (EditText)
findViewById(R.id.inputNombre_Clinica);
		inputFecha_Cita = (EditText)
findViewById(R.id.inputFecha_Cita);
		inputDescripcion = (EditText)
findViewById(R.id.inputDescripcion);*/
```

```
            // Create button
            Button btnCreateCita = (Button)
findViewById(R.id.btnCreateCita);

            // button click event
            btnCreateCita.setOnClickListener(new View.OnClickListener()
{

                  @Override
                  public void onClick(View view) {
                        // creating new cita in background thread
                        new CrearNuevaCita().execute();
                  }
            });
      }

      /**
       * Background Async Task to Create new cita
       * */
      class CrearNuevaCita extends AsyncTask<String, String, String> {

            /**
             * Before starting background thread Show Progress Dialog
             * */
            @Override
            protected void onPreExecute() {
                  super.onPreExecute();
                  pDialog = new
ProgressDialog(com.zwait.cliente.NuevaCitaClienteActivity.this);
                  pDialog.setMessage("Creando Cita..");
                  pDialog.setIndeterminate(false);
                  pDialog.setCancelable(true);
                  pDialog.show();
            }

            /**
             * Creating cita
             * */
            protected String doInBackground(String... args) {


                  String id_entidad =
inputId_Entidad.getText().toString();
                  String id_cita = inputId_Cita.getText().toString();
/*                String nombre_clinica =
inputNombre_Clinica.getText().toString();
                  String fecha_cita =
inputFecha_Cita.getText().toString();
                  String descripcion =
inputDescripcion.getText().toString();*/

                  // Building Parameters
                  List<NameValuePair> params = new
ArrayList<NameValuePair>();
                  params.add(new BasicNameValuePair("id_entidad",
id_entidad));
                  params.add(new BasicNameValuePair("id_cita",
id_cita));
            /*    params.add(new BasicNameValuePair("nombre_clinica",
nombre_clinica));
```

```
                  params.add(new BasicNameValuePair("fecha_cita",
fecha_cita));
                  params.add(new BasicNameValuePair("descripcion",
descripcion));*/

                  // getting JSON Object
                  // Note that create cita url accepts POST method
                  JSONObject json =
jsonParser.makeHttpRequest(url_crear_cita_cliente,
                              "POST", params);

                  // check log cat from response
                  Log.d("Create Response", json.toString());

                  // check for success tag
                  try {
                        int success = json.getInt(TAG_SUCCESS);

                        if (success == 1) {
                              // successfully created cita
                              Intent i = new
Intent(getApplicationContext(),
com.zwait.cliente.TablonCitasClienteActivity.class);
                              startActivity(i);

                              // closing this screen
                              finish();
                        } else {
                              // failed to create cita
                        }
                  } catch (JSONException e) {
                        e.printStackTrace();
                  }

                  return null;
            }

            /**
             * After completing background task Dismiss the progress
dialog
             * **/
            protected void onPostExecute(String file_url) {
                  // dismiss the dialog once done
                  pDialog.dismiss();
            }

      }
}
```

/Zwait_P16/src/com/zwait/cliente/TablonCitasClienteActivity.java

```
package com.zwait.cliente;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

import org.apache.http.NameValuePair;
//import org.apache.http.NameValuePair;
import org.json.JSONArray;
```

```
import org.json.JSONException;
import org.json.JSONObject;

import android.app.ListActivity;
import android.app.ProgressDialog;
import android.content.Intent;
import android.os.AsyncTask;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ListAdapter;
import android.widget.ListView;
import android.widget.SimpleAdapter;
import android.widget.TextView;

import com.zwait.JSONParser;
import com.zwait.R;

public class TablonCitasClienteActivity extends ListActivity {

	// Progress Dialog
	private ProgressDialog pDialog;

	// Creating JSON Parser object
	JSONParser jParser = new JSONParser();

	ArrayList<HashMap<String, String>> citasList;

	// url to get all citas list
	private static String url_all_citas_cliente =
"http://10.0.2.2/android_connect/get_all_citas_cliente.php";
	// JSON Node names
	private static final String TAG_SUCCESS = "success";

	private static final String TAG_CITAS = "citas";
	private static final String TAG_PID = "pid";

	private static final String TAG_ID_ENTIDAD = "id_entidad";
	private static final String TAG_ID_CITA = "id_cita";

	// citas JSONArray
	//JSONArray citas = null;
	JSONArray citas = null;

	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.tablon_citas);

		// Hashmap for ListView
		citasList = new ArrayList<HashMap<String, String>>();

		// Loading citas in Background Thread
		new LoadAllCitas().execute();

		// Get listview
		ListView lv = getListView();

		// on seleting single cita
```

```
            // launching Edit Cita Screen
            lv.setOnItemClickListener(new OnItemClickListener() {

                  @Override
                  public void onItemClick(AdapterView<?> parent, View
view,
                              int position, long id) {
                        // getting values from selected ListItem
                        String pid = ((TextView)
view.findViewById(R.id.pid)).getText()
                                    .toString();

                        // Starting new intent
                        Intent in = new Intent(getApplicationContext(),

                  com.zwait.cliente.VerCitaActivity.class);
                        // sending pid to next activity
                        in.putExtra(TAG_PID, pid);

                        // starting new activity and expecting some
response back
                        startActivityForResult(in, 100);
                  }
            });

      }

      // Response from Edit Cita Activity
      @Override
      protected void onActivityResult(int requestCode, int resultCode,
Intent data) {
            super.onActivityResult(requestCode, resultCode, data);
            // if result code 100
            if (resultCode == 100) {
                  // if result code 100 is received
                  // means user edited/deleted cita
                  // reload this screen again
                  Intent intent = getIntent();
                  finish();
                  startActivity(intent);
            }

      }

      /**
       * Background Async Task to Load all cita by making HTTP Request
       * */
      class LoadAllCitas extends AsyncTask<String, String, String> {

            /**
             * Before starting background thread Show Progress Dialog
             * */
            @Override
            protected void onPreExecute() {
                  super.onPreExecute();
                  pDialog = new
ProgressDialog(com.zwait.cliente.TablonCitasClienteActivity.this);
                  pDialog.setMessage("Cargando citas. Por favor
espere...");
                  pDialog.setIndeterminate(false);
                  pDialog.setCancelable(false);
```

```
                pDialog.show();
            }

            /**
             * getting All citas from url
             * */
            protected String doInBackground(String... args) {
                // Building Parameters
                List<NameValuePair> params = new
ArrayList<NameValuePair>();
                // getting JSON string from URL
                JSONObject json =
jParser.makeHttpRequest(url_all_citas_cliente, "GET", params);
                // Check your log cat for JSON reponse
                Log.d("All Citas: ", json.toString());

                try {
                    // Checking for SUCCESS TAG
                    int success = json.getInt(TAG_SUCCESS);

                    if (success == 1) {
                        // citas found
                        // Getting Array of Citas
                        citas = json.getJSONArray(TAG_CITAS);

                        // looping through All Citas
                        for (int i = 0; i < citas.length(); i++) {

                            JSONObject c =
citas.getJSONObject(i);

                            // Storing each json item in
variable
                            String id = c.getString(TAG_PID);
                            //String name =
c.getString(TAG_NAME);
                            String id_entidad =
c.getString(TAG_ID_ENTIDAD);
                            String id_cita =
c.getString(TAG_ID_CITA);


                            // creating new HashMap
                            HashMap<String, String> map = new
HashMap<String, String>();

                            // adding each child node to HashMap
key => value
                            map.put(TAG_PID, id);

                            map.put(TAG_ID_ENTIDAD, id_entidad);
                            map.put(TAG_ID_CITA, id_cita);

                            // adding HashList to ArrayList
                            citasList.add(map);
                        }
                    } else {
                        // no citas found
                        // Launch Add New cita Activity
                        Intent i = new
Intent(getApplicationContext(),
```

```
      com.zwait.cliente.NuevaCitaClienteActivity.class);
                              // Closing all previous activities

      i.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
                              startActivity(i);
                        }
                  } catch (JSONException e) {
                        e.printStackTrace();
                  }

                  return null;
            }

            /**
             * After completing background task Dismiss the progress
dialog
             * **/
            protected void onPostExecute(String file_url) {
                  // dismiss the dialog after getting all citas
                  pDialog.dismiss();
                  // updating UI from Background Thread
                  runOnUiThread(new Runnable() {
                        public void run() {
                              /**
                               * Updating parsed JSON data into ListView
                               * */
                              ListAdapter adapter = new SimpleAdapter(

      com.zwait.cliente.TablonCitasClienteActivity.this, citasList,
                                          R.layout.list_item, new
String[] { TAG_PID,
                                                TAG_ID_ENTIDAD,
TAG_ID_CITA},
                                               new int[] {
R.id.pid, R.id.id_entidad, R.id.id_cita });


                              // updating listview
                              setListAdapter(adapter);
                        }
                  });

            }

      }
}
```

/Zwait_P16/src/com/zwait/cliente/VerCitaActivity.java

```
package com.zwait.cliente;

import java.util.ArrayList;
import java.util.List;

import org.apache.http.NameValuePair;
import org.apache.http.message.BasicNameValuePair;
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;
```

```
import android.app.Activity;
import android.app.ProgressDialog;
import android.content.Intent;
import android.os.AsyncTask;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import com.zwait.*;
public class VerCitaActivity extends Activity {



	TextView txtId_Entidad;
	TextView txtId_Cita;
	TextView txtNombre_Clinica;
	TextView txtFecha_Cita;
	TextView txtDescripcion;


	TextView txtCreatedAt;

	String pid;

	// Progress Dialog
	private ProgressDialog pDialog;

	// JSON parser class
	JSONParser jsonParser = new JSONParser();

	// single cita url
	private static final String url_detalles_cita =
"http://10.0.2.2/android_connect/get_detalles_cita.php";



	// JSON Node names
	private static final String TAG_SUCCESS = "success";
	private static final String TAG_CITA = "cita";
	private static final String TAG_PID = "pid";
	private static final String TAG_ID_ENTIDAD = "id_entidad";
	private static final String TAG_ID_CITA = "id_cita";
	private static final String TAG_NOMBRE_CLINICA = "nombre_clinica";
	private static final String TAG_FECHA_CITA = "fecha_cita";
	private static final String TAG_DESCRIPCION = "descripcion";

	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.ver_detalles_cita);


		// getting cita details from intent
		Intent i = getIntent();

		// getting cita id (pid) from intent
		pid = i.getStringExtra(TAG_PID);

		// Getting complete cita details in background thread
		new GetDetallesCita().execute();
```

```
	}

	/**
	 * Background Async Task to Get complete cita details
	 * */
	class GetDetallesCita extends AsyncTask<String, String, String> {

		/**
		 * Before starting background thread Show Progress Dialog
		 * */
		@Override
		protected void onPreExecute() {
			super.onPreExecute();
			pDialog = new ProgressDialog(VerCitaActivity.this);
			//pDialog.setMessage("Loading cita details. Please
wait...");
			pDialog.setMessage("Cargando detalles de citas. Por
favor espere...");
			pDialog.setIndeterminate(false);
			pDialog.setCancelable(true);
			pDialog.show();
		}

		/**
		 * Getting cita details in background thread
		 * */
		protected String doInBackground(String... params) {

			// updating UI from Background Thread
			runOnUiThread(new Runnable() {
				public void run() {
					// Check for success tag
					int success;
					try {
						// Building Parameters
						List<NameValuePair> params = new
ArrayList<NameValuePair>();
						params.add(new
BasicNameValuePair("pid", pid));

						// getting cita details by making
HTTP request
						// Note that cita details url will
use GET request
						JSONObject json =
jsonParser.makeHttpRequest(
								url_detalles_cita,
"GET", params);

						// check your log for json response

						Log.d("Single Cita Details",
json.toString());

						// json success tag
						success = json.getInt(TAG_SUCCESS);
						if (success == 1) {
							// successfully received cita
details

							//JSONArray citaObj = json
```

```
                                    JSONArray citaObj = json

      .getJSONArray(TAG_CITA); // JSON Array

                                    // get first cita object from
JSON Array
                                    JSONObject cita =
citaObj.getJSONObject(0);

                                    // cita with this pid found
                                    // Edit Text

                                    txtId_Entidad = (TextView)
findViewById(R.id.inputId_Entidad);
                                    txtId_Cita = (TextView)
findViewById(R.id.inputId_Cita);
                                    txtNombre_Clinica = (TextView)
findViewById(R.id.inputNombre_Clinica);
                                    txtFecha_Cita = (TextView)
findViewById(R.id.inputFecha_Cita);
                                    txtDescripcion = (TextView)
findViewById(R.id.inputDescripcion);

                                    // display cita data in
EditText

      txtId_Entidad.setText(cita.getString(TAG_ID_ENTIDAD));

      txtId_Cita.setText(cita.getString(TAG_ID_CITA));

      txtNombre_Clinica.setText(cita.getString(TAG_NOMBRE_CLINICA));

      txtFecha_Cita.setText(cita.getString(TAG_FECHA_CITA));

      txtDescripcion.setText(cita.getString(TAG_DESCRIPCION));

                                    }else{
                                          // cita with pid not found
                                    }
                              } catch (JSONException e) {
                                    e.printStackTrace();
                              }
                        }
                  });

                  return null;
            }


            /**
             * After completing background task Dismiss the progress
dialog
             * **/
            protected void onPostExecute(String file_url) {
                  // dismiss the dialog once got all details
                  pDialog.dismiss();
            }
      }

}
```

/Zwait_P16/src/com/zwait/clinica/ClinicaActivity.java

```
package com.zwait.clinica;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;

import com.zwait.AgendaAct;
import com.zwait.AyudaAct;
import com.zwait.IntencionesAct;
import com.zwait.LoginActivity;

import com.zwait.R;

import com.zwait.cliente.ClienteActivity;
import com.zwait.library.UserFunctions;

public class ClinicaActivity extends Activity {
      UserFunctions userFunctions;

      Button btnViewCitas;
      Button btnNewCita;
      Button btnIntenciones;

      Button btnLogout;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        /**
         * Dashboard Screen for the application
         * */
        // Check login status in database
        userFunctions = new UserFunctions();
        if(userFunctions.isUserLoggedIn(getApplicationContext())){
            //setContentView(R.layout.dashboard);
            setContentView(R.layout.clinica);
            // botones
            btnViewCitas = (Button) findViewById(R.id.btnViewCitas);

            btnNewCita = (Button) findViewById(R.id.btnCreateCita);
            btnIntenciones= (Button) findViewById(R.id.btnIntenciones);
            btnLogout = (Button) findViewById(R.id.btnLogout);

            // view citas click event
            btnViewCitas.setOnClickListener(new View.OnClickListener() {

                  @Override
                  public void onClick(View view) {
                        // Launching All citas Activity
                        Intent i = new Intent(getApplicationContext(),
com.zwait.clinica.TablonCitasActivity.class);
                        startActivity(i);
```

```
                }
            });

            // view citas click event
            btnNewCita.setOnClickListener(new View.OnClickListener() {

                @Override
                public void onClick(View view) {
                    // Launching create new cita activity

                    Intent i = new Intent(getApplicationContext(),
NuevaCitaActivity.class);
                    startActivity(i);

                }
            });



            // view intenciones click event
            btnIntenciones.setOnClickListener(new View.OnClickListener()
{

                @Override
                public void onClick(View view) {
                    // Launching create new cita activity
                    Intent i = new Intent(getApplicationContext(),
IntencionesAct.class);
                    startActivity(i);

                }
            });




            btnLogout.setOnClickListener(new View.OnClickListener() {

                public void onClick(View arg0) {
                    // TODO Auto-generated method stub

      userFunctions.logoutUser(getApplicationContext());
                    Intent login = new
Intent(getApplicationContext(), LoginActivity.class);
                login.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
                startActivity(login);
                // Closing dashboard screen
                finish();
                }
            });

        }else{
            // user is not logged in show login screen
            Intent login = new Intent(getApplicationContext(),
LoginActivity.class);
            login.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
            startActivity(login);
            // Closing dashboard screen
```

```
            finish();
        }

    }

    @Override
    public boolean  onCreateOptionsMenu(Menu menu) {
      // si se pulsa el botón MENU

        super.onCreateOptionsMenu(menu);

        // se crea el menú a partir del recurso menu
        getMenuInflater().inflate(R.menu.menu, menu);

        // a cada elemento del menú se asocia un objeto Intent
        menu.findItem(R.id.item1).setIntent(new Intent(this,
ClienteActivity.class));
        menu.findItem(R.id.item2).setIntent(new Intent(this,
ClinicaActivity.class));
        menu.findItem(R.id.item3).setIntent(new Intent(this,
AgendaAct.class));
        menu.findItem(R.id.item4).setIntent(new Intent(this,
AyudaAct.class));

        // el método debe devolver un booleano
        return true;
    }

      @Override
      public boolean onOptionsItemSelected(MenuItem item) {
            // inicia la actividad solicitada

            super.onOptionsItemSelected(item);
            startActivity(item.getIntent());
            return true;
      }

}
```

/Zwait_P16/src/com/zwait/clinica/EditarCitaActivity.java

```
package com.zwait.clinica;

import java.util.ArrayList;
import java.util.Calendar;
import java.util.List;

import org.apache.http.NameValuePair;
import org.apache.http.message.BasicNameValuePair;
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import android.app.Activity;
import android.app.DatePickerDialog;
import android.app.ProgressDialog;
import android.app.TimePickerDialog;
import android.content.Intent;
import android.net.Uri;
import android.os.AsyncTask;
import android.os.Bundle;
```

```
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.DatePicker;
import android.widget.EditText;
import android.widget.TimePicker;

import com.zwait.JSONParser;
import com.zwait.R;



/**
 *    Application Name : Zwait
 *    Author : Francisco Javier Sánchez Díaz
 *    Website : http://www.zwait.com
 *    Android SDK : 2.2
 *    For more details visit http://www.zwait.com
 */



public class EditarCitaActivity extends Activity implements
           OnClickListener {

      EditText txtId_Entidad;
      EditText txtId_Cita;
      EditText txtNombre_Clinica;
      EditText txtFecha_Cita;
      EditText txtDescripcion;


      EditText txtCreatedAt;
      Button btnSave;
      Button btnDelete;

      String pid;

      // Progress Dialog
      private ProgressDialog pDialog;

      // JSON parser class
      JSONParser jsonParser = new JSONParser();

      // single cita url
      private static final String url_detalles_cita =
"http://10.0.2.2/android_connect/get_detalles_cita.php";

      // url to update cita
      private static final String url_update_cita =
"http://10.0.2.2/android_connect/update_cita.php";

      // url to delete cita
      private static final String url_delete_cita =
"http://10.0.2.2/android_connect/delete_cita.php";


      // JSON Node names
      private static final String TAG_SUCCESS = "success";
      private static final String TAG_CITA = "cita";
```

```
	private static final String TAG_PID = "pid";
	private static final String TAG_ID_ENTIDAD = "id_entidad";
	private static final String TAG_ID_CITA = "id_cita";
	private static final String TAG_NOMBRE_CLINICA = "nombre_clinica";
	private static final String TAG_FECHA_CITA = "fecha_cita";
	private static final String TAG_DESCRIPCION = "descripcion";



	//fin editCitaActivity


	// Widget GUI


	Button btnCalendar, btnTimePicker;
	//Button btnAceptar;

	EditText txtDate, txtTime;
	EditText inputFecha_Cita;

	// Variable for storing current date and time
	private int mYear, mMonth, mDay, mHour, mMinute;
	private int fecha;

	/** Called when the activity is first created. */
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.editar_cita);

		btnCalendar = (Button) findViewById(R.id.btnCalendar);
		btnTimePicker = (Button) findViewById(R.id.btnTimePicker);

		txtDate = (EditText) findViewById(R.id.txtDate);
		txtTime = (EditText) findViewById(R.id.txtTime);
		inputFecha_Cita= (EditText)
findViewById(R.id.inputFecha_Cita);

		btnCalendar.setOnClickListener(this);
		btnTimePicker.setOnClickListener(this);


		// save button
				btnSave = (Button) findViewById(R.id.btnSave);
				btnDelete = (Button)
findViewById(R.id.btnDelete);

				// getting cita details from intent
				Intent i = getIntent();

				// getting cita id (pid) from intent
				pid = i.getStringExtra(TAG_PID);

				// Getting complete cita details in background
thread
				new GetDetallesCita().execute();

				// save button click event
				btnSave.setOnClickListener(new
View.OnClickListener() {
```

```
                                @Override
                                public void onClick(View arg0) {
                                      // starting background task to
update cita
                                      new SalvarDetallesCita().execute();
                                }
                          });

                          // Delete button click event
                          btnDelete.setOnClickListener(new
View.OnClickListener() {

                                @Override
                                public void onClick(View arg0) {
                                      // deleting cita in background
thread
                                      new DeleteCita().execute();
                                }
                          });



      } // fin metodo onCreate

      public void llamadaTelefono(View view) {

            txtId_Entidad = (EditText)
findViewById(R.id.inputId_Entidad);
            String id_entidad = txtId_Entidad.getText().toString();

             Intent intent = new Intent(Intent.ACTION_DIAL,
                    Uri.parse("tel:"+id_entidad));

             startActivity(intent);
      }

      public void mandarSms(View view) {


                  txtId_Entidad = (EditText)
findViewById(R.id.inputId_Entidad);
                  txtId_Cita = (EditText)
findViewById(R.id.inputId_Cita);
                  txtNombre_Clinica = (EditText)
findViewById(R.id.inputNombre_Clinica);
                  txtFecha_Cita = (EditText)
findViewById(R.id.inputFecha_Cita);
                  txtDescripcion = (EditText)
findViewById(R.id.inputDescripcion);

                  String id_entidad =
txtId_Entidad.getText().toString();
                  String id_cita = txtId_Cita.getText().toString();
                  String nombre_clinica =
txtNombre_Clinica.getText().toString();
                  String fecha_cita =
txtFecha_Cita.getText().toString();
                  String descripcion =
txtDescripcion.getText().toString();
```

```
                Intent sendSms = new Intent(Intent.ACTION_SENDTO,
                        Uri.parse("sms:"+id_entidad));


                sendSms.putExtra("sms_body", "Código de entidad:
"+id_entidad+"\nCódigo de cita: "+id_cita+"\nNombre
Clínica:"+nombre_clinica+"\nFecha de la
Cita:"+fecha_cita+"\nDescripción:"+descripcion);


                startActivity(sendSms);
        }

        public void mandarCorreo(View view) {
                txtId_Entidad = (EditText)
findViewById(R.id.inputId_Entidad);
                txtId_Cita = (EditText) findViewById(R.id.inputId_Cita);
                txtNombre_Clinica = (EditText)
findViewById(R.id.inputNombre_Clinica);
                txtFecha_Cita = (EditText)
findViewById(R.id.inputFecha_Cita);
                txtDescripcion = (EditText)
findViewById(R.id.inputDescripcion);

                String id_entidad = txtId_Entidad.getText().toString();
                String id_cita = txtId_Cita.getText().toString();
                String nombre_clinica =
txtNombre_Clinica.getText().toString();
                String fecha_cita = txtFecha_Cita.getText().toString();
                String descripcion = txtDescripcion.getText().toString();

                 Intent intent = new Intent(Intent.ACTION_SEND);
                 intent.setType("text/plain");
                 intent.putExtra(Intent.EXTRA_SUBJECT, "Notificación
Clínica");
                 intent.putExtra(Intent.EXTRA_TEXT,"Código de entidad:
"+id_entidad+"\nCódigo de cita: "+id_cita+"\nNombre
Clínica:"+nombre_clinica+"\nFecha de la
Cita:"+fecha_cita+"\nDescripción:"+descripcion);
                // intent.putExtra(Intent.EXTRA_TEXT, "Su cita ha sido
modificada. Recuerde consultar su nueva cita en el apartado \"Cliente\"
de Zwait. Rogamos disculpe las molestias. 1 saludo!");
                 intent.putExtra(Intent.EXTRA_EMAIL,
                                                 new String[]
{"sanchezdiazfranciscojavier@gmail.com" });
                 startActivity(intent);
        }

        @Override
        public void onClick(View v) {

                if (v == btnCalendar) {

                        // Process to get Current Date
                        final Calendar c = Calendar.getInstance();
                        mYear = c.get(Calendar.YEAR);
                        mMonth = c.get(Calendar.MONTH);
                        mDay = c.get(Calendar.DAY_OF_MONTH);

                        // Launch Date Picker Dialog
```

```
                DatePickerDialog dpd = new DatePickerDialog(this,
                        new DatePickerDialog.OnDateSetListener() {

                            @Override
                            public void onDateSet(DatePicker
view, int year,
                                    int monthOfYear, int
dayOfMonth) {
                                // Display Selected date in
textbox
                                txtDate.setText(year + "-"
                                //inputFecha_Cita.setText(year
+ "-"
                                        + (monthOfYear +
1) + "-" + dayOfMonth);

      inputFecha_Cita.setText(txtDate.getText()+ "  "+
txtTime.getText());

      //inputFecha_Cita.setText(inputFecha_Cita.getText());

      //inputFecha_Cita.setText(txtDate.getText());
                            }
                        }, mYear, mMonth, mDay);
                dpd.show();
            }
            if (v == btnTimePicker) {

                // Process to get Current Time
                final Calendar c = Calendar.getInstance();
                mHour = c.get(Calendar.HOUR_OF_DAY);
                mMinute = c.get(Calendar.MINUTE);

                // Launch Time Picker Dialog
                TimePickerDialog tpd = new TimePickerDialog(this,
                        new TimePickerDialog.OnTimeSetListener() {

                            @Override
                            public void onTimeSet(TimePicker
view, int hourOfDay,
                                    int minute) {
                                // Display Selected time in
textbox

      //txtDate.setText(txtDate.getText()+ "  "+ hourOfDay + ":" +
minute);

                                txtTime.setText(hourOfDay +
":" + minute);

      //inputFecha_Cita.setText(inputFecha_Cita.getText()+" " +
hourOfDay + ":" + minute);

      //inputFecha_Cita.setText(inputFecha_Cita.getText());

      //txtDate.setText(txtDate.getText()+ "  "+ txtTime.getText());

      inputFecha_Cita.setText(txtDate.getText()+ "  "+
txtTime.getText());
```

```
                                    }
                              }, mHour, mMinute, false);
                        tpd.show();

                  }


            }


// EditCita
            /**
             * Background Async Task to Get complete cita details
             * */
            class GetDetallesCita extends AsyncTask<String, String, String> {

                  /**
                   * Before starting background thread Show Progress Dialog
                   * */
                  @Override
                  protected void onPreExecute() {
                        super.onPreExecute();
                        pDialog = new ProgressDialog(EditarCitaActivity.this);

                        pDialog.setMessage("Cargando detalles de citas. Por
favor espere...");
                        pDialog.setIndeterminate(false);
                        pDialog.setCancelable(true);
                        pDialog.show();
                  }

                  /**
                   * Getting cita details in background thread
                   * */
                  protected String doInBackground(String... params) {

                        // updating UI from Background Thread
                        runOnUiThread(new Runnable() {
                              public void run() {
                                    // Check for success tag
                                    int success;
                                    try {
                                          // Building Parameters
                                          List<NameValuePair> params = new
ArrayList<NameValuePair>();
                                          params.add(new
BasicNameValuePair("pid", pid));

                                          // getting cita details by making
HTTP request
                                          // Note that cita details url will
use GET request
                                          JSONObject json =
jsonParser.makeHttpRequest(
                                                      url_detalles_cita,
"GET", params);


                                          // check your log for json response
```

```
                                    Log.d("Single Cita Details",
json.toString());

                                    // json success tag
                                    success = json.getInt(TAG_SUCCESS);
                                    if (success == 1) {
                                          // successfully received cita
details
                                          //JSONArray citaObj = json
                                          JSONArray citaObj = json

      .getJSONArray(TAG_CITA); // JSON Array

                                          // get first cita object from
JSON Array
                                          JSONObject cita =
citaObj.getJSONObject(0);

                                          // cita with this pid found
                                          // Edit Text

                                          txtId_Entidad = (EditText)
findViewById(R.id.inputId_Entidad);
                                          txtId_Cita = (EditText)
findViewById(R.id.inputId_Cita);
                                          txtNombre_Clinica = (EditText)
findViewById(R.id.inputNombre_Clinica);
                                          txtFecha_Cita = (EditText)
findViewById(R.id.inputFecha_Cita);
                                          txtDescripcion = (EditText)
findViewById(R.id.inputDescripcion);

                                          // display cita data in
EditText
      txtId_Entidad.setText(cita.getString(TAG_ID_ENTIDAD));
      txtId_Cita.setText(cita.getString(TAG_ID_CITA));
      txtNombre_Clinica.setText(cita.getString(TAG_NOMBRE_CLINICA));
      txtFecha_Cita.setText(cita.getString(TAG_FECHA_CITA));
      txtDescripcion.setText(cita.getString(TAG_DESCRIPCION));

                                    }else{
                                          // cita with pid not found
                                    }
                              } catch (JSONException e) {
                                    e.printStackTrace();
                              }
                        }
                  });

                  return null;
            }

            /**
```

```
             * After completing background task Dismiss the progress
dialog
             * **/
            protected void onPostExecute(String file_url) {
                  // dismiss the dialog once got all details
                  pDialog.dismiss();
            }
      }

      /**
       * Background Async Task to  Save cita Details
       * */
      class SalvarDetallesCita extends AsyncTask<String, String, String>
{

            /**
             * Before starting background thread Show Progress Dialog
             * */
            @Override
            protected void onPreExecute() {
                  super.onPreExecute();
                  pDialog = new ProgressDialog(EditarCitaActivity.this);
                  pDialog.setMessage("Guardando cita ...");
                  pDialog.setIndeterminate(false);
                  pDialog.setCancelable(true);
                  pDialog.show();
            }

            /**
             * Saving cita
             * */
            protected String doInBackground(String... args) {

                  // getting updated data from EditTexts
                  String id_entidad =
txtId_Entidad.getText().toString();
                  String id_cita = txtId_Cita.getText().toString();
                  String nombre_clinica =
txtNombre_Clinica.getText().toString();
                  String fecha_cita =
txtFecha_Cita.getText().toString();
                  String descripcion =
txtDescripcion.getText().toString();

                  // Building Parameters
                  List<NameValuePair> params = new
ArrayList<NameValuePair>();
                  params.add(new BasicNameValuePair(TAG_PID, pid));

                  params.add(new BasicNameValuePair(TAG_ID_ENTIDAD,
id_entidad));
                  params.add(new BasicNameValuePair(TAG_ID_CITA,
id_cita));
                  params.add(new BasicNameValuePair(TAG_NOMBRE_CLINICA,
nombre_clinica));
                  params.add(new BasicNameValuePair(TAG_FECHA_CITA,
fecha_cita));
                  params.add(new BasicNameValuePair(TAG_DESCRIPCION,
descripcion));

                  // sending modified data through http request
```

```
                // Notice that update cita url accepts POST method
                JSONObject json =
jsonParser.makeHttpRequest(url_update_cita,
                            "POST", params);

                // check json success tag
                try {
                      int success = json.getInt(TAG_SUCCESS);

                      if (success == 1) {
                            // successfully updated
                            Intent i = getIntent();
                            // send result code 100 to notify about
cita update
                            setResult(100, i);
                            finish();
                      } else {
                            // failed to update cita
                      }
                } catch (JSONException e) {
                      e.printStackTrace();
                }

                return null;
          }


          /**
           * After completing background task Dismiss the progress
dialog
           * **/
          protected void onPostExecute(String file_url) {
                // dismiss the dialog once cita updated
                pDialog.dismiss();
          }
     }

     /**************************************************************
      * Background Async Task to Delete Cita
      * */
     class DeleteCita extends AsyncTask<String, String, String> {

          /**
           * Before starting background thread Show Progress Dialog
           * */
          @Override
          protected void onPreExecute() {
                super.onPreExecute();
                pDialog = new ProgressDialog(EditarCitaActivity.this);

                pDialog.setMessage("Borrando Cita...");
                pDialog.setIndeterminate(false);
                pDialog.setCancelable(true);
                pDialog.show();
          }

          /**
           * Deleting cita
           * */
          protected String doInBackground(String... args) {
```

```
                  // Check for success tag
                  int success;
                  try {
                        // Building Parameters
                        List<NameValuePair> params = new
ArrayList<NameValuePair>();
                        params.add(new BasicNameValuePair("pid", pid));

                        // getting cita details by making HTTP request
                        JSONObject json = jsonParser.makeHttpRequest(

                                    url_delete_cita, "POST", params);

                        // check your log for json response
                        Log.d("Delete Cita", json.toString());

                        // json success tag
                        success = json.getInt(TAG_SUCCESS);
                        if (success == 1) {
                              // cita successfully deleted
                              // notify previous activity by sending
code 100
                              Intent i = getIntent();
                              // send result code 100 to notify about
cita deletion
                              setResult(100, i);
                              finish();
                        }
                  } catch (JSONException e) {
                        e.printStackTrace();
                  }

                  return null;
            }

            /**
             * After completing background task Dismiss the progress
dialog
             * **/
            protected void onPostExecute(String file_url) {
                  // dismiss the dialog once cita deleted
                  pDialog.dismiss();

            }

      }


} //fin class
```

/Zwait_P16/src/com/zwait/clinica/NuevaCitaActivity.java

```
package com.zwait.clinica;

import java.util.ArrayList;
import java.util.Calendar;
import java.util.List;

import org.apache.http.NameValuePair;
import org.apache.http.message.BasicNameValuePair;
import org.json.JSONException;
```

```
import org.json.JSONObject;

import android.app.Activity;
import android.app.DatePickerDialog;
import android.app.ProgressDialog;
import android.app.TimePickerDialog;
import android.content.Intent;
import android.os.AsyncTask;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.DatePicker;
import android.widget.EditText;
import android.widget.TimePicker;

import com.zwait.JSONParser;
import com.zwait.R;



public class NuevaCitaActivity extends Activity implements
OnClickListener {
	// Progress Dialog
	private ProgressDialog pDialog;

	JSONParser jsonParser = new JSONParser();

	EditText inputId_Entidad;
	EditText inputId_Cita;
	EditText inputNombre_Clinica;
	EditText inputFecha_Cita;
	EditText inputDescripcion;



	// url to create new cita
	private static String url_crear_cita_clinica =
"http://10.0.2.2/android_connect/crear_cita_clinica.php";


	// JSON Node names
	private static final String TAG_SUCCESS = "success";

	// Widget GUI


		Button btnCalendar, btnTimePicker;


		EditText txtDate, txtTime;
		//EditText inputFecha_Cita;

		// Variable for storing current date and time
		private int mYear, mMonth, mDay, mHour, mMinute;
		//private int fecha;
```

```
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.add_cita);


		btnCalendar = (Button) findViewById(R.id.btnCalendar);
		btnTimePicker = (Button) findViewById(R.id.btnTimePicker);

		txtDate = (EditText) findViewById(R.id.txtDate);
		txtTime = (EditText) findViewById(R.id.txtTime);
		//inputFecha_Cita= (EditText)
findViewById(R.id.inputFecha_Cita);


		// Edit Text
		inputId_Entidad = (EditText)
findViewById(R.id.inputId_Entidad);
		inputId_Cita = (EditText) findViewById(R.id.inputId_Cita);
		inputNombre_Clinica = (EditText)
findViewById(R.id.inputNombre_Clinica);
		inputFecha_Cita = (EditText)
findViewById(R.id.inputFecha_Cita);
		inputDescripcion = (EditText)
findViewById(R.id.inputDescripcion);

		btnCalendar.setOnClickListener(this);
		btnTimePicker.setOnClickListener(this);


		// Create button
		Button btnCreateCita = (Button)
findViewById(R.id.btnCreateCita);

		// button click event
		btnCreateCita.setOnClickListener(new View.OnClickListener()
{

			@Override
			public void onClick(View view) {
				// creating new cita in background thread
				new CrearNuevaCita().execute();
			}
		});
	} // fin onCreate

	@Override
	public void onClick(View v) {

		if (v == btnCalendar) {

			// Process to get Current Date
			final Calendar c = Calendar.getInstance();
			mYear = c.get(Calendar.YEAR);
			mMonth = c.get(Calendar.MONTH);
			mDay = c.get(Calendar.DAY_OF_MONTH);

			// Launch Date Picker Dialog
			DatePickerDialog dpd = new DatePickerDialog(this,
```

```
                                new DatePickerDialog.OnDateSetListener() {

                                      @Override
                                      public void onDateSet(DatePicker
view, int year,
                                                  int monthOfYear, int
dayOfMonth) {
                                            // Display Selected date in
textbox
                                            txtDate.setText(year + "-"

                                                  + (monthOfYear +
1) + "-" + dayOfMonth);

      inputFecha_Cita.setText(txtDate.getText()+ "  "+
txtTime.getText());

                                      }
                                }, mYear, mMonth, mDay);
                        dpd.show();
                  }
                  if (v == btnTimePicker) {

                        // Process to get Current Time
                        final Calendar c = Calendar.getInstance();
                        mHour = c.get(Calendar.HOUR_OF_DAY);
                        mMinute = c.get(Calendar.MINUTE);

                        // Launch Time Picker Dialog
                        TimePickerDialog tpd = new TimePickerDialog(this,
                                    new TimePickerDialog.OnTimeSetListener() {

                                      @Override
                                      public void onTimeSet(TimePicker
view, int hourOfDay,
                                                  int minute) {
                                            // Display Selected time in
textbox

      //txtDate.setText(txtDate.getText()+ "  "+ hourOfDay + ":" +
minute);

                                            txtTime.setText(hourOfDay +
":" + minute);

      //inputFecha_Cita.setText(inputFecha_Cita.getText()+" " +
hourOfDay + ":" + minute);

      //inputFecha_Cita.setText(inputFecha_Cita.getText());

      //txtDate.setText(txtDate.getText()+ "  "+ txtTime.getText());

      inputFecha_Cita.setText(txtDate.getText()+ "  "+
txtTime.getText());



                                      }
                                }, mHour, mMinute, false);
                        tpd.show();
```

```
		}
	}



	/**
	 * Background Async Task to Create new cita
	 * */
	class CrearNuevaCita extends AsyncTask<String, String, String> {

		/**
		 * Before starting background thread Show Progress Dialog
		 * */
		@Override
		protected void onPreExecute() {
			super.onPreExecute();
			pDialog = new ProgressDialog(NuevaCitaActivity.this);
			pDialog.setMessage("Creando Cita..");
			pDialog.setIndeterminate(false);
			pDialog.setCancelable(true);
			pDialog.show();
		}

		/**
		 * Creating cita
		 * */
		protected String doInBackground(String... args) {


			String id_entidad =
inputId_Entidad.getText().toString();
			String id_cita = inputId_Cita.getText().toString();
			String nombre_clinica =
inputNombre_Clinica.getText().toString();
			String fecha_cita =
inputFecha_Cita.getText().toString();
			String descripcion =
inputDescripcion.getText().toString();

			// Building Parameters
			List<NameValuePair> params = new
ArrayList<NameValuePair>();
			params.add(new BasicNameValuePair("id_entidad",
id_entidad));
			params.add(new BasicNameValuePair("id_cita",
id_cita));
			params.add(new BasicNameValuePair("nombre_clinica",
nombre_clinica));
			params.add(new BasicNameValuePair("fecha_cita",
fecha_cita));
			params.add(new BasicNameValuePair("descripcion",
descripcion));

			// getting JSON Object
			// Note that create cita url accepts POST method

			JSONObject json =
jsonParser.makeHttpRequest(url_crear_cita_clinica,
                                "POST", params);
```

```
                        // check log cat from response
                        Log.d("Create Response", json.toString());

                        // check for success tag
                        try {
                              int success = json.getInt(TAG_SUCCESS);

                              if (success == 1) {
                                    // successfully created cita

                                    Intent i = new
Intent(getApplicationContext(),
com.zwait.clinica.TablonCitasActivity.class);
                                    startActivity(i);

                                    // closing this screen
                                    finish();
                              } else {
                                    // failed to create cita
                              }
                        } catch (JSONException e) {
                              e.printStackTrace();
                        }

                        return null;
                  }

                  /**
                   * After completing background task Dismiss the progress
dialog
                   * **/
                  protected void onPostExecute(String file_url) {
                        // dismiss the dialog once done
                        pDialog.dismiss();
                  }

            }
}
```

/Zwait_P16/src/com/zwait/clinica/TablonCitasActivity.java

```
package com.zwait.clinica;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;

import org.apache.http.NameValuePair;
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import android.app.ListActivity;
import android.app.ProgressDialog;
import android.content.Intent;
import android.os.AsyncTask;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ListAdapter;
import android.widget.ListView;
import android.widget.SimpleAdapter;
import android.widget.TextView;


import com.zwait.JSONParser;

import com.zwait.R;

public class TablonCitasActivity extends ListActivity {

	// Progress Dialog
	private ProgressDialog pDialog;

	// Creating JSON Parser object
	JSONParser jParser = new JSONParser();

	//ArrayList<HashMap<String, String>> citasList;
	ArrayList<HashMap<String, String>> citasList;

	// url para obtener la lista de todas las citas
	private static String url_all_citas =
"http://10.0.2.2/android_connect/get_all_citas_clinica.php";

	// JSON Node names
	private static final String TAG_SUCCESS = "success";
	private static final String TAG_CITAS = "citas";
	private static final String TAG_PID = "pid";
	private static final String TAG_ID_ENTIDAD = "id_entidad";
	private static final String TAG_ID_CITA = "id_cita";

	// citas JSONArray
	JSONArray citas = null;

	@Override
	public void onCreate(Bundle savedInstanceState) {
```

```
            super.onCreate(savedInstanceState);
            setContentView(R.layout.tablon_citas);

            // Hashmap for ListView
            citasList = new ArrayList<HashMap<String, String>>();


            // Loading citas in Background Thread
            new LoadAllCitas().execute();


            // Get listview
            ListView lv = getListView();

            // on seleting single cita
            // launching Edit Cita Screen
            lv.setOnItemClickListener(new OnItemClickListener() {

                  @Override
                  public void onItemClick(AdapterView<?> parent, View
view,
                              int position, long id) {
                        // getting values from selected ListItem
                        String pid = ((TextView)
view.findViewById(R.id.pid)).getText()
                                    .toString();

                        // Starting new intent
                        Intent in = new Intent(getApplicationContext(),


      com.zwait.clinica.EditarCitaActivity.class);

                        // sending pid to next activity
                        in.putExtra(TAG_PID, pid);

                        // starting new activity and expecting some
response back
                        startActivityForResult(in, 100);
                  }
            });

      }

      // Response from Edit Cita Activity
      @Override
      protected void onActivityResult(int requestCode, int resultCode,
Intent data) {
            super.onActivityResult(requestCode, resultCode, data);
            // if result code 100
            if (resultCode == 100) {
                  // if result code 100 is received
                  // means user edited/deleted cita
                  // reload this screen again
                  Intent intent = getIntent();
                  finish();
                  startActivity(intent);
            }

      }
```

```
	/**
	 * Background Async Task to Load all cita by making HTTP Request
	 * */
	class LoadAllCitas extends AsyncTask<String, String, String> {
		/**
		 * Before starting background thread Show Progress Dialog
		 * */
		@Override
		protected void onPreExecute() {
			super.onPreExecute();
			pDialog = new
ProgressDialog(TablonCitasActivity.this);
			pDialog.setMessage("Cargando citas. Por favor
espere...");
			pDialog.setIndeterminate(false);
			pDialog.setCancelable(false);
			pDialog.show();
		}

		/**
		 * getting All citas from url
		 * */
		protected String doInBackground(String... args) {
			// Building Parameters
			List<NameValuePair> params = new
ArrayList<NameValuePair>();
			// getting JSON string from URL
			JSONObject json =
jParser.makeHttpRequest(url_all_citas, "GET", params);

			// Check your log cat for JSON reponse
			Log.d("All Citas: ", json.toString());

			try {
				// Checking for SUCCESS TAG
				int success = json.getInt(TAG_SUCCESS);

				if (success == 1) {
					// citas found
					// Getting Array of Citas

					citas = json.getJSONArray(TAG_CITAS);


					// looping through All Citas

					for (int i = 0; i < citas.length(); i++) {

						JSONObject c =
citas.getJSONObject(i);

						// Storing each json item in
variable
						String id = c.getString(TAG_PID);
						//String name =
c.getString(TAG_NAME);
						String id_entidad =
c.getString(TAG_ID_ENTIDAD);
						String id_cita =
c.getString(TAG_ID_CITA);
```

```
                                    // creating new HashMap
                                    HashMap<String, String> map = new
HashMap<String, String>();

                                    // adding each child node to HashMap
key => value
                                    map.put(TAG_PID, id);
                                    //map.put(TAG_NAME, name);
                                    map.put(TAG_ID_ENTIDAD, id_entidad);
                                    map.put(TAG_ID_CITA, id_cita);

                                    // adding HashList to ArrayList

                                    citasList.add(map);
                              }
                        } else {
                              // no citas found
                              // Launch Add New cita Activity
                              Intent i = new
Intent(getApplicationContext(),
                                          NuevaCitaActivity.class);


                              // Closing all previous activities

      i.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
                              startActivity(i);
                        }
                  } catch (JSONException e) {
                        e.printStackTrace();
                  }

                  return null;
            }

            /**
             * After completing background task Dismiss the progress
dialog
             * **/
            protected void onPostExecute(String file_url) {
                  // dismiss the dialog after getting all citas
                  pDialog.dismiss();
                  // updating UI from Background Thread
                  runOnUiThread(new Runnable() {
                        public void run() {
                              /**
                               * Updating parsed JSON data into ListView
                               * */
                              ListAdapter adapter = new SimpleAdapter(

                                          TablonCitasActivity.this,
citasList,
                                          R.layout.list_item, new
String[] { TAG_PID,
                                                TAG_ID_ENTIDAD,
TAG_ID_CITA},
                                          new int[] {
R.id.pid, R.id.id_entidad, R.id.id_cita });
```

```
                                    // updating listview
                                    setListAdapter(adapter);
                              }
                        });

                  }

            }
}
```

/Zwait_P16/src/com/zwait/library/DatabaseHandler.java

```
package com.zwait.library;

import java.util.HashMap;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

public class DatabaseHandler extends SQLiteOpenHelper {

      // All Static variables
      // Database Version
      private static final int DATABASE_VERSION = 1;

      // Database Name
      private static final String DATABASE_NAME = "android_api";

      // Login table name
      private static final String TABLE_LOGIN = "login";

      // Login Table Columns names
      private static final String KEY_ID = "id";
      private static final String KEY_NAME = "name";
      private static final String KEY_EMAIL = "email";
      private static final String KEY_UID = "uid";
      private static final String KEY_CREATED_AT = "created_at";

      public DatabaseHandler(Context context) {
            super(context, DATABASE_NAME, null, DATABASE_VERSION);
      }

      // Creating Tables
      @Override
      public void onCreate(SQLiteDatabase db) {
            String CREATE_LOGIN_TABLE = "CREATE TABLE " + TABLE_LOGIN +
"("
                        + KEY_ID + " INTEGER PRIMARY KEY,"
                        + KEY_NAME + " TEXT,"
                        + KEY_EMAIL + " TEXT UNIQUE,"
                        + KEY_UID + " TEXT,"
                        + KEY_CREATED_AT + " TEXT" + ")";
            db.execSQL(CREATE_LOGIN_TABLE);
      }

      // Upgrading database
      @Override
```

```
	public void onUpgrade(SQLiteDatabase db, int oldVersion, int
newVersion) {
		// Drop older table if existed
		db.execSQL("DROP TABLE IF EXISTS " + TABLE_LOGIN);

		// Create tables again
		onCreate(db);
	}

	/**
	 * Storing user details in database
	 * */
	public void addUser(String name, String email, String uid, String
created_at) {
		SQLiteDatabase db = this.getWritableDatabase();

		ContentValues values = new ContentValues();
		values.put(KEY_NAME, name); // Name
		values.put(KEY_EMAIL, email); // Email
		values.put(KEY_UID, uid); // Email
		values.put(KEY_CREATED_AT, created_at); // Created At

		// Inserting Row
		db.insert(TABLE_LOGIN, null, values);
		db.close(); // Closing database connection
	}

	/**
	 * Getting user data from database
	 * */
	public HashMap<String, String> getUserDetails(){
		HashMap<String,String> user = new HashMap<String,String>();
		String selectQuery = "SELECT  * FROM " + TABLE_LOGIN;

	  SQLiteDatabase db = this.getReadableDatabase();
	  Cursor cursor = db.rawQuery(selectQuery, null);
	  // Move to first row
	  cursor.moveToFirst();
	  if(cursor.getCount() > 0){
		user.put("name", cursor.getString(1));
		user.put("email", cursor.getString(2));
		user.put("uid", cursor.getString(3));
		user.put("created_at", cursor.getString(4));
	  }
	  cursor.close();
	  db.close();
		// return user
		return user;
	}

	/**
	 * Getting user login status
	 * return true if rows are there in table
	 * */
	public int getRowCount() {
		String countQuery = "SELECT  * FROM " + TABLE_LOGIN;
		SQLiteDatabase db = this.getReadableDatabase();
		Cursor cursor = db.rawQuery(countQuery, null);
		int rowCount = cursor.getCount();
		db.close();
		cursor.close();
```

```
            // return row count
            return rowCount;
      }

      /**
       * Re crate database
       * Delete all tables and create them again
       * */
      public void resetTables(){
            SQLiteDatabase db = this.getWritableDatabase();
            // Delete All Rows
            db.delete(TABLE_LOGIN, null, null);
            db.close();
      }

}
```

/Zwait_P16/src/com/zwait/library/JSONParser.java

```
package com.zwait.library;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.UnsupportedEncodingException;
import java.util.List;

import org.apache.http.HttpEntity;
import org.apache.http.HttpResponse;
import org.apache.http.NameValuePair;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.entity.UrlEncodedFormEntity;
import org.apache.http.client.methods.HttpPost;
import org.apache.http.impl.client.DefaultHttpClient;
import org.json.JSONException;
import org.json.JSONObject;

import android.util.Log;

public class JSONParser {

      static InputStream is = null;
      static JSONObject jObj = null;
      static String json = "";

      // constructor
      public JSONParser() {

      }

      public JSONObject getJSONFromUrl(String url, List<NameValuePair>
params) {

            // Making HTTP request
            try {
                  // defaultHttpClient
                  DefaultHttpClient httpClient = new
DefaultHttpClient();
```

```
                HttpPost httpPost = new HttpPost(url);
                httpPost.setEntity(new UrlEncodedFormEntity(params));

                HttpResponse httpResponse =
httpClient.execute(httpPost);
                HttpEntity httpEntity = httpResponse.getEntity();
                is = httpEntity.getContent();

            } catch (UnsupportedEncodingException e) {
                e.printStackTrace();
            } catch (ClientProtocolException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            }

            try {
                BufferedReader reader = new BufferedReader(new
InputStreamReader(
                        is, "iso-8859-1"), 8);
                StringBuilder sb = new StringBuilder();
                String line = null;
                while ((line = reader.readLine()) != null) {
                    sb.append(line + "\n");
                }
                is.close();
                json = sb.toString();
                Log.e("JSON", json);
            } catch (Exception e) {
                Log.e("Buffer Error", "Error converting result " +
e.toString());
            }

            // try parse the string to a JSON object
            try {
                jObj = new JSONObject(json);
            } catch (JSONException e) {
                Log.e("JSON Parser", "Error parsing data " +
e.toString());
            }

            // return JSON String
            return jObj;

    }
}
```

/Zwait_P16/src/com/zwait/library/UserFunctions.java

```
package com.zwait.library;

import java.util.ArrayList;
import java.util.List;

import org.apache.http.NameValuePair;
import org.apache.http.message.BasicNameValuePair;
import org.json.JSONObject;

import android.content.Context;
```

```
public class UserFunctions {

	private JSONParser jsonParser;

	private static String loginURL = "http://10.0.2.2/ah_login_api/";
	private static String registerURL =
"http://10.0.2.2/ah_login_api/";

	private static String login_tag = "login";
	private static String register_tag = "register";

	// constructor
	public UserFunctions(){
		jsonParser = new JSONParser();
	}

	/**
	 * function make Login Request
	 * @param email
	 * @param password
	 * */
	public JSONObject loginUser(String email, String password){
		// Building Parameters
		List<NameValuePair> params = new ArrayList<NameValuePair>();
		params.add(new BasicNameValuePair("tag", login_tag));
		params.add(new BasicNameValuePair("email", email));
		params.add(new BasicNameValuePair("password", password));
		JSONObject json = jsonParser.getJSONFromUrl(loginURL,
params);
		// return json
		// Log.e("JSON", json.toString());
		return json;
	}

	/**
	 * function make Login Request
	 * @param name
	 * @param email
	 * @param password
	 * */
	public JSONObject registerUser(String name, String email, String
password){
		// Building Parameters
		List<NameValuePair> params = new ArrayList<NameValuePair>();
		params.add(new BasicNameValuePair("tag", register_tag));
		params.add(new BasicNameValuePair("name", name));
		params.add(new BasicNameValuePair("email", email));
		params.add(new BasicNameValuePair("password", password));

		// getting JSON Object
		JSONObject json = jsonParser.getJSONFromUrl(registerURL,
params);
		// return json
		return json;
	}

	/**
	 * Function get Login status
	 * */
	public boolean isUserLoggedIn(Context context){
```

```
            DatabaseHandler db = new DatabaseHandler(context);
            int count = db.getRowCount();
            if(count > 0){
                  // user logged in
                  return true;
            }
            return false;
      }

      /**
       * Function to logout user
       * Reset Database
       * */
      public boolean logoutUser(Context context){
            DatabaseHandler db = new DatabaseHandler(context);
            db.resetTables();
            return true;
      }

}
```

PHP

C:\Program Files\EasyPHP-DevServer-13.1VC9\data\localweb\ah_login_api\config.php

```
<?php

define("DB_HOST", "localhost");
define("DB_USER", "root");
define("DB_PASSWORD", "");
define("DB_DATABASE", "android_api");
?>
```

C:\Program Files\EasyPHP-DevServer-13.1VC9\data\localweb\ah_login_api\DB_Connect.php

```
<?php

class DB_Connect {

    function __construct() {

    }

    function __destruct() {
        $this->close();
    }

    public function connect() {
        require_once 'config.php';
        $con = mysql_connect(DB_HOST, DB_USER, DB_PASSWORD);
        mysql_select_db(DB_DATABASE);

        return $con;
    }

    public function close() {
        mysql_close();
    }

}

?>
```

C:\Program Files\EasyPHP-DevServer-13.1VC9\data\localweb\ah_login_api\DB_Functions.php

```
<?php

class DB_Functions {

    private $db;


    function __construct() {
        require_once 'DB_Connect.php';
```

```
        $this->db = new DB_Connect();
        $this->db->connect();
    }

    function __destruct() {

    }

    /**
     * Guardamos nuevos usuarios
     * Devolvemos detalle del usuario
     */
    public function storeUser($name, $email, $password) {
        $uuid = uniqid('', true);
        $hash = $this->hashSSHA($password);
        $encrypted_password = $hash["encrypted"]; // password encriptada
        $salt = $hash["salt"]; // salt
        $result = mysql_query("INSERT INTO users(unique_id, name, email,
encrypted_password, salt, created_at) VALUES('$uuid', '$name', '$email',
'$encrypted_password', '$salt', NOW())");
        // comprobamos que la query se ejecutó correctamente
        if ($result) {
            // obtener detalle del usuario
            $uid = mysql_insert_id();
            $result = mysql_query("SELECT * FROM users WHERE uid =
$uid");
            // devolvemos detalle del usuario
            return mysql_fetch_array($result);
        } else {
            return false;
        }
    }

    /**
     * Obtenemos usuario por email y contraseña
     */
    public function getUserByEmailAndPassword($email, $password) {
        $result = mysql_query("SELECT * FROM users WHERE email =
'$email'") or die(mysql_error());
        // verificamos el resultado de la query
        $no_of_rows = mysql_num_rows($result);
        if ($no_of_rows > 0) {
            $result = mysql_fetch_array($result);
            $salt = $result['salt'];
            $encrypted_password = $result['encrypted_password'];
            $hash = $this->checkhashSSHA($salt, $password);

            if ($encrypted_password == $hash) {
                // Se ha autenticado al usuario correctamente
                return $result;
            }
        } else {
            // usuario no encontrado
            return false;
        }
    }

    /**
     * Comprobamos si el usuario existe o no
     */
    public function isUserExisted($email) {
```

```
        $result = mysql_query("SELECT email from users WHERE email =
'$email'");
        $no_of_rows = mysql_num_rows($result);
        if ($no_of_rows > 0) {
            // usuario existe
            return true;
        } else {
            // usuario no existe
            return false;
        }
    }

    /**
     * Encriptamos la password
     * devolvemos el salt y la password encriptada
     */
    public function hashSSHA($password) {

        $salt = sha1(rand());
        $salt = substr($salt, 0, 10);
        $encrypted = base64_encode(sha1($password . $salt, true) .
$salt);
        $hash = array("salt" => $salt, "encrypted" => $encrypted);
        return $hash;
    }

    /**
     * Desencriptamos la password
     * Devolvemos una hash string
     */
    public function checkhashSSHA($salt, $password) {

        $hash = base64_encode(sha1($password . $salt, true) . $salt);

        return $hash;
    }

}

?>
```

C:\Program Files\EasyPHP-DevServer-13.1VC9\data\localweb\ah_login_api\Index.php

```
<?php
  /**
 * Comprobamos la petición POST
 */
if (isset($_POST['tag']) && $_POST['tag'] != '') {
    // obtenemos tag
    $tag = $_POST['tag'];

    require_once 'DB_Functions.php';
    $db = new DB_Functions();

    // Empezamos a preparar la respuesta en forma de Array que luego
convertiremos en un json
    $response = array("tag" => $tag, "success" => 0, "error" => 0);

    // evaluamos tag
    if ($tag == 'login') {
```

```
        // chequeamos el login
        $email = $_POST['email'];
        $password = $_POST['password'];

        // obtenemos y comprobamos el usuario por email y password
        $user = $db->getUserByEmailAndPassword($email, $password);
        if ($user != false) {
            // usuario encontrado
            // marcamos el json como correcto con success = 1
            $response["success"] = 1;
            $response["uid"] = $user["unique_id"];
            $response["user"]["name"] = $user["name"];
            $response["user"]["email"] = $user["email"];
            $response["user"]["created_at"] = $user["created_at"];
            $response["user"]["updated_at"] = $user["updated_at"];
            echo json_encode($response);
        } else {
            // usuario no encontrado
            // marcamos el json con error = 1
            $response["error"] = 1;
            $response["error_msg"] = "Email o password incorrecto!";
            echo json_encode($response);
        }
    } else if ($tag == 'register') {
        // Registrar nuevo usuario
        $name = $_POST['name'];
        $email = $_POST['email'];
        $password = $_POST['password'];

        // comprobamos si ya existe el usuario
        if ($db->isUserExisted($email)) {
            // usuario ya existe - marcamos error 2
            $response["error"] = 2;
            $response["error_msg"] = "Usuario ya existe";
            echo json_encode($response);
        } else {
            // guardamos el nuevo usuario
            $user = $db->storeUser($name, $email, $password);
            if ($user) {
                // usuario guardado correctamente
                $response["success"] = 1;
                $response["uid"] = $user["unique_id"];
                $response["user"]["name"] = $user["name"];
                $response["user"]["email"] = $user["email"];
                $response["user"]["created_at"] = $user["created_at"];
                $response["user"]["updated_at"] = $user["updated_at"];
                echo json_encode($response);
            } else {
                // fallo en la inserción del usuario
                $response["error"] = 1;
                $response["error_msg"] = "Ocurrió un error durante el
registro";
                echo json_encode($response);
            }
        }
    } else {
        echo "Petición no válida";
    }
} else {
    echo "Acceso denegado";
}
```

```
?>
```

C:\Program Files\EasyPHP-DevServer-13.1VC9\data\localweb\android_connect\crear_cita_cliente.php

```
<?php

/*
 * Following code will create a new cita row
 * All cita details are read from HTTP Post Request
 */

// array for JSON response
$response = array();

// check for required fields
//if (isset($_POST['id_entidad']) && isset($_POST['id_cita']) &&
isset($_POST['nombre_clinica']) && isset($_POST['fecha_cita']) &&
isset($_POST['descripcion'])) {
  if (isset($_POST['id_entidad']) && isset($_POST['id_cita'])) {
      $id_entidad = $_POST['id_entidad'];
      $id_cita = $_POST['id_cita'];
      /*$nombre_clinica = $_POST['nombre_clinica'];
      $fecha_cita = $_POST['fecha_cita'];
    $descripcion = $_POST['descripcion'];*/

    // include db connect class
    require_once __DIR__ . '/db_connect.php';

    // connecting to db
    $db = new DB_CONNECT();

    // mysql inserting a new row
  //  $result = mysql_query("INSERT INTO gestorcitas(id_entidad,
id_cita, nombre_clinica, fecha_cita, descripcion) VALUES('$id_entidad',
'$id_cita', '$nombre_clinica','$fecha_cita', '$descripcion')");
    $result = mysql_query("INSERT INTO gestorcitas(id_entidad, id_cita)
VALUES('$id_entidad', '$id_cita')");
      // check if row inserted or not
    if ($result) {
        // successfully inserted into database
        $response["success"] = 1;
        $response["message"] = "Cita successfully created.";

        // echoing JSON response
        echo json_encode($response);
    } else {
        // failed to insert row
        $response["success"] = 0;
        $response["message"] = "Oops! An error occurred.";

        // echoing JSON response
        echo json_encode($response);
    }
} else {
    // required field is missing
    $response["success"] = 0;
    $response["message"] = "Required field(s) is missing";

    // echoing JSON response
    echo json_encode($response);
```

```
}
?>
```

C:\Program Files\EasyPHP-DevServer-13.1VC9\data\localweb\android_connect\crear_cita_clinica.php

```
<?php

/*
 * Following code will create a new cita row
 * All cita details are read from HTTP Post Request
 */

// array for JSON response
$response = array();

// check for required fields
if (isset($_POST['id_entidad']) && isset($_POST['id_cita']) &&
isset($_POST['nombre_clinica']) && isset($_POST['fecha_cita']) &&
isset($_POST['descripcion'])) {

      $id_entidad = $_POST['id_entidad'];
      $id_cita = $_POST['id_cita'];
      $nombre_clinica = $_POST['nombre_clinica'];
      $fecha_cita = $_POST['fecha_cita'];
    $descripcion = $_POST['descripcion'];

    // include db connect class
    require_once __DIR__ . '/db_connect.php';

    // connecting to db
    $db = new DB_CONNECT();

    // mysql inserting a new row
    $result = mysql_query("INSERT INTO gestorcitas(id_entidad, id_cita,
nombre_clinica, fecha_cita, descripcion) VALUES('$id_entidad',
'$id_cita', '$nombre_clinica','$fecha_cita', '$descripcion')");
    // check if row inserted or not
    if ($result) {
        // successfully inserted into database
        $response["success"] = 1;
        $response["message"] = "Cita successfully created.";

        // echoing JSON response
        echo json_encode($response);
    } else {
        // failed to insert row
        $response["success"] = 0;
        $response["message"] = "Oops! An error occurred.";

        // echoing JSON response
        echo json_encode($response);
    }
} else {
    // required field is missing
    $response["success"] = 0;
    $response["message"] = "Required field(s) is missing";

    // echoing JSON response
    echo json_encode($response);
}
```

```
?>
```

C:\Program Files\EasyPHP-DevServer-13.1VC9\data\localweb\android_connect\db_config.php

```
<?php

/*
 * All database connection variables
 */

define('DB_USER', "root"); // db user
define('DB_PASSWORD', ""); // db password (mention your db password
here)
define('DB_DATABASE', "zwait"); // database name
define('DB_SERVER', "localhost"); // db server
?>
```

C:\Program Files\EasyPHP-DevServer-13.1VC9\data\localweb\android_connect\db_connect.php

```
<?php

/**
 * A class file to connect to database
 */
class DB_CONNECT {

    // constructor
    function __construct() {
        // connecting to database
        $this->connect();
    }

    // destructor
    function __destruct() {
        // closing db connection
        $this->close();
    }

    /**
     * Function to connect with database
     */
    function connect() {
        // import database connection variables
        require_once __DIR__ . '/db_config.php';

        // Connecting to mysql database
        $con = mysql_connect(DB_SERVER, DB_USER, DB_PASSWORD) or
die(mysql_error());

        // Selecing database
        $db = mysql_select_db(DB_DATABASE) or die(mysql_error()) or
die(mysql_error());

        // returing connection cursor
        return $con;
    }

    /**
```

```
     * Function to close db connection
     */
    function close() {
        // closing db connection
        mysql_close();
    }

}

?>
```

C:\Program Files\EasyPHP-DevServer-13.1VC9\data\localweb\android_connect\delete_cita.php

```
<?php

/*
 * Following code will delete a cita from table
 * A cita is identified by cita id (pid)
 */

// array for JSON response
$response = array();

// check for required fields
if (isset($_POST['pid'])) {
    $pid = $_POST['pid'];

    // include db connect class
    require_once __DIR__ . '/db_connect.php';

    // connecting to db
    $db = new DB_CONNECT();

    // mysql update row with matched pid
       $result = mysql_query("DELETE FROM gestorcitas WHERE pid =
$pid");

    // check if row deleted or not
    if (mysql_affected_rows() > 0) {
        // successfully updated
        $response["success"] = 1;
        $response["message"] = "Cita successfully deleted";

        // echoing JSON response
        echo json_encode($response);
    } else {
        // no cita found
        $response["success"] = 0;
        $response["message"] = "No cita found";

        // echo no users JSON
        echo json_encode($response);
    }
} else {
    // required field is missing
    $response["success"] = 0;
    $response["message"] = "Required field(s) is missing";

    // echoing JSON response
    echo json_encode($response);
```

```
}
?>
```

C:\Program Files\EasyPHP-DevServer-13.1VC9\data\localweb\android_connect\get_all_citas_cliente.php

```
<?php

/*
 * Following code will list all the citas
 */

// array for JSON response
$response = array();


// include db connect class
require_once __DIR__ . '/db_connect.php';

// connecting to db
$db = new DB_CONNECT();

// get all citas from citas table
$result = mysql_query("SELECT *FROM gestorcitas") or die(mysql_error());

// check for empty result
if (mysql_num_rows($result) > 0) {
    // looping through all results
    // citas node
    $response["citas"] = array();

    while ($row = mysql_fetch_array($result)) {
        // temp user array
        $cita = array();

            $cita["pid"] = $row["pid"];
        $cita["id_entidad"] = $row["id_entidad"];
        $cita["id_cita"] = $row["id_cita"];
        $cita["nombre_clinica"] = $row["nombre_clinica"];
            $cita["fecha_cita"] = $row["fecha_cita"];
        $cita["descripcion"] = $row["descripcion"];


        $cita["created_at"] = $row["created_at"];
        $cita["updated_at"] = $row["updated_at"];


        // push single cita into final response array
            array_push($response["citas"], $cita);
    }
    // success
    $response["success"] = 1;

    // echoing JSON response
    echo json_encode($response);
} else {
    // no citas found
    $response["success"] = 0;
```

```
    $response["message"] = "No citas found";

    // echo no users JSON
    echo json_encode($response);
}
?>
```

C:\Program Files\EasyPHP-DevServer-13.1VC9\data\localweb\android_connect\get_all_citas_clinica.php

```
<?php

/*
 * Following code will list all the citas
 */

// array for JSON response
$response = array();


// include db connect class
require_once __DIR__ . '/db_connect.php';

// connecting to db
$db = new DB_CONNECT();

// get all citas from gestorcitas table
$result = mysql_query("SELECT *FROM gestorcitas") or die(mysql_error());

// check for empty result
if (mysql_num_rows($result) > 0) {
    // looping through all results
    // citas node
      $response["citas"] = array();

    while ($row = mysql_fetch_array($result)) {
        // temp user array
        $cita = array();

            $cita["pid"] = $row["pid"];
        $cita["id_entidad"] = $row["id_entidad"];
        $cita["id_cita"] = $row["id_cita"];
        $cita["nombre_clinica"] = $row["nombre_clinica"];
            $cita["fecha_cita"] = $row["fecha_cita"];
        $cita["descripcion"] = $row["descripcion"];


        $cita["created_at"] = $row["created_at"];
        $cita["updated_at"] = $row["updated_at"];


        // push single cita into final response array
            array_push($response["citas"], $cita);
    }
    // success
    $response["success"] = 1;
```

```
    // echoing JSON response
    echo json_encode($response);
} else {
    // no citas found
    $response["success"] = 0;
    $response["message"] = "No citas found";

    // echo no users JSON
    echo json_encode($response);
}
?>
```

C:\Program Files\EasyPHP-DevServer-13.1VC9\data\localweb\android_connect\get_detalles_citas.php

```
<?php

/*
 * Following code will get single cita details
 * A cita is identified by cita id (pid)
 */

// array for JSON response
$response = array();


// include db connect class
require_once __DIR__ . '/db_connect.php';

// connecting to db
$db = new DB_CONNECT();

// check for post data
if (isset($_GET["pid"])) {
    $pid = $_GET['pid'];

    // get a cita from gestorcitas table
      $result = mysql_query("SELECT *FROM gestorcitas WHERE pid =
$pid");

    if (!empty($result)) {
        // check for empty result
        if (mysql_num_rows($result) > 0) {

            $result = mysql_fetch_array($result);

            $cita = array();

                    $cita["pid"] = $result["pid"];
                    $cita["id_entidad"] = $result["id_entidad"];
                    $cita["id_cita"] = $result["id_cita"];
                    $cita["nombre_clinica"] = $result["nombre_clinica"];
                    $cita["fecha_cita"] = $result["fecha_cita"];
                    $cita["descripcion"] = $result["descripcion"];


            $cita["created_at"] = $result["created_at"];
            $cita["updated_at"] = $result["updated_at"];
            // success
```

```
            $response["success"] = 1;

            // user node
            $response["cita"] = array();

            array_push($response["cita"], $cita);

            // echoing JSON response
            echo json_encode($response);
        } else {
            // no cita found
            $response["success"] = 0;
            $response["message"] = "No cita found";

            // echo no users JSON
            echo json_encode($response);
        }
    } else {
        // no cita found
        $response["success"] = 0;
        $response["message"] = "No cita found";

        // echo no users JSON
        echo json_encode($response);
    }
} else {
    // required field is missing
    $response["success"] = 0;
    $response["message"] = "Required field(s) is missing";

    // echoing JSON response
    echo json_encode($response);
}
?>
```

C:\Program Files\EasyPHP-DevServer-13.1VC9\data\localweb\android_connect\update_cita.php

```
<?php

/*
 * Following code will update a cita information
 * A cita is identified by cita id (pid)
 */

// array for JSON response
$response = array();

// check for required fields
if (isset($_POST['pid']) &&  isset($_POST['id_entidad']) &&
isset($_POST['id_cita']) && isset($_POST['nombre_clinica']) &&
isset($_POST['fecha_cita']) && isset($_POST['descripcion'])) {

      $pid = $_POST['pid'];
      $id_entidad = $_POST['id_entidad'];
      $id_cita = $_POST['id_cita'];
      $nombre_clinica = $_POST['nombre_clinica'];
      $fecha_cita = $_POST['fecha_cita'];
    $descripcion = $_POST['descripcion'];

    // include db connect class
```

```
    require_once __DIR__ . '/db_connect.php';

    // connecting to db
    $db = new DB_CONNECT();

    // mysql update row with matched pid
      $result = mysql_query("UPDATE gestorcitas SET id_entidad =
'$id_entidad', id_cita = '$id_cita', nombre_clinica = '$nombre_clinica',
fecha_cita ='$fecha_cita', descripcion='$descripcion' WHERE pid =
$pid");

    // check if row inserted or not
    if ($result) {
        // successfully updated
        $response["success"] = 1;
        $response["message"] = "Cita successfully updated.";

        // echoing JSON response
        echo json_encode($response);
    } else {

    }
} else {
    // required field is missing
    $response["success"] = 0;
    $response["message"] = "Required field(s) is missing";

    // echoing JSON response
    echo json_encode($response);
}
?>
```

/Zwait_P16/AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.zwait"

    android:versionCode="1"
    android:versionName="1.0" >

    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="19" />

   <!--  Internet Permissions -->
    <uses-permission android:name="android.permission.INTERNET" />

     <!--  Call_phone Permissions -->
    <uses-permission android:name="android.permission.CALL_PHONE"/>


   <application
       android:allowBackup="true"
        android:configChanges="keyboardHidden|orientation"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >



        <activity
            android:name=".Zwait"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

         <activity
            android:label="Menu"
            android:name=".MenuAct"></activity>

          <activity
            android:label="Acerca de"
            android:name=".AcercaDe"
             android:theme="@android:style/Theme.Dialog"></activity>



        <activity
            android:label="Cliente"
            android:name=".ClienteAct"></activity>

          <activity
            android:label="Agenda"
            android:name=".AgendaAct"></activity>
```

```
          <activity
            android:label="Clinica"
            android:name=".ClinicaAct"></activity>

        <activity
            android:name="com.zwait.AyudaAct"></activity>



        <activity
            android:label="Clínica"
            android:name="com.zwait.clinica.ClinicaActivity"
></activity>

         <!--  Login Activity -->
        <activity
            android:label="Login Account"
            android:name=".LoginActivity"></activity>

        <!--  Register Activity -->
        <activity
            android:label="Register New Account"
            android:name=".RegisterActivity"></activity>




           <!-- All Citas Activity -->
        <activity
            android:name="com.zwait.clinica.TablonCitasActivity"
            android:label="Todas Las Citas" >
        </activity>


         <!-- Add Cita Activity -->
        <activity
            android:name="com.zwait.clinica.NuevaCitaActivity"
            android:label="Añadir Nueva Cita" >
        </activity>


            <!-- Edit Cita Activity -->
        <activity
            android:name="com.zwait.clinica.EditarCitaActivity"
            android:label="Editar Cita" >
        </activity>



         <!-- IntencionesActivity -->
        <activity
            android:name=".IntencionesAct"
            android:label="Servicios" >
        </activity>

           <!-- IntencionesClienteActivity -->
        <activity
            android:name=".IntencionesClienteAct"
            android:label="Servicios Cliente" >
        </activity>
```

```
        <!-- ClienteActivity -->
        <activity
            android:name="com.zwait.cliente.ClienteActivity"
            android:label="Cliente" >
        </activity>


        <!-- Ver Citas Activity -->
        <activity
            android:name="com.zwait.cliente.VerCitaActivity"
            android:label="Ver Cita" >
        </activity>

        <!-- All Citas Activity -->
        <activity
            android:name="com.zwait.cliente.TablonCitasClienteActivity"
            android:label="Todas Las Citas" >
        </activity>

            <!-- Add Cita Activity -->
        <activity
            android:name="com.zwait.cliente.NuevaCitaClienteActivity"
            android:label="Añadir Nueva Cita" >
        </activity>


    </application>


</manifest>
```

Web

C:\Program Files\EasyPHP-DevServer-13.1VC9\data\localweb\zwait_db\acceso.php

```
<?php
session_start();
if(!isset($_SESSION['acceso'])or $_SESSION['acceso']==false){
      header("LOCATION:entrada.php");
      }
      else{
            echo"Bienvenido al sistema";
            ?>
            <a href="desactivacion.php">Desconectar</a>
            <?php
            }
            ?>
<html><head></head><body>
</body>
</html>
```

C:\Program Files\EasyPHP-DevServer-13.1VC9\data\localweb\zwait_db\activación.php

```
<html><head></head><body>
<?php
include("conexion.php"); // Establece la conexión
session_start();
$_SESSION['acceso']=false; // pone la variable acceso a FALSE

$v01=$_GET['usuario'];  // Recogemos los datos
$v02=$_GET['clave'];

// Buscamos el usuario y la clave en la tabla usuarios
$ssql="select * from usuarios where usuario='$v01' and clave='$v02'";

mysql_query($ssql,$descriptor);
$resultado=mysql_query($ssql,$descriptor);
$row=mysql_fetch_array($resultado);

if($row['usuario']<>null){ // Si hay coincidencia ponemos la variable
ACCESO a TRUE
      $_SESSION['acceso']=true;
      }else{
            $_SESSION['acceso']=false;}
            header("LOCATION:entrada.php");     // redireccionamos la
web a la página de entrada
?>

</body>
</html>
```

C:\Program Files\EasyPHP-DevServer-13.1VC9\data\localweb\zwait_db\altas.php

```
<?php
session_start();
if(!isset($_SESSION['acceso'])or $_SESSION['acceso']==false){
      header("LOCATION:entrada.php");
      }
      else{
            echo"Bienvenido al sistema";
            ?>
            <a href="desactivacion.php">Desconectar</a>
            <?php
            }
            ?>
<?php
// Estableciendo la conexión
include("conexion.php");

// Recogida de datos
$id_entidad=$_POST['id_entidad'];
$id_cita=$_POST['id_cita'];
$nombre_clinica=$_POST['nombre_clinica'];
$fecha_cita=$_POST['fecha_cita'];
$descripcion=$_POST['descripcion'];



// nombre y ruta de la imagen
//$ima="./images/".$cod.".jpg";

// creamos la sentencia SQL
$ssql="insert into gestorcitas(id_entidad, id_cita, nombre_clinica,
fecha_cita,
descripcion)values('$id_entidad','$id_cita','$nombre_clinica','$fecha_ci
ta','$descripcion')";

// ejecutamos la sentencia SQL
mysql_query($ssql,$descriptor);

// copiamos la imagen que nos ha llegado a su carpeta
/*$ruta=$ima;
COPY($_FILES["imagen"]["tmp_name"],$ruta);*/

// redireccionar a la web listados
header("LOCATION:listado.php");

?>
```

C:\Program Files\EasyPHP-DevServer-13.1VC9\data\localweb\zwait_db\bajas.php

```
<?php
session_start();
if(!isset($_SESSION['acceso'])or $_SESSION['acceso']==false){
      header("LOCATION:entrada.php");
      }
      else{
            echo"Bienvenido al sistema";
            ?>
```

```
            <a href="desactivacion.php">Desconectar</a>
            <?php
            }
            ?>
<?php
// Estableciendo la conexión
include("conexion.php");
// recogida de datos
$id_entidad=$_POST['id_entidad'];
$id_cita=$_POST['id_cita'];
// creamos la sentencia SQL
$ssql="DELETE FROM gestorcitas WHERE id_entidad='$id_entidad' and
id_cita='$id_cita'" ;
// ejecutamos la sentencia SQL
mysql_query($ssql,$descriptor);
// redireccionamos a la pagina:listado.php
header("LOCATION:listado.php");
?>
```

C:\Program Files\EasyPHP-DevServer-13.1VC9\data\localweb\zwait_db\conexion.php

```
<?php
$servidor="localhost";
$usuario="root";
$clave="";
$basedatos="zwait";

$descriptor=mysql_connect($servidor,$usuario,$clave);
mysql_select_db($basedatos);
?>
```

C:\Program Files\EasyPHP-DevServer-13.1VC9\data\localweb\zwait_db\desactivacion.php

```
<html><head></head><body>
<?php
//ponemos la varible global ACCESO a FALSE y nos redirecciona de nuevo a
entrada.php
include("conexion.php");
session_start();
$_SESSION['acceso']=false;
header("LOCATION:entrada.php");
?>

</body>
</html>
```

C:\Program Files\EasyPHP-DevServer-13.1VC9\data\localweb\zwait_db\entrada.php

```
<html><head></head>
<body bgcolor="#6495ED">
<center>
<h1>Zwait</h1>
<h2> Tu gestor de restrasos para citas </h2>
<IMG SRC="http://localhost/zwait_db/images/Icon_64.png"><br>
```

```
<?php
	session_start();

	if(!isset($_SESSION['acceso']) or $_SESSION['acceso']==false){

?>
	<form action="activacion.php" method="get">
		<p>Usuario<input name="usuario" type="text"/><br>
			Clave<input name="clave" type="password"/><br>
			<input type="submit" name="button" value="Enviar"/>
			<input type="reset" name="button2" value="Borrar"/>
		</p>
	</form>

<?php
	}elseif($_SESSION['acceso']==true){
		echo "Bienvenido al sistema ";
?>
<a href="desactivacion.php"> Desconectar</a><br>
<a href="menu.php"> Menu</a>


<?php } ?>

</body>
</html>
```

C:\Program Files\EasyPHP-DevServer-13.1VC9\data\localweb\zwait_db\formulario_altas.php

```
<?php
session_start();
if(!isset($_SESSION['acceso'])or $_SESSION['acceso']==false){
	header("LOCATION:entrada.php");
	}
	else{
		echo"Bienvenido al sistema";
		?>
		<a href="desactivacion.php">Desconectar</a>
		<?php
		}
		?>
<html><head></head>
	<body bgcolor="#6495ED">
<center>
<h1>Zwait</h1>
<h2> Tu gestor de restrasos para citas </h2>
<IMG SRC="http://localhost/zwait_db/images/Icon_64.png">



		<form action="altas.php" method="post"
enctype="multipart/form-data">
			<table width="400" border="0">
			<tr><td colspan="4"> ALTA DE CITAS</td></tr>
			<tr>
				<td> Id_entidad </td><td><input type="text"
name="id_entidad"/></td>
				<td> Id_cita </td><td><input type="text"
name="id_cita"/></td>
			</tr>
```

```
                  <tr>
                        <td> Nombre_cl&iacute;nica </td><td colspan="3">
<input type="text" name="nombre_clinica"/></td>
                  </tr>

                  <tr>
                        <td> Fecha_cita </td><td><input type="text"
name="fecha_cita"/></td>
                        <td> Descripci&oacute;n </td><td><input
type="text" name="descripcion"/></td>
                  </tr>

                  <!-- <tr>
                        <td> Imagen </td><td colspan="3"> <input
type="file" name="imagen"/></td>
                  </tr> -->


                  <tr>
                        <td colspan="2"><input type="submit"
name="button" value="Enviar"/></td>
                        <td colspan="2"><input type="reset"
name="button2" value="Restablecer"/></td>
                  </tr>

                  </table>
                  </form>
                  </body>
                  </html>
```

C:\Program Files\EasyPHP-DevServer-13.1VC9\data\localweb\zwait_db\formulario_bajas.php

```
<?php
session_start();
if(!isset($_SESSION['acceso'])or $_SESSION['acceso']==false){
      header("LOCATION:entrada.php");
      }
      else{
            echo"Bienvenido al sistema";
            ?>
            <a href="desactivacion.php">Desconectar</a>
            <?php
            }
            ?>
<html><head></head>
<body bgcolor="#6495ED">
<center>
<h1>Zwait</h1>
<h2> Tu gestor de restrasos para citas </h2>
<IMG SRC="http://localhost/zwait_db/images/Icon_64.png">
      <form action="bajas.php" method="post" enctype="multipart/form-
data" name="form1">

            <table width="400" boder="0">
```

```
                        <tr><td colspan="2"> BAJAS DE CITAS </td></tr>
                        <tr>
                              <td> Introduzca Id_entidad </td>
                              <td><input type="text" name="id_entidad"
id="id_entidad"/></td>
                        </tr>

                        <tr>
                              <td> Introduzca Id_cita </td>
                              <td><input type="text" name="id_cita"
id="id_cita"/></td>
                        </tr>

                        <tr>
                              <td colspan="2"><input type="submit"
name="button" id="button" value="Enviar"/></td>
                              <td colspan="2"><input type="reset"
name="button2" id="button2" value="Restablecer"/></td>
                        </tr>

            </table>

      </form>

</body>
</html>
```

C:\Program Files\EasyPHP-DevServer-13.1VC9\data\localweb\zwait_db\formulario_modificación.php

```
<?php
session_start();
if(!isset($_SESSION['acceso'])or $_SESSION['acceso']==false){
      header("LOCATION:entrada.php");
      }
      else{
            echo"Bienvenido al sistema";
            ?>
            <a href="desactivacion.php">Desconectar</a>
            <?php
            }
            ?>
<html><head></head>
<body bgcolor="#6495ED">
<center>
<h1>Zwait</h1>
<h2> Tu gestor de restrasos para citas </h2>
<IMG SRC="http://localhost/zwait_db/images/Icon_64.png">
<?php
// Establecimiento de la conexion
include("conexion.php");
// recogida de datos
$id_entidad=$_POST['id_entidad'];
$id_cita=$_POST['id_cita'];
// creamos la sentencia SQL
```

```
$ssql="select * from gestorcitas where id_entidad='$id_entidad' and
id_cita='$id_cita'";
// ejecutamos la sentencia SQL
$resultado=mysql_query($ssql,$descriptor);
$row=mysql_fetch_array($resultado);
?>
            <form action="modificacion.php" method="post"
enctype="multipart/form-data">
                  <table width="400" border="0">
                  <tr><td colspan="4"> MODIFICAR CITA SELECCIONADA
</td></tr>
                  <tr>
                        <td> Id_entidad </td>
                        <td><input name="id_entidad" type="text"
value="<?php echo $row['id_entidad'];?>" readonly="readonly"/></td>

                        <td> Id_cita </td>
                        <td><input name="id_cita" type="text"
value="<?php echo $row['id_cita'];?>" readonly="readonly"/></td>
                        <!-- <td> Id_cita </td><td><input type="text"
name="id_cita" id="id_cita" value="<?php echo $row['id_cita'];?>"/></td>
-->
                  </tr>

                  <tr>
                        <td> nombre_cl&iacute;nica </td><td colspan="3">
<input type="text" name="nombre_clinica" type="text" value="<?php echo
$row['nombre_clinica'];?>"/></td>
                  </tr>

                  <tr>
                        <td> Fecha_cita </td><td><input
name="fecha_cita" type="text" value="<?php echo $row['fecha_cita'];?>"
/></td>
                        <td> Descripci&oacute;n </td><td><input
type="text" name="descripcion" value="<?php echo $row['descripcion'];?>"
/></td>
                  </tr>


                  <tr>
                        <td colspan="2"><input type="submit"
name="button" value="Enviar"/></td>
                        <td colspan="2"><input type="reset"
name="button2" value="Restablecer"/></td>
                  </tr>

                  </table>
            </form>
      </body>
</html>
```

C:\Program Files\EasyPHP-DevServer-13.1VC9\data\localweb\zwait_db\formulario_recursivo.php

```
<?php
session_start();
if(!isset($_SESSION['acceso'])or $_SESSION['acceso']==false){
      header("LOCATION:entrada.php");
      }
      else{
            echo"Bienvenido al sistema";
            ?>
            <a href="desactivacion.php">Desconectar</a>
            <BR>
            <BR>
            <?php
            }
            ?>
<html>
<head></head>
<body bgcolor="#6495ED">
<center>
<h1>Zwait</h1>
<h2> Tu gestor de restrasos para citas </h2>
<IMG SRC="http://localhost/zwait_db/images/Icon_64.png"><br>

<!-- Condicional -->
<?php

if(isset($_POST['botonenviar'])){
// Estableciendo la conexión
include("conexion.php");

// Recogida de datos
$id_entidad=$_POST['id_entidad'];
$id_cita=$_POST['id_cita'];
$nombre_clinica=$_POST['nombre_clinica'];
$fecha_cita=$_POST['fecha_cita'];
$descripcion=$_POST['descripcion'];

// nombre y ruta de la imagen
//$ima="./images/".$cod.".jpg";

// creamos la sentencia SQL
$ssql="insert into gestorcitas(id_entidad, id_cita, nombre_clinica,
fecha_cita, descripcion) values('$id_entidad', '$id_cita',
'$nombre_clinica', '$fecha_cita', '$descripcion')";

// ejecutamos la sentencia SQL
mysql_query($ssql,$descriptor);

// copiamos la imagen que nos ha llegado a su carpeta
//$ruta=$ima;
//COPY($_FILES["imagen"]["tmp_name"],$ruta);


}



//<!-- Sección de listado -->
```

```php
// Estableciendo la conexión
include("conexion.php");

// creamos la sentencia SQL y su ejecución
$ssql="select * from gestorcitas";
$resultado=mysql_query($ssql,$descriptor);

// recorremos todo el array
while($row=mysql_fetch_array($resultado)){
      echo "Id_entidad = ".$row['id_entidad']."<BR>";
      echo "Id_cita = ".$row['id_cita']."<BR>";
      echo "Nombre_Cl&iacute;nica = ".$row['nombre_clinica']."<BR>";
      echo "Fecha_cita = ".$row['fecha_cita']."<BR>";
      echo "Descripci&oacute;n = ".$row['descripcion']."<BR>"."<BR>";

}


?>
<!-- Sección de formulario -->
<html><head></head>
      <body>
            <form action="formulario_recursivo.php" method="post"
enctype="multipart/form-data">
                  <table width="400" border="0">
                  <tr><td colspan="4"> ALTA DE CITAS (recursivo)
</td></tr>
                  <tr>
                        <td> Id_entidad </td><td><input type="text"
name="id_entidad"/></td>
                        <td> Id_cita </td><td><input type="text"
name="id_cita"/></td>
                  </tr>

                  <tr>
                        <td> Nombre_cl&iacute;nica </td><td colspan="3">
<input type="text" name="nombre_clinica"/></td>
                  </tr>

                  <tr>
                        <td> fecha_cita </td><td><input type="text"
name="fecha_cita"/></td>
                        <td> descripci&oacute;n </td><td><input
type="text" name="descripcion"/></td>
                  </tr>

                  <!-- <tr>
                        <td> Imagen </td><td colspan="3"> <input
type="file" name="imagen"/></td>
                  </tr> -->


                  <tr>
                        <td colspan="2"><input type="submit"
name="botonenviar" id="botonenviar" value="Enviar"/></td>
                        <td colspan="2"><input type="reset"
name="button2" id="button2" value="Restablecer"/></td>
                  </tr>

                  </table>
```

```
                </form>
                </body>
                </html>


</body>
</html>
```

C:\Program Files\EasyPHP-DevServer-13.1VC9\data\localweb\zwait_db\formulario_seleccion_registro.php

```
<?php
session_start();
if(!isset($_SESSION['acceso'])or $_SESSION['acceso']==false){
      header("LOCATION:entrada.php");
      }
      else{
            echo"Bienvenido al sistema";
            ?>
            <a href="desactivacion.php">Desconectar</a>
            <?php
            }
            ?>
<html><head></head>
<body bgcolor="#6495ED">
<center>
<h1>Zwait</h1>
<h2> Tu gestor de restrasos para citas </h2>
<IMG SRC="http://localhost/zwait_db/images/Icon_64.png">
      <form action="formulario_modificacion.php" method="post"
enctype="multipart/form-data" name="form1">

            <table width="400" boder="0">
            <br>
                        <tr><td colspan="2"> MODIFICACI&Oacute;N DE
CITAS </td></tr>
                        <tr>
                              <td> Introduzca Id_entidad e Id_cita </td>
</tr>
                        <tr>
                              <td><input type="text" name="id_entidad"
id="id_entidad"/></td>
                              <td><input type="text" name="id_cita"
id="id_entidad"/></td>
                        </tr>

                        <tr>
                              <td><input type="submit" name="button"
id="button" value="Enviar"/></td>
                              <td><input type="reset" name="button2"
id="button2" value="Restablecer"/></td>
                        </tr>

            </table>

      </form>
```

```
</body>
</html>
```

C:\Program Files\EasyPHP-DevServer-13.1VC9\data\localweb\zwait_db\listado.php

```
<html><head>

<title>Listado de citas</title>
</head>
<body  bgcolor="#6495ED">
<center>
<h1>Zwait</h1>
<h2> Tu gestor de restrasos para citas </h2>
<IMG SRC="http://localhost/zwait_db/images/Icon_64.png">

<h3>Listado de citas</h3>

<?php
// Estableciendo la conexión
include("conexion.php");

// creamos la sentencia SQL
$ssql="select * from gestorcitas";
$resultado=mysql_query($ssql,$descriptor);


while($row=mysql_fetch_array($resultado)){
      echo "Id_entidad = ".$row['id_entidad']."<BR>";
      echo "Id_cita = ".$row['id_cita']."<BR>";
      echo "Nombre_Cl&iacute;nica = ".$row['nombre_clinica']."<BR>";
      echo "Fecha_cita = ".$row['fecha_cita']."<BR>";
      echo "Descripci&oacute;n = ".$row['descripcion']."<BR>"."<BR>";

}


?>
</body>
</html>
```

C:\Program Files\EasyPHP-DevServer-13.1VC9\data\localweb\zwait_db\listado.php

```
<html><head>

<title>Listado de citas</title>
</head>
<body  bgcolor="#6495ED">
<center>
<h1>Zwait</h1>
<h2> Tu gestor de restrasos para citas </h2>
<IMG SRC="http://localhost/zwait_db/images/Icon_64.png">

<h3>Listado de citas</h3>
```

```php
<?php
// Estableciendo la conexión
include("conexion.php");

// creamos la sentencia SQL
$ssql="select * from gestorcitas";
$resultado=mysql_query($ssql,$descriptor);


while($row=mysql_fetch_array($resultado)){
      echo "Id_entidad = ".$row['id_entidad']."<BR>";
      echo "Id_cita = ".$row['id_cita']."<BR>";
      echo "Nombre_Cl&iacute;nica = ".$row['nombre_clinica']."<BR>";
      echo "Fecha_cita = ".$row['fecha_cita']."<BR>";
      echo "Descripci&oacute;n = ".$row['descripcion']."<BR>"."<BR>";

}


?>
</body>
</html>
```

C:\Program Files\EasyPHP-DevServer-13.1VC9\data\localweb\zwait_db\menu.php

```php
<html>
      <head></head>
      <body bgcolor="#6495ED">
<center>
<h1>Zwait</h1>
<h2> Tu gestor de restrasos para citas </h2>
<IMG SRC="http://localhost/zwait_db/images/Icon_64.png"><br>

            <p><a href="listado.php">Listado de Citas</a></p>
            <p><a href="formulario_altas.php">Altas de Citas</a></p>
            <p><a href="formulario_bajas.php">Bajas de Citas</a></p>
            <p><a
href="formulario_seleccion_registro.php">Modificaci&oacute;n de
Citas</a></p>
            <p><a href="formulario_recursivo.php">Formulario
Altas(Recursivo)</a></p>

</body>
</html>
```

C:\Program Files\EasyPHP-DevServer-13.1VC9\data\localweb\zwait_db\modificacion.php

```php
<?php
session_start();
if(!isset($_SESSION['acceso'])or $_SESSION['acceso']==false){
      header("LOCATION:entrada.php");
      }
      else{
            echo"Bienvenido al sistema";
            ?>
            <a href="desactivacion.php">Desconectar</a>
            <?php
```

```
            }
            ?>
<html><head></head><body>
<?php

// Estableciendo la conexión
include("conexion.php");
$id_entidad=$_POST['id_entidad'];
$id_cita=$_POST['id_cita'];
$nombre_clinica=$_POST['nombre_clinica'];
$fecha_cita=$_POST['fecha_cita'];
$descripcion=$_POST['descripcion'];

// creamos la sentencia SQL
$ssql="UPDATE gestorcitas SET id_cita='$id_cita',
nombre_clinica='$nombre_clinica', fecha_cita='$fecha_cita',
descripcion='$descripcion' where id_entidad='$id_entidad'";

// ejecutamos la sentencia SQL
mysql_query($ssql,$descriptor);

// redireccionar a la web listados
header("LOCATION:listado.php");

?>
</body>
</html>
```

Bases de datos

```
CREATE TABLE gestorcitas (
      pid int(11) primary key auto_increment,
      id_entidad varchar(100) not null,
      id_cita varchar(100) not null,
      nombre_clinica varchar(100) not null,
      fecha_cita datetime not null,
      descripcion text,
      created_at timestamp default now(),
      updated_at timestamp
);
create table users(
      uid int(11) primary key auto_increment,
      unique_id varchar(30) not null unique,
      name varchar(60) not null,
      email varchar(100) not null unique,
      encrypted_password varchar(90) not null,
      salt varchar(10) not null,
      created_at datetime,
      updated_at datetime null
);
```