Universidad de Valladolid

Escuela Técnica Superior
Ingenieros de Telecomunicación

Trabajo de fin de grado

Grado en Ingeniería de Tecnologías
Específicas de Telecomunicación
Mención en Ingeniería Telemática

Aplicación Android del sistema de evaluación de contenidos: e-Liza

Autor: Julio Dehesa Martín

Tutora: Míriam Antón Rodríguez

TÍTULO: Aplicación Android del Sistema de Evaluación de Contenidos: e-Liza

AUTOR: JULIO DEHESA MARTÍN

TUTOR: MÍRIAM ANTÓN RODRÍGUEZ

DEPARTAMENTO: Teoría de la Señal y Comunicaciones e Ingeniería Telemática

---

MIEMBROS DEL TRIBUNAL:

PRESIDENTE: Míriam Antón Rodríguez

SECRETARIO: David González Ortega

VOCAL: Mario Martínez Zarzuela

SUPLENTE: María Ángeles Pérez Juárez

FECHA: 18-05-2015

CALIFICACIÓN:

Resumen del Proyecto

Este trabajo de fin de grado nace de la necesidad del alumnado de desarrollar las tareas de aprendizaje y evaluación. Dados los avances tecnológicos que se derivan de Internet y de los múltiples dispositivos que manejamos a diario, la enseñanza tal y como se conocía hace bien poco ha cambiado. En la actualidad gracias a los ordenadores, dispositivos móviles y el uso que éstos hacen de Internet, la enseñanza ha diversificado sus métodos para resultar más cómoda para los docentes, más instructiva para el alumnado y más accesible para todos.

Esta herramienta de evaluación basada en *GBL* (Aplicación del aprendizaje basada en juegos) se desarrolló para el gestor de contenidos educativos *Moodle*, que aún sigue operativo.

El desarrollo del trabajo de fin de grado ha tenido como actividad principal la de proyectar esta herramienta en su uso para dispositivos móviles, acompañándola además de resultados estadísticos personalizados que facilitarán al alumno seguir su progresión. También se ha incluido, como herramienta adicional (con el único fin de motivar al alumno al uso de la aplicación móvil), una sección de logros que el alumno irá completando conforme vaya obteniendo buenos resultados en su evaluación.

Todo esto, desde el punto de vista del desarrollador de la aplicación, se traduce en la creación, acceso y actualización de datos alojados en las bases de datos de *e-Liza,* ya existentes, usando el servicio *Web* REST. Algunos de las funcionalidades mencionadas anteriormente no precisan de acceso *Web* por lo que, por simplicidad y coherencia en la gestión de datos, se ha implementado una base de datos local que reside en el dispositivo móvil de la aplicación.

Esta versión paralela de e-Liza no sustituye a la versión web de *Moodle* si no que convive con ella y en parte la complementa, dotando a la aplicación de funcionalidades adicionales que el alumno podrá usar en la versión móvil.

El objetivo principal de esta aplicación, es el de servir de apoyo a la metodología de aprendizaje de las asignaturas impartidas por el Grupo de Telemática e Imagen en la Universidad de Valladolid y también es el de diversificar los métodos de aprendizaje disponibles para los alumnos.

PALABRAS CLAVE

Android, e-Liza, *MySQL*, *SQLite*, PHP, REST, *eLearning*, Java.

Abstract

This project was born from the need of students to perform learning and evaluation tasks. Given the technological advances derived from Internet and the many devices we handle every day, the teaching as it was understood very recently, has changed. Today, thanks to computers, mobile devices and the use that they do of Internet, the teaching has diversified its methods to be more comfortable for teachers, more instructive for students and more accessible for everyone.

This evaluation tool based on GBL (*Game Based Learning*) was developed for the educational content manager called *Moodle*, that it is still operative.

The development of the end of grade work aims at the projection of the eLiza tool on the mobile platforms, giving customized statistical results to eLiza that will allow the student to follow its progression. It has been also included, as an additional tool (with an unique purpose: to motivate the students to use the mobile application), an achievements section that will be completed by the students as they reach good results in them evaluation.

From the developer point of view, all of this is translated to the data creation, access and updating that are hosted in the already existing e-Liza data bases, using the REST Web service. Some of the previous mentioned functionalities do not need web access and because of that, a local data base has been implemented on the application local device, due to simplicity and fastness.

This eLiza parallel version does not substitute to the current Moodle web version, it coexists with it and partly complements it, giving it additional functionalities that can be used by the students in the mobile version.

The application goal is to provide support to the learning methodology of the subjects given by the Grupo de Telemática e Imagen in the Valladolid University and it is also aimed to diversify the learning methods that are enable for students.

Key words

Android, e-Liza, *MySQL*, *SQLite*, PHP, REST, *eLearning*, Java.

Gracias a mi madre por su apoyo incondicional y paciencia.

Gracias a mi pareja por estar siempre a mi lado.

Gracias a mis compañeros por todas las experiencias que hemos compartido.

Gracias a Miriam Antón Rodríguez por darme la oportunidad de realizar este trabajo de fin de grado.

Índice

Índice de Ilustraciones

Índice de tablas

Índice de diagramas

"La programación de hoy es una carrera entre ingenieros de software luchando para construir programas más grandes, mejores y aprueba de idiotas, contra el universo intentando construir idiotas mejores y más grandes. Hasta ahora, el universo va ganando".

Rick Cook, autor estadounidense en The Wizardry Compiled

Parte 1: Introducción y conocimientos previos.

Capítulo 1: Introducción. Motivación. Objetivos.

La creación de Internet ha supuesto un cambio evolutivo en la forma, lugar y duración del desempeño de nuestras actividades, tanto laborales como de ocio y educativas. Hoy en día nos encontramos con que muchas de las aplicaciones *Web* que usábamos años atrás, antes de la irrupción de los *smartphones*, tienen su versión móvil.

El principal motivo de esta duplicidad de aplicaciones es la accesibilidad que confiere a estos servicios estar presentes en la plataforma móvil. El usuario puede acceder desde su terminal móvil a los contenidos *Web* mediante aplicaciones que adaptan el uso de los servicios de escritorio a su terminal. Son numerosos los casos en prensa online, redes sociales y *Webs* de compra y venta entre otros.

Por todo ello, es lógico que el campo de la educación también se vea afectada por esta apremiante actualización de los canales de comunicación con el público final. En la actualidad la Universidad de Valladolid ya dispone de ciertas aplicaciones móviles como el acceso a recursos *Moodle*, la consulta de notas UVa y PAU y la aplicación BiblioUVa. Con la realización de este trabajo de fin de grado, la Universidad de Valladolid sigue extendiendo la presencia de los contenidos *Web* en los terminales móviles.

Este trabajo de fin de grado parte del módulo *e-Liza* ya desarrollado para *Moodle* el cual fue denominado en un primer momento como *Jeopardi* (inspirado en un concurso de televisión de preguntas y respuestas) y que posteriormente cambió su nombre por *e-Liza*. El módulo *e-Liza* está basado en el concepto GBL de aprendizaje basado en juegos. Esto se conoce habitualmente como *e-Learning*, filosofía de aprendizaje de la cual participan multitud de plataformas. En la actualidad una de las más utilizadas es *Moodle*.

Definimos *e-Learning* como el conjunto de los procesos de enseñanza llevados a cabo a través de Internet, en los cuales el personal docente y el alumnado no tienen contacto físico pero sí están comunicados (Barbera, 2008). Esta comunicación se puede realizar de forma

síncrona o asíncrona y mediante ella, se lleva a cabo una función didáctica continuada. Las principales ventajas del e-*Learning* son (Sevilla):

- Desaparecen las barreras espacio-temporales: debido al acceso a los contenidos en cualquier momento y lugar.
- La formación es flexible: debido a la diversificación de los métodos de aprendizaje.
- El alumno decide: es quien elige qué aprender y cuándo. Participando de manera activa en la elección de su itinerario de aprendizaje.
- La actualización de contenidos, al contrario de lo que sucede con los libros de texto, tiene coste 0 y es muy sencilla y rápida.

Con la realización de este trabajo de fin de grado se pretende trasladar el módulo de *e-Liza* existente en *Moodle* a los dispositivos de sistema operativo Android, de modo que fuese más accesible para los alumnos y también le confiriese una motivación extra al alumno en su *e-Learning*. De modo que tanto el módulo *Web* como la aplicación Android conviviesen, pudiendo ser usadas indistintamente por el usuario. No sólo se han trasladado las funciones existentes sino que además se han desarrollado funciones adicionales con el único objetivo de motivar al alumno a usar esta aplicación móvil. Las nuevas funcionalidades son una clasificación ajustada al curso y asignatura del alumno, en la que se muestra el puntaje obtenido por cada usuario. También, como novedad, se ha incluido un apartado de logros, que al igual que la clasificación se ha divido por asignaturas. Cada logro tiene 3 niveles de superación bronce, plata y el logro se verá completado con el oro. Con este último apartado lo que pretendemos es que el alumno se empeñe en la consecución del oro en todos y cada uno de los logros y así fomentar su aprendizaje.

Capítulo 2: Conocimientos previos

Al iniciar el proyecto ya tenía ciertos conocimientos de programación de aplicaciones en Android, ya que cursé la asignatura de Desarrollo de Aplicaciones para Dispositivos Móviles.

Entre los conocimientos con los que inicié la realización de este trabajo de fin de grado está la creación de *activities.* Los *activities* son parte fundamental de cualquier aplicación Android, en la que como mínimo habrá una Activity. Conocía su estructura general de métodos, así como su ciclo de vida. Esto comprende, desde su creación en la llamada a *onCreate,* en el que se inician los elementos de la interfaz de usuario y las funcionalidades de arranque de la actividad, hasta su parada o pausa, en las que se realizan tareas de salvado de datos. Para mostrar la interfaz de usuario de la forma más sencilla usaba estructuras estáticas definidas en XML en los denominados *layouts*. En ellos situaba elementos como campos de edición de texto, texto o botones con los que el usuario interactuaba. De la interacción del usuario con estos elementos se realizan llamadas asíncronas a métodos sujetos a eventos.

También poseía nociones básicas sobre el paso de parámetros entre *activities*, el ciclo de vida de una aplicación en Android y la interacción entre actividades. Por el contrario

desconocía el uso de *threads* de modo que el programa discurría a lo largo de un único hilo que soportaba todas las operaciones de la aplicación de forma secuencial. Relacionado con la interacción entre *activities*¸ el manejo de actividades del *stack* de *activities* de Android era otro de los conceptos que ya había manejado.

Otro de los procesos de Android que conocía desde el inicio del trabajo era el acceso a *content providers*. Estos proveedores de contenido solamente son necesarios y útiles cuando la aplicación comparte datos con otras aplicaciones. En la realización de la aplicación Android de *eLiza* no ha sido necesaria la implementación de proveedores de contenido, ya que opera independientemente de otras aplicaciones.

Capítulo 3: Descripción del problema

Internet comenzó dando servicio a *Webs* alojadas en Internet a las que en un principio accedíamos mediante un equipo de escritorio y posteriormente con ordenadores portátiles. Unos años más tarde irrumpió el *smartphone.* Este terminal móvil nos permite acceder a cualquier recurso alojado en los servidores de contenidos de Internet. De este modo, la evolución tecnológica muestra una clara tendencia, desde los comienzos de Internet, aumentar la accesibilidad de sus contenidos (Emmanuel Bertin y Noel Crespi, 2013).

Si bien es cierto que el acceso a estos dispositivos no está al alcance de todo el mundo, hoy en día es complicado dar con una persona que no maneje un teléfono móvil con acceso a Internet. Pese a la crisis económica, el mercado de la venta de dispositivos móviles es uno de los que se mantiene a flote a pesar de la tempestad. Las compañías telefónicas y los fabricantes de dispositivos han sabido crear la necesidad del uso de *smartphones* en la población. Uno de los factores desencadenantes de esta necesidad es el acceso a contenidos y servicios que se presta de forma gratuita, mediante aplicaciones, en los *smartphones* y *tablets*.

En este contexto nos encontramos en un mundo constantemente conectado, la globalización ha dispuesto todos los contenidos imaginables de forma rápida y sencilla en las manos de todo el mundo. En la actualidad la población tiene conocimiento del uso de estas tecnologías, y día a día la sección de población usuaria de *tablets* y *smartphones* crece año tras año. Como consecuencia de la expansión de los *smartphones*, son muchas *Web* las que tienen su réplica en versión móvil. De este modo sus contenidos en Internet pueden ser accedidos indistintamente desde dos plataformas diferentes. Así en cada caso la información se adecúa al terminal de acceso y se muestra de la forma más adecuada. Los casos de movilidad de páginas *Web* a una plataforma móvil son numerosos. Prácticamente podemos acceder desde el móvil a los mismos servicios que desde un ordenador.

El sector de la educación no es una excepción en la expansión tecnológica que se está experimentando. Los estudios demuestran que el uso de nuevas metodologías de docencia, basadas en la tecnología, favorece el aprendizaje y al mismo tiempo reducen el tiempo de instrucción y los costos de la enseñanza (Sevilla). Algunos alumnos son capaces de formarse fácilmente usando herramientas escritas u orales, pero a aquellos a los que estos métodos les

resultan poco motivadores encuentran un gran impulso a su formación en el uso de medios tecnológicos de docencia. Debido a las nuevas necesidades laborales que se desprenden del avance tecnológico, es aconsejable que los alumnos sean formados mediante un método de aprendizaje basado en la participación activa, la investigación y la experimentación.

Por todo esto, se hace necesaria la actualización de los métodos de formación, de modo que se adecúen a la expansión tecnológica y aprovechen todas las ventajas que éstos ponen a nuestro alcance.

“La mayoría de los buenos programadores no programan por esperar una paga o adulación del público, lo hacen porque programar es divertido.”

Linus Torvalds, ingeniero de software, desarrollador del kernel de Linux.

Parte 2: Trabajo desarrollado

Capítulo 4: Teoría

1. Los dispositivos móviles

1. Introducción

Es evidente que los dispositivos móviles han cambiado el modo de relacionarse con Internet, las necesidades e incluso el modo en el que nos relacionamos entre nosotros. Esta revolución es equiparable a la que es su día produjo Internet en su aparición y parece que esto es sólo la punta del iceberg. Los dispositivos móviles no dejan de mejorar, cada vez son más potentes, tienen una mejor cámara y más capacidad de almacenamiento. Y lo mejor de todo, el mercado sigue aceptando esta constante evolución porque la sociedad así lo demanda. Esto provoca que la creación de aplicaciones para dispositivos móviles sea una fuente inagotable de trabajo para los desarrolladores (Klemens, 2010).

2. Historia

El comienzo del uso de la telefonía móvil se remonta a la década de 1940 en Estados Unidos, cuando se empezaba a ver el uso que podría tener en el ámbito civil. Comenzaron usando un sistema de radio analógico codificado en amplitud logrando una mejora tiempo después con la modulación en frecuencia.

Ilustración 1: Teléfono móvil años 40

A pesar de lo novedoso del invento, la repuesta de los consumidores no fue la esperada. Debido al peso, tamaño y coste que tenía el aparato (Emmanuel Bertin y Noel Crespi, 2013).

La primera compañía en el mercado de la telefonía fue Bell. Fabricó terminales móviles para vehículos. Estos terminales permanecieron en el mercado hasta 1985 donde fueron expulsados por la tecnología 1G.

Aunque estos terminales móviles nos parezcan enormes y muy incómodos de transportar, en su momento supusieron una enorme revolución en el mundo de las telecomunicaciones.

Ilustración 2: Terminal móvil 1G

Posteriormente, en la década de los 90 tuvo lugar la irrupción del sistema 2G. La mejora fue la inclusión del sistema de comunicación GSM. Fue posible mejorar la calidad de la comunicación así como la seguridad de la llamada. Pero lo más importante fue la simplificación de la tecnología del terminal que lo hacía más económico y permitía la fabricación de terminales de menor tamaño. Todo esto impulsó su comercialización.

Una generación de transición denominada el 2.5G llegó a los pocos años. Sus novedades fueron la incorporación de los SMS y MMS, para el envío de mensajes y archivos multimedia. También se incorporaron las redes EDGE y GPRS que aumentaban la velocidad de transferencia de datos llegando hasta 384 Kbps.

Ilustración 3: Terminales 2G y 2.5G

La tercera generación 3G, proporcionó una velocidad de acceso a Internet que posibilitaba la gestión de contenidos multimedia. La red UMTS proporciona una velocidad de descarga de datos de hasta 7.2Mbps. Por tanto los terminales se fabricaron en consonancia a estas posibilidades. Se les dotó de gran pantalla, potencia de procesamiento y capacidad de almacenamiento.

Ilustración 4: Terminales móviles 3G

La cuarta generación 4G, que acaba de llegar al mercado, sigue mejorando las ventajas por las que ya destacó la generación 3G. En su versión teórica define una velocidad de descarga de 100Mbps en movimiento y de 1Gbps en reposo. Por tanto, esto posibilita el uso de comunicaciones en tiempo real, como videoconferencias, o el uso de *streams* de audio o vídeo.

Ilustración 5: Terminales móviles 4G

2. Los *smartphones*

El término *smartphone* no es más que el nombre comercial con el que se conoce a un teléfono móvil que tiene unas ciertas características. Algunas de ellas son las de tener una mayor capacidad de almacenar datos y ser más potente que un celular convencional. Pese a la limitación de capacidad de almacenamiento de algunos de terminales, conviene mencionar que muchos de ellos poseen una ranura para tarjetas de memoria de modo que su capacidad

de almacenamiento es ampliable. Esto confiere al terminal la capacidad de acceder a recursos y realizar operaciones que con otro dispositivo móvil no podríamos (Ahson, 2012).

Posee elementos de entrada y salida, con los que se comunica con la red, con el usuario, o con otros dispositivos. En la actualidad estos teléfonos tienden a poseer una pantalla táctil (aunque también pueden encontrarse *smartphones* con teclado físico y botones que manejan el terminal), con la que interactuar con el terminal y al menos una cámara de fotos, aunque es ya muy común que incorporen 2 cámaras, una trasera de buena calidad, y otra frontal de menor resolución.

Otras características comunes entre la mayoría de *smartphones* son la función multitarea y el acceso a Internet vía *Wi-Fi,* red 3G o 4G en los dispositivos más avanzados, mediante los cuales los terminales se comunican con la red.

También existe otra funcionalidad que ya poseían los teléfonos móviles antes de la llegada de los *smartphones*, que es el *Bluetooth*, permite la comunicación con dispositivos cercanos para funcionalidades de manos libres o reproducción de música inalámbrica. El más avanzado de estos sistemas es el NFC, un campo de comunicación de alta velocidad a corta distancia, con la que se están implementando la transferencia de archivos o el pago automático.

Actualmente las nuevas funcionalidades de los teléfonos móviles han atraído nuevas necesidades por lo que es común encontrar *smartphones* con GPS o acelerómetros. También admiten el procesado de documentos PDF y Microsoft Office lo que les confiere una gran utilidad en entornos de negocios y educativos.

La finalidad de todas estas utilidades es instalar aplicaciones en el dispositivo que hagan uso de estos componentes y así nos ofrezcan un gran abanico de funcionalidades.

3. Aplicaciones para los *smartphones*

Una aplicación móvil o '*app'* es una aplicación informática diseñada para ser ejecutada en teléfonos inteligentes, tabletas y otros dispositivos móviles. Por lo general se encuentran disponibles a través de plataformas de distribución, operadas por las compañías propietarias de los sistemas operativos móviles como Android, iOS, *BlackBerry* OS, *Windows Phone*, entre otros (Gironés, 2012).

El mundo en el que vivimos hoy en día está constantemente conectado. Conectado a Internet, a otros dispositivos, o entre nosotros. En estos 3 escenarios las aplicaciones móviles actúan ofreciéndonos multitud de servicios y ayudándonos a completar nuestras tareas cotidianas de una forma más sencilla y accesible.

Así, podemos definir una aplicación móvil como "un elemento ejecutable que primeramente descargamos e instalamos en nuestro dispositivo y posteriormente utilizamos para el fin que haya sido creado" (Gironés, 2012).

En este punto es necesario mencionar que todas las aplicaciones no pueden ser utilizadas por todos los dispositivos móviles y que no todas son gratuitas. Cada aplicación móvil es programada para un determinado sistema operativo móvil y con compatibilidad para unas determinadas versiones de ese sistema operativo. Los principales sistemas operativos móviles hoy en día son Android, iOS, *Windows Phone* y *BlackBerry* OS.

Android es un sistema operativo de con un *kernel* Linux y respaldado por Google. Es el más usado en la actualidad, y en España a Diciembre de 2013 es el que corre en el 89% de los dispositivos móviles (Date, 2014).

iOS es un sistema operativo propiedad de Apple y que corre en los dispositivos iPhone y iPad que comercializa la misma marca. En España el 7,6% de los dispositivos móviles corren con iOS a Diciembre de 2013.

Es un sistema operativo móvil desarrollado por Microsoft que principalmente corre sobre dispositivos fabricados por NOKIA. A diciembre de 2013 el 3% de los dispositivos en España corre con *Windows Phone*.

BlackBerryOS es un sistema operativo desarrollado por RIM y que se comercializa en los terminales que *BlackBerry* comercializa. La cuota de mercado que posee en la actualidad es muy baja y sólo el 0,2% de los dispositivos corre con *BlackBerryOS*.

Es interesante tener en cuenta estas cifras ya que el desarrollo de aplicaciones móviles irá de la mano. Ya que uno de los objetivos principales es llegar a la mayor cantidad de público.

En modo en el que cada uno de estos sistemas operativos distribuye las aplicaciones también varía así Android tiene el *PlayStore*, iOS el *iPhone App Store*, *Windows Phone* el *Windows Phone Marketplace* y BlackBerry el *BB App World.*

	Android	**iOS**	**Windows Phone**	**BlackBerry OS**
Desarrollador	Google	Apple Inc.	Microsoft	RIM
Terminales	Principales fabricantes de terminales	Fabricados por Apple*: iPad, iPhone, iTouch*	Principalmente los fabricados por Nokia	Los fabricados por BlackBerry
Proveedor de aplicaciones	*PlayStore*	*App Store*	*Marketplace*	*BB App World*

Símbolo				
Uso (España, Diciembre 2013)	89%	7,6%	3%	0,2%

Tabla 1: Tabla simplificada de sistemas operativos móviles

Fuente: (Date, 2014)

4. Sistemas operativos móviles

1. Android

1. Historia

Google adquiere Android Inc. en 2005. Android era una pequeña empresa de desarrollo de aplicaciones móviles que acababa de ser creada. En ese mismo año comienzan a trabajar en la máquina virtual que soportaría el sistema operativo: Dalvik (Cummings, 2013).

En 2007 se crea el consorcio Handset Alliance para desarrollar estándares orientados a terminales móviles. El consorcio lo forman Google, Intel, Texas Instruments, Motorola, T-Mobile, Samsung, Ericsson, Toshiba, Vodafone, NTT DoCoMo, Sprint Nextel y otros. Como objetivo: promover el diseño y difusión de Android.

En 2008 aparece el primer móvil en el que corre Android. En octubre de ese mismo año, Google libera el código fuente de Android y abre el *Android Market*.

A partir de ese momento, Google desarrolla nuevas versiones de Android y reemplaza el *Android Market* por *Google Play Store.* También actualiza sus versiones del SDK.

2. Apariencia

Android posee un escritorio inicial de disposición vertical formado por cuantas ventanas queramos añadir. En este escritorio podemos situar accesos directos a aplicaciones y agrupar estos en carpetas (Nudelman, 2013).

También podemos añadir elementos de información relacionados con una aplicación llamados *widgets*. En la parte inferior del escritorio tenemos cuatro aplicaciones fijas y el acceso al menú, en la parte superior la barra de estado que al deslizar hacia abajo despliega el centro de notificaciones.

Dentro del menú tenemos todas las aplicaciones que estén instaladas en nuestro dispositivo así como aquellas que vienen por defecto en el sistema operativo del dispositivo.

Ilustración 6: Apariencia sistema operativo Android

3. Características

Las características que diferencian a Android de los demás sistemas operativos (Gironés, 2012):

- Es una plataforma abierta. Es de desarrollo libre, basado en Linux.
- Se puede adaptar a cualquier tipo de *hardware*. Actualmente el uso más extendido de Android se centra en móviles y *tablets*, pero el sistema operativo Android puede ser usado en relojes o electrodomésticos.
- Una aplicación diseñada para una determinada versión de Android funciona en todos los dispositivos que corren en esa versión.
- Portabilidad de procesador asegurada. El desarrollo en Java permite la ejecución de aplicaciones en cualquier tipo de procesador.
- Arquitectura inspirada en elementos de Internet. El diseño de la interfaz de usuario se realiza en XML, estándar de Internet. Por tanto, se adaptará al tamaño de la pantalla del dispositivo.
- Optimizado para baja potencia y poca memoria. Debido al uso de una máquina virtual Dalvik.

4. Arquitectura

La arquitectura del sistema operativo Android se define como describe la siguiente figura (Google 2014):

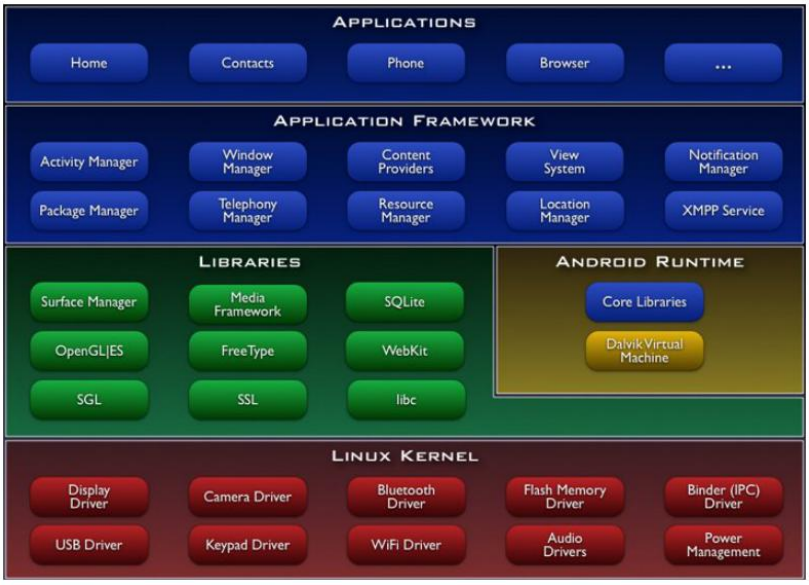

Ilustración 7: Arquitectura sistema operativo móvil Android

El núcleo Android está formado por el sistema operativo Linux, versión 2.6. Se encarga de la seguridad, manejo de memoria, el multiproceso, la pila de protocolos y los *drivers*. Esta capa abstrae el *hardware* del *software,* es decir, es la única capa dependiente del *hardware*.

El *runtime* de Android está basado en el concepto de máquina virtual utilizado en Java. Dada las limitaciones de los dispositivos en los que correría Android, la máquina virtual de Java era inviable, por lo que Google decidió diseñar la suya propia: Dalvik. Para convivir con estas limitaciones usa ficheros Dalvik ejecutables .dex optimizados para ahorrar memoria y está basado en un sistema de memoria de registros. La aplicación se ejecuta en su propio proceso Linux con su propia instancia de máquina virtual. El *runtime* también incluye las *core libraries*, la mayoría de las librerías del lenguaje Java.

2. iOS (Apple, 2015)

Es un sistema operativo creado por *Apple Inc.* inicialmente fue creado para el iPhone, pero con el tiempo fue adaptado para los demás dispositivos móviles como iPad y *iPad touch*. Es una variante del núcleo Darwin que se encuentra en Mac OS. Este sistema operativo móvil está basado en el concepto de manipulación directa. Dispone de un menú general en el que se sitúan todas las aplicaciones, que podemos agrupar en carpetas, en su parte inferior un grupo fijo de aplicaciones o carpetas llamada *Dock* y en la parte superior la barra de estado.

Ilustración 8: Apariencia del sistema operativo iOS

1. Historia

Por el año 2005 Steve Jobs creía que el teléfono móvil iba a ser el dispositivo más útil para el acceso de información. Por eso en aquel entonces, Apple decidió dejar de evolucionar su *NewtonPDA* y centrarse en el nuevo *iTunes* e *iPod* (Morrissey, 2010).

Apple reveló la existencia de un sistema operativo orientado a móviles en Enero de 2007, meses después fue presentado el primer *iPhone*. Aunque entonces se conocía al sistema operativo como OS X, y no sería hasta 2009 cuando se le bautizase como iOS.

Apple acostumbra a presentar una actualización de su sistema operativo cada año acompañado de la salida al mercado de un nuevo terminal móvil:

Modelo	Salida a la venta	SO	Relevancia
iPhone*, posteriormente llamado *iPhone 2G	Julio 2007	iOS 1, llamado *iPhone* OS	Sin soporte para aplicaciones nativas o de terceros

iPhone 3G	Julio 2008	iOS 2	Se crea la *AppStore*, y con ello permiten aplicaciones de terceros
iPhone 3GS	Junio 2009	iOS 3	Sale a la venta el primer *iPad*
iPhone 4	Junio 2010	iOS4	Aparece la multitarea, un *GamingCenter* y grabado de vídeos en HD. También es usado por el *iPad 2*.
iPhone 4S	Diciembre 2011	iOS 5	Incluye el asistente de voz Siri y *iCloud*. También es usado por el *iPad 3*.
iPhone 5	Septiembre 2012	iOS 6	Mapas de fuente Apple con navegación. También es usado por el *iPad 4* y Mini.
iPhone 5S y 5C	Octubre 2013	iOS 7	Identificación de huella dactilar. También es usado por el *iPad Air* y Mini 2.
iPhone 6 y 6 Plus	Septiembre 2014	iOS 8	Incluye llamada *WiFi,* aplicaciones de salud, *iCloud Drive* y *ApplePlay*.

Tabla 2: Evolución histórica de iOS

2. Arquitectura

La arquitectura del sistema operativo iOS viene representada por esta figura (Apple, 2015):

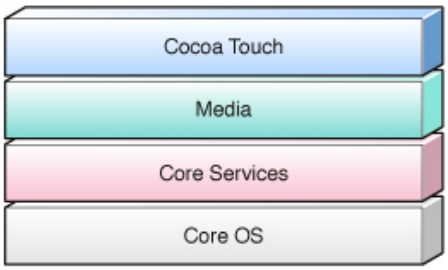

Ilustración 9: Arquitectura del sistema operativo móvil iOS

- *Cocoa Touch*: Es la capa más importante para el desarrollo de aplicaciones iOS. Posee un conjunto de *frameworks* que proporciona el API de *Cocoa* para desarrollar aplicaciones. *Cocoa Touch* proviene de *Cocoa*, la API ya existente en la plataforma MAC. Esta capa está formada por varios *frameworks*:

- UIKit: contiene todas las clases que se necesitan para el desarrollo de una interfaz de usuario. Permite crear interfaces gráficas y gestionar eventos. Incluye también acceso a los sensores del dispositivo.
- *Foundation Framework*: define las clases básicas, acceso y manejo de objetos, servicios del sistema operativo.
- Servicio de notificaciones *Apple push*: posibilidad de alertar al usuario de que una aplicación tiene nuevos datos.
- *Address Book UI framework*: permite crear interfaces estándar para acceder a la agenda (leer, editar, seleccionar contactos).
- App e-mail: permite componer y encolar en el buzón de salida mensajes de correo.
- *Map Kit framework*: permite crear interfaces para embeber mapas.
- Soporte *Peer to peer*: permite comunicaciones P2P utilizando el soporte *Bonjour* implementación del marco de trabajo Zeroconf, tecnología que permite descubrir los servicios disponibles en una red local.

- Media: Provee los servicios de gráficos y multimedia a la capa superior.
  - Tecnologías gráficas: funcionalidades avanzadas para interfaces gráficos como: Quartz 2D, *Core animation, OpenGL* ES
  - Tecnologías de audio:
    - reproducción de sonido (AV *Foundation*)
    - soporte nativo para generación, grabación, mezcla y reproducción de audio (*Core Audio framework*), se incluye también acceso a la capacidad de vibración
    - *OpenAL* (API de acceso a dispositivos de audio)
  - Tecnologías de reproducción de vídeo a través de *Media Player framework* y formatos soportados: H.264 y MPEG-4

- *Core Services*: Contiene los servicios fundamentales del sistema que usan todas las aplicaciones.
  - *Address Book*: acceso a los contactos.
  - *Core Data*: permite gestionar modelos de datos de una aplicación que siguen el patrón MVC.
  - *Core Foundation*: interfaces en C para la gestión de datos (*arrays*, *strings*, *date*, *socket*, *threads*, etc...) .

- *Core Location*: permite obtener la localización del dispositivos utilizando el *hardware* disponible (GPS, *cell* ID o señal *WiFi*). Acceso a brújula en la última versión.
- *Foundation framework*: proporciona *wrappers* en *Objective*-C para acceso a la funcionalidad del *Core Foundation*.
- *Store kit framework*: permite gestionar transacciones de pago a través de *iTunes*.
- *SQLite*: soporte a base de datos SQL.
- XML*Support*: soporte a manipulación de ficheros XML.

- *Core* OS: Contiene las características de bajo nivel: ficheros del sistema, manejo de memoria, seguridad, drivers del dispositivo.
  - *CFNetwork*: interfaces para trabajar con protocolos de red (BSD *sockets*, SSL o TLS, DNS, HTTP, HTTPS, FTP, *Bonjour*).
  - *Accessory Support*: gestión de dispositivos externos conectados al dispositivo por Bluetooth o por el conector.
  - *Security*: gestión de certificados, claves públicas y privadas, políticas de confianza, cifrado simétrico, generación de números aleatorios.
  - *System*: interfaces de acceso a bajo nivel a la funcionalidad del *kernel* (*drivers*, acceso a memoria, sistema de ficheros,...).

3. Windows Phone

1. Introducción

En 1996 Microsoft lanza un sistema operativo de dispositivos limitados Windows CE (*Embedded Compact*). Características (Microsoft, 2014):

- Utilización de una versión reducida de *Windows* (mantiene el interfaz de usuario y aplicaciones *Office*).
- Sistema operativo multitarea.
- Capacidades multimedia.
- Adaptación de múltiples protocolos, conectividad inalámbrica.

2. Apariencia

Windows Phone consta de una única pantalla formada por múltiples ventanas en las que se distribuyen todas las aplicaciones del dispositivo. Esta pantalla es personalizable, podemos adaptar tamaño y color de nuestros iconos. También ofrece la creación de *tyles* que son accesos directos a concretos procesos de las aplicaciones.

Ilustración 10: Apariencia de Windows Phone

3. Arquitectura

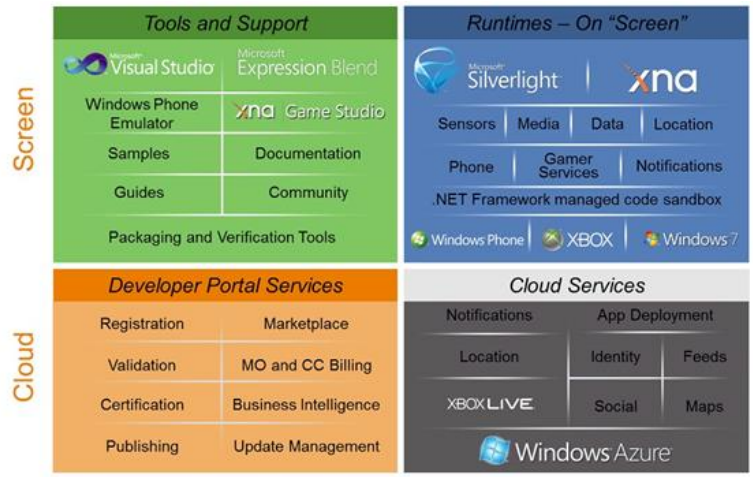

Ilustración 11: Arquitectura del sistema operativo móvil Windows Phone

La arquitectura se divide en cuatro componentes como se aprecia en la figura superior (Microsoft, 2014):

- *Runtime - On Screen*: basado en las plataformas *Silverlight* y XNA. Todo el desarrollo se realiza con código manejado (C#) siguiendo un modelo *sandbox* que permite el fácil desarrollo de aplicaciones seguras. Se desarrolla en dos entornos ya existentes para el mundo *Windows*:
    - *Silverlight*: desarrollo de aplicaciones para Internet.
    - XNA: desarrollo de juegos para plataformas *Windows* (basado en entorno .NET).

  La adaptación para *Windows Phone* de aplicaciones ya desarrolladas en estos entornos son mínimas:
    - Tamaño de la pantalla.
    - Nuevas funcionalidades del teléfono: Sensores y Servicios de localización (relacionado con el componente *cloud*)
- *Tools*: basado en las herramientas *Microsoft Visual Studio* y *Expression Blend*. Todas las herramientas de desarrollo están integradas en *Visual Studio 2010 Express for Windows Phone*:
    - *Visual Studio 2010*: IDE para aplicaciones *Windows Phone*, se pueden desarrollar tanto aplicaciones *Silverlight* como XNA.
    - *Expression Blend:* permite el diseño de interfaces gráficas basadas en XAML (la lógica de estas aplicaciones se desarrollará en *Visual Studio 2010*).
    - *Windows Phone Emulator*: para la depuración y testeo de aplicaciones.
    - XNA *Game Studio*: funcionalidades específicas para juegos.

- *Cloud Services*: basado en *Windows Azure, Xbox Live services*, servicios de notificación, localización y otros servicios web. Facilita la integración de las aplicaciones con servicios web: notificaciones, localización, identidad, redes sociales, servicios de mapas, *feeds*.

- Aplicaciones utilizando la plataforma *Windows Azure* (*Windows cloud*).

- *Portal Services*: gestión de la tienda de aplicaciones para *Windows Phone*. Facilita todos los servicios relacionados con la tienda de aplicaciones: registro y validación, certificación, publicación y gestión de actualizaciones, facturación y lógica del negocio.

4. BlackBerry OS

1. Características

BlackBerry OS es un sistema operativo propietario desarrollado por BlackBerry Ltd. (inicialmente conocida como *Research in Motion Limited* o RIM) para su línea de teléfonos inteligentes (Blackberry, 2014). Proporciona multitarea y soporta dispositivos de entrada especializados: *trackwheel* (o rueda de scroll), *trackball* (rueda que funciona como puntero), *trackpad* y pantalla táctil pensado inicialmente para el usuario profesional soporte de correo corporativo sincronización con las aplicaciones de correo Microsoft Exchange, Lotus Domino o Novell *GroupWise* calendario, tareas, notas, contactos (basado en *BlackBerry Enterprise Server*) última versión: BlackBerry 10, incompatible con las versiones 7.x anteriores (aunque se sigue dando soporte para estas versiones previas).

Programación de aplicaciones para BlackBerry OS 7.1 y anteriores, dos opciones: JavaScript/CSS/HTML o Java que posee mejor integración con el S.O. y con interfaces de usuario más completas.

Programación de aplicaciones para BlackBerry 10.x y anteriores, varias opciones: aplicaciones nativas desarrolladas en C/C++, JavaScript/CSS/HTML5 se pueden usar en BlackBerry 7.x y convertir a iOS y Android o aplicaciones basadas en Adobe AIR que posee las siguientes características (Blackberry, 2014):

- Un entorno de ejecución multisistema operativo permite a los desarrolladores combinar HTML, *JavaScript*, Adobe Flash y Flex, y para desarrollar *Rich Internet Applications* (RIAs)
- Un único instalador de cada aplicación permite instalarla en diferentes dispositivos y S.O. (ordenadores de sobremesa, *netbooks*, *tablets*, *smartphones* y TVs inteligentes)

En el caso de Android se pueden convertir aplicaciones Android a BlackBerry (generando un fichero con el sistema de empaquetamiento propio de BlackBerry) con la limitación de que no se soportan todas las APIs de Android 4.2.2 (*Jelly Bean*).

2. Apariencia

El sistema operativo BlackBerry OS consta de una pantalla principal donde se muestra la fecha, la hora, el calendario y las alarmas del dispositivo. En su parte inferior consta de seis accesos directos programables. Accediendo al menú se muestran las carpetas de las que consta el sistema, como las de multimedia, aplicaciones, juegos y la configuración. Dentro de cada una están los accesos directos a nuestras aplicaciones (BlackBerry, 2015).

Ilustración 12: Apariencia del sistema operativo BlackBerry OS

3. Arquitectura

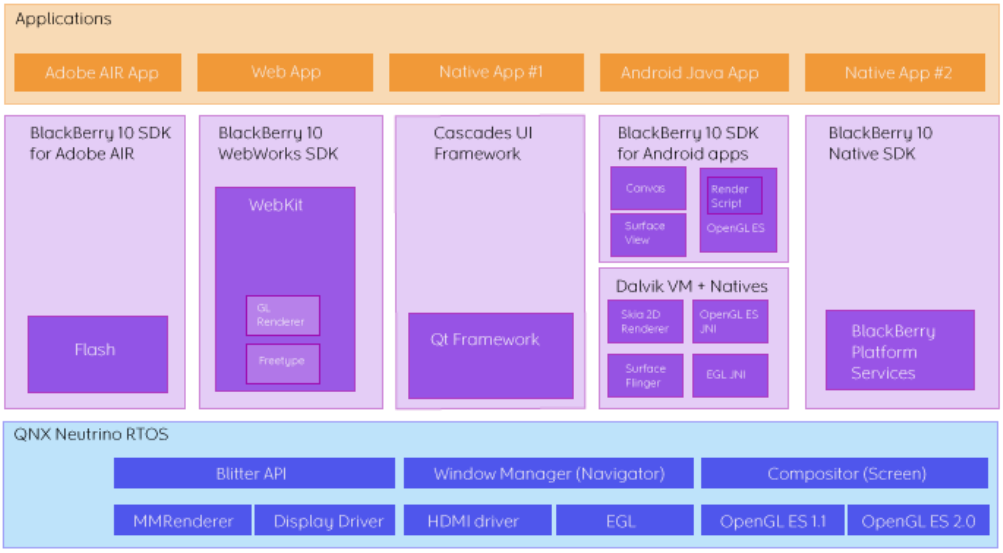

Ilustración 13: Arquitectura del sistema operativo móvil BlackBerry OS

- El núcleo de BlackBerry 10 es el sistema operativo de tiempo real (RTOS) QNX Neutrino que es un sistema de tipo *microkernel*: un *kernel* muy reducido proporciona servicios básicos (incluyendo comunicación entre procesos), resto de servicios del sistema operativo proporcionados por procesos en modo usuario. También incluye un *window compositor* que realiza transformaciones sobre las ventanas y las combina BlackBerry 10.

- *Webworks SDK*: entorno de desarrollo basado en el motor webkit, usa el hardware disponible para producir gráficos 2D y 3D para programación de aplicaciones basadas en tecnologías web (no nativas).

- *Cascades UI Framework*: basado en el entorno Qt para el desarrollo de interfaces gráficas en C/C++ (aplicaciones nativas, acceso a otras APIs de funciones no gráficas).

- *BlackBerry Runtime for Android applications*: para la ejecución de aplicaciones Android (previamente rempaquetadas) incluye una máquina virtual Dalvik para la ejecución de código Android.

- BlackBerry 10 SDK for Adobe AIR: para aplicaciones desarrolladas usando las tecnologías incluidas en Adobe AIR.

5. Comparativa entre los sistemas operativos móviles

	iOS	**Android**	**Windows Phone**	**BlackBerry OS**
Compañía	Apple	Open Handset Alliance	Windows	RIM
Núcleo del SO	Mac OS X	Linux	Windows NT	Mobile OS
Lenguaje de programación	Swift, Objetive-C, C++	Java, C++	C#, muchos	Java
Familia CPU soportada	ARM	ARM, MIPS, Power, x86	ARM	ARM
Licencia de *software*	Propietaria	*Software* libre y abierto	Propietaria	Propietaria
Año de lanzamiento	2007	2008	2010	2003
Motor de navegador *Web*	WebKit	WebKit	Pocket Internet Explorer	WebKit
Soporte *Flash*	No	Sí	No	Sí
Número de aplicaciones	1.300.000	1.400.000	350.000	500.000
Coste de publicar	$99 / año	$25 una vez	$99 / año	Sin coste
Plataforma de desarrollo	Mac	Windows, Mac, Linux	Windows	Windows, Mac
Actualizaciones automáticas del SO	Sí	Depende del fabricante	Depende del fabricante	Sí
Soporte de memoria externa	No	Sí	Sí	Sí
Fabricante único	Sí	No	No	Sí
Variedad de dispositivos	Modelo único	Muy alta	Media	Baja

Tipo de pantalla	Capacitiva	Capacitiva, resistiva	Capacitiva	Capacitiva, resistiva
Aplicaciones nativas	Sí	Sí	Sí	No

Tabla 3: Comparativa sistemas operativos móviles

(Apple, 2015) (Blackberry, 2014) (Google, Android developer, 2014) (Microsoft, 2014).

6. Conclusión

Dadas las características de los sistemas operativos explicados se ha elegido Android para implementar la aplicación *eLiza*. Debido a que es el sistema operativo más extendido. Por razones como que la mayoría de los fabricantes comercializan sus terminales con este sistema operativo. Se deduce de este hecho que el rango de precios en los terminales es amplio por lo que implementando una aplicación en Android se llega a la mayoría de la gente. No sería de este modo si eligiésemos iOS, sabiendo que los terminales que comercializa *Apple* tiene un precio muy elevado, y de este modo no se llegaría a tanto público.

5. Android

1. Librerías nativas

Android incluye un conjunto de librerías en C/C++ llamadas librerías nativas. Compiladas en el código nativo del procesador. Algunas de ellas son (Cummings, 2013):

- *System C library*: derivación de la librería BSD de C, adaptada a Linux.
- *Media Framework*: Soporta códecs de reproducción y grabación en multitud de formatos audio/vídeo (Gironés, 2012).
- *Surface Manager*: Maneja el acceso al sistema de representación gráfica 2D, 3D.
- *WebKit*: Soporta un moderno navegador *Web*. La misma librería de *Google Chrome* y *Safari.*
- SGL: Motor de gráficos 2D.
- Librerías 3D: Implementación basada en *OpenGL*. Usado en el acelerador *hardware* 3D.
- *FreeType*: Fuentes en *BitMap* y renderizado virtual.
- *SQLite*: Motor de bases de datos relacionales.
- *SSL*: Proporciona servicios de encriptación *Secure Socket Layer*.

2. Versiones

1. Android 1.0 Nivel de API 1(septiembre 2008, Apple Pie)

La primera versión de Android, nunca llegó a utilizarse de manera comercial, de ahí que carezca de sentido hacer un desarrollo de la misma. Características desde las que parten las posteriores versiones de Android (Cummings, 2013):

- Incluía el entonces *Android Market*: mercado para la descarga y actualización de aplicaciones.
- Navegador *Web* para visualizar páginas *Webs*.
- Cámara: aunque no era configurable a nivel de resolución tonos etc.
- Posibilidad de creación de carpetas que permiten la agrupación de aplicaciones dentro de una carpeta en la pantalla de inicio.
- Acceso a servidores de correo electrónico. Sincronización de *Gmail* con la aplicación de *Gmail*.

Ilustración 14: Android 1.0

- Soporta sincronización de *Google Contacts*, *Calendar*, *Maps* con *Latitude*, *Street View*, *Google Sync*, *Google Search*.
- Mensajería instantánea *Google Talk* (actualmente *Hangouts*), mensajes de texto y MMS.
- Reproductor de medios, administración, importación, y reproducción de archivos multimedia.
- Las notificaciones aparecen en la barra de estado, con opciones de aviso configurables en términos de LED, tono o vibración.
- Otras aplicaciones internas del dispositivo como: Alarma, Calculadora, Marcación (teléfono), Pantalla de inicio *(launcher*), Imágenes (Galería) y ajustes.
- Soporte para *Wi-Fi* y *Bluetooth*.

2. *Android 1.1 Nivel de API 2 (febrero 2009, Banana Bread)*

Apenas añadió funcionalidades. Si se quiere desarrollar una aplicación compatible a todos los dispositivos Android, está sería la opción a escoger. Añadió la posibilidad de guardar en el sistema de almacenamiento los archivos adjuntos de mensajes (Cummings, 2013).

Ilustración 15: Android 1.1

Se extiende por defecto el tamaño de la pantalla cuando una llamada está en curso. Además de ofrecer la posibilidad de esconder o mostrar el marcador.

3. *Android 1.5 Nivel de API 3 (abril 2009, Cupcake)*

Se trata de la primera versión con números significativos de usuarios (6% a principios del 2011). Incluye el soporte de teclados virtuales personalizados por terceros, la predicción de palabras en la escritura con el diccionario, la inclusión de palabras en el diccionario predictivo, así como la capacidad de grabación de audio y vídeo en los formatos MPEG-4 y 3GP (Cummings, 2013).

Ilustración 16: Android 1.5

Aparecen los *widgets* de escritorio y *live folders*. Incorpora soporte *bluetooth* estéreo, por lo que permite conectarse automáticamente a auriculares bluetooth. Las transacciones entre ventanas se realizan mediante animaciones, lo que confiere al terminal un funcionamiento más visual y atractivo al usuario.

Se añadió al navegador la característica de selección de texto de las páginas *Web*, y la posibilidad de copiar, cortar y pegar este texto.

Mediante un sensor de posición del móvil, esta versión implementa la vista vertical u horizontal, también llamada *portrait*. De este modo, al girar el móvil la pantalla se girará (si la aplicación está adaptada).

4. *Android 1.6 Nivel de API 4 (septiembre 2009, Donut)*

Permite capacidades de búsqueda avanzada en todo el dispositivo. Incorpora *gestures* y *multitouch* gracias a una herramienta de desarrollo *GestureBuilder*. Permite la síntesis de texto a voz: *text-to-speech* (Cummings, 2013).

Ilustración 17: Android 1.6

También se facilita que una aplicación pueda trabajar con diferentes densidades de pantalla. Soporte para la resolución de pantallas WVGA. Aparece un nuevo atributo XML, *onClick*, que puede efectuarse en una vista.

Mejora del *Android Market* en el que los desarrolladores de aplicaciones pueden incluir sus contenidos en los resultados de búsqueda y para los usuarios se facilitan capturas de la aplicación para visualizar antes de instalar la aplicación.

Soporte para CDMA/EVDO, 802.1x y VPNs orientado a la conexión a redes. Por último, incluye mejoras en la aplicación de la cámara.

5. *Android 2.0 Nivel de API 5 (octubre 2009, Éclair)*

Esta versión de API no cuenta casi con usuarios, dado que la mayoría de los fabricantes pasaron directamente de la versión 1.6 a la 2.1 (Cummings, 2013).

Ilustración 18: Android 2.0

Dentro de las novedades que ofrece esta versión destaca la incorporación de un API para el manejo del *bluetooth* 2.1. Nueva funcionalidad que permite sincronizar adaptadores para conectarlo a cualquier dispositivo.

Se incluye la optimización de la velocidad de respuesta del *hardware* y GUI renovada. Ofrece un servicio centralizado de manejo de cuentas. Mejora la gestión de contactos. Una vez más se aumenta el número de tamaños de ventana y resoluciones soportadas.

También renueva la interfaz del navegador y soporte para HTML5. Mejoras en el calendario y soporte para *Microsoft Exchange*. La clase *Motion-event* ahora soporta eventos en pantallas multi-táctil.

La aplicación de la cámara se mejora, incluyendo la configuración manual de *flash*, *zoom* digital, balance de blanco, efectos y enfoque macro.

6. *Android 2.1 Nivel de API 7 (enero 2010, Éclair)*

Es considerada una actualización menor, una serie de mejoras en la versión, de ahí que siga siendo la versión Éclair (Cummings, 2013).

En el paquete *WebKit* se incluyen nuevos métodos para manipular bases de datos almacenadas en la *Web*. También se permite obtener permisos de geolocalización, y modificarlos en *WebView*. Se incorporan mecanismos para administrar la configuración de la caché de aplicaciones, almacenamiento Web, y modificar la resolución de pantalla.

La galería de fotos también vio una importante remodelación en 3D con la ayuda de *CoolIris* que logro una de las aplicaciones visualmente más atractivas integradas para el sistema operativo hasta la fecha.

7. Android 2.2 Nivel de API 8 (mayo 2010, Froyo)

Como característica más destacada se puede indicar la mejora de velocidad de ejecución de las aplicaciones (ejecución del código de la CPU de 2 a 5 veces más rápido que en la versión 2.1 de acuerdo a varios *benchmarks*). Esto se consigue con la introducción de un nuevo compilador JIT de la máquina *Dalvik* y al motor de Java V8. El navegador Web también sufre mejoras, como el soporte de *Adobe Flash* 10.1 utilizado en Chrome o la incorporación del campo "subir fichero" en un formulario (Cummings, 2013).

Ilustración 19: Android 2.2

El desarrollo de aplicaciones permite las siguientes novedades: se puede preguntar al usuario si desea instalar una aplicación en un medio de almacenamiento externo (como una tarjeta SD) o si desea mover a la tarjeta una aplicación instalada en la memoria del dispositivo. Las aplicaciones se actualizan de forma automática cuando aparece una versión. Proporciona servicio para la copia de seguridad de datos que se puede realizar desde la propia aplicación para garantizar al usuario el mantenimiento de sus datos. Por último, se facilita que las aplicaciones interaccionen con el reconocimiento de voz que terceras partes proporcionen nuevos motores de reconocimiento.

Se mejora la conectividad: ahora es posible utilizar nuestro teléfono para dar acceso a Internet a otros dispositivos el llamado *tehering* que es dar conexión *Wi-Fi* a otros dispositivos mediante nuestra conexión 3G, tanto por USB como por *Wi-Fi*.

Se añaden varias mejoras en diferentes componentes: En la API gráfica *OpenGL ES* se pasa a soportar la versión 2.0. También se puede realizar fotos o vídeos en cualquier orientación y configurar otros ajustes de la cámara.

8. *Android 2.3 Nivel de API 9 (noviembre 2010, Gingerbread)*

Ilustración 20: Android 2.3

Debido al éxito de Android en las nuevas *tablets*, ahora soporta mayores tamaños de pantalla y resoluciones (WXGA y superiores). Incorpora un nuevo interfaz de usuario con un diseño actualizado

Se incluye soporte nativo para varias cámaras, pensando en la segunda cámara usada en videoconferencia. La incorporación de esta segunda cámara ha proporcionado la inclusión de reconocimiento facial para identificar el usuario del terminal (Cummings, 2013).

La máquina virtual *Dalvik* de Android introduce un nuevo recolector de basura que minimiza las pausas de aplicación, ayudando a garantizar una mejor animación y el aumento de la capacidad de respuesta en juegos y aplicaciones similares. Se trata de corregir así una de las lacras de este sistema operativo móvil, que en versiones previas no ha sido capaz de cerrar bien las aplicaciones en desuso. Se dispone de mayor apoyo para el desarrollo de código nativo (NDK). También se mejora la gestión de energía y control de aplicaciones. Y se cambia el sistema de ficheros, que pasa de YAFFS a ext4.

Entre otras novedades, conviene destacar que el soporte nativo para telefonía sobre Internet VoIP/SIP. El soporte para la tecnología NFC. Las facilidades en el audio, gráficos y entradas para los desarrolladores de juegos. El soporte nativo para más sensores (como giroscopios y barómetros). Un gestor de descargas para las descargas largas.

9. *Android 2.3.3 Nivel de API 10 (febrero 2011, Gingerbread)*

Al igual que en el caso de *Éclair*, Android 2.2.3 es una actualización de la versión *Gingerbread* 2.3.1 que agrega soporte a tecnologías NFC, además del Nivel de API 10, incluye también otra serie de pequeñas mejoras: nueva versión mejorada de *Swype*, mayor rendimiento de batería, mejora en la fluidez general y pequeños cambios estéticos (Cummings, 2013).

10. *Android 3.0 Nivel de API 11 (febrero 2011, Honeycomb)*

Para mejorar la experiencia de Android en las nuevas *tablets* se lanza la versión 3.0 optimizada para dispositivos con pantallas grandes. La nueva interfaz de usuario ha sido completamente rediseñada con paradigmas nuevos para la interacción, navegación y personalización. La nueva interfaz se pone a disposición de todas las aplicaciones, incluso las construidas para versiones anteriores de la plataforma. Las principales novedades de este SDK son:

Ilustración 21: Android 3.0

- Para poder adaptar las interfaces de usuarios a pantallas más grandes se incorporan las siguientes prestaciones: resolución por defecto WXGA (1280x800), escritorio 3D con widgets rediseñados, nuevos componentes y vistas, notificaciones mejoradas, arrastrar y soltar, nuevo cortar y pegar, barra de acciones para que las aplicaciones

dispongan de un menú contextual siempre presente y otras características para aprovechar las pantallas más grandes.

- Se mejora la reproducción de animaciones 2D/3D gracias al renderizador *OpenGL* acelerado por hardware. El nuevo motor de gráficos *Rederscript* saca un gran rendimiento de los gráficos en Android e incorpora su propia API.
- Consiste en la primera versión de la plataforma que soporta procesadores multi-núcleo. La máquina virtual *Dalvik* ha sido optimizada para permitir multi-procesado, lo que permite una ejecución más rápida de las aplicaciones, incluso aquellas que son de hilo único.
- Se incorpora varias mejoras multimedia, como listas de reproducción M·U a través de HTTP *LiveScreaming*, soporte a la protección de derechos musicales (DRM) y soporte para la transferencia de archivos multimedia a través de USB con protocolos MTP y PTP.
- En esta versión se añaden nuevas alternativas de conectividad, como las nuevas APIS de *Bluetooth* A2DP y HSP con *streaming* de audio.
- También, se permite conectar teclados completos por USB o *Bluetooth*. El uso de los dispositivos en un entorno empresarial es mejorado. Entre las novedades introducidas, se destacan las nuevas políticas administrativas con encriptación del almacenamiento, caducidad de contraseña y mejoras para administrar los dispositivos de empresa de forma eficaz.

Por último, hay que decir, que a pesar de la nueva interfaz gráfica optimizada para *tablets*, Android 3.0 es compatible con las aplicaciones creadas para versiones anteriores. La tecla de menú, inexistente en las nuevas *tablets*, es reemplazada por un menú que aparece en la barra acción.

11. Android 4.0 Nivel de API 14 (octubre 2011, Ice Cream Sandwich)

Android 4.0 ofrece más prestaciones y una mayor flexibilidad que las versiones anteriores. Las principales ventajas que ofrece esta versión son (Cummings, 2013):

- Interfaz de usuario mucho más flexible y con prestaciones. Permite funciones multitarea más avanzadas, que permite el desplazamiento entre tareas, visualizar aplicaciones recientes, gestión de mensajes entrantes… Incorpora un diseño integrado consiguiendo así eliminar los botones de la carcasa. Incluye *widgets* redimensionables y acceso directo a determinadas tareas y notificaciones sin necesidad de desbloquear el teléfono.

Ilustración 22: Android 4.0

- Se mejora la velocidad a la hora de eliminar notificaciones, tareas y pantallas emergentes del navegador, así como la entrada de texto. Permite un control de consumo que puede adaptarse a los usuarios que tienen planes con consumo límite de descarga. Otra característica es que incorpora herramientas de accesibilidad más avanzadas.

- En cuanto a la comunicación, Android 4.0 está diseñado pensando en el ritmo de vida de los usuarios y en la manera de compartir experiencias, de que incluya una nueva aplicación People, que permite conectarse a las redes sociales integradas en un formato más cómodo. Se crea un calendario unificado que engloba todas las tareas de cualquier índole en un mismo lugar. Se mejoran las funciones de la cámara, como la detección de rostro. Es posible gestionar, mostrar y compartir todos los vídeos y fotos con la nueva aplicación de galería.
- Manejo más sofisticado de los videos, así como facilidades para realizar capturas de pantalla.
- En esta versión se añaden nuevas funciones de navegación y correo electrónico, permite acceder a más prestaciones y comunicarse con la misma facilidad.
- Por último, Android 4.0 amplía los límites de la comunicación e incluye sorprendentes y sofisticadas funciones. Prueba de ello es, por ejemplo, la posibilidad del desbloqueo del terminal a través del reconocimiento facial, Android *Beam* con tecnología sin cables NFC, que permite intercambiar de forma instantánea cualquier tipo de aplicación e información entre dos terminales que dispongan de la tecnología NFC o *Wi-Fi Direct* y *Bluetooth* con tecnología HDP, que permite a los usuarios conectarse directamente con dispositivos parecidos a través de una red *Wi-Fi* para establecer una comunicación más fiable y rápida. De esta manera, se elimina la necesidad de tener que realizar una conexión a Internet o usar el terminal como punto de acceso a la red Wi-Fi.

12. Android 4.1 Nivel de API 15(junio 2012, Jelly Beam)

Su objetivo es la mejora de la estabilidad y la funcionalidad de la interfaz de usuario. Para ello, se implementó el núcleo de Linux. También se llevaron a cabo una serie de mejoras en lo que se denominó *Project Butter*, que permitió aumentar hasta en 60 FPS las transiciones de la interfaz de usuario, dando mayor sensación de fluidez (Cummings, 2013).

Ilustración 23: Android 4.1

Otra de las mejoras se centró en la barra de notificaciones, que ofrece mayor integración de acciones. Los *widgets* en esta versión al ser añadidos se ajustan al tamaño disponible en pantalla en el caso de no caber en ella.

El teclado predictivo no sólo nos sugiere la palabra que estamos intentando escribir si no que antes de escribirla nos muestra las sugerencias más probables de cuál es la palabra que queremos escribir, basándose en el contexto.

También la entrada de texto por voz sufrió un gran cambio. Y es que ahora no necesitamos tener una conexión a Internet para poder hacer uso de esta funcionalidad, porque ahora está integrada en el dispositivo.

13. Android 4.2 Nivel de API 16 (septiembre 2012, Jelly Beam)

Entre sus principales actualizaciones de 4.1 se encuentra el *Photo Sphere* o lo que es lo mismo, la captura de imágenes panorámicas con la cámara en el plano horizontal y vertical.

Una de las más útiles actualizaciones fue algo similar a *Swype*, hoy lo conocemos como *Gesture Typing*. Y es la posibilidad de escribir sobre el teclado dibujando trazos sin levantar el dedo de la pantalla. De este modo se agiliza y facilita la escritura por teclado.

La barra de notificaciones siguió su evolución iniciada en versiones anteriores. En este punto Google incorporó una cuadrícula dividida en secciones para acceder a las configuraciones de pantalla, conectividad, sonido, rotación… a esto lo llamó *Quick Settings*.

Como última novedad se incluyó la posibilidad de añadir *widgets* en la pantalla de bloqueo y acceso rápido para acceder directamente a la cámara (Cummings, 2013).

14. Android 4.3 Nivel de API 17 (julio 2013, Jelly Beam)

Incluye mejoras en la representación de las formas redondeadas y el texto, la velocidad con la que se muestran las imágenes así como el soporte para *Open GL ES* 3.0. La aceleración del *hardware* 2D optimiza el flujo de comandos de dibujo convirtiéndolo en un formato GPU más eficiente y reorganizado (Cummings, 2013).

También ha añadido soporte para perfiles restringidos: permite crear ambientes separados para cada usuario en el mismo dispositivo.

El sistema de *bluetooth* puede recibir en el dispositivo conectado notificaciones de las aplicaciones del terminal móvil y mostrarlas.

15. Android 4.4 Nivel de API 18 (octubre 2013, KitKat)

Se optimiza el rendimiento del dispositivo en inactividad y otras especificaciones técnicas como la RAM. Mejora el rendimiento del sistema con la sincronización debido a fallos detectados en 4.3 (Cummings, 2013).

Ilustración 24: Android 4.4

Todas las aplicaciones alcanzan el volumen máximo permitido, este error se debía a *bugs* que hacían que unas aplicaciones sonaran más altas que otras.

Arreglo de fallos en la seguridad de la depuración USB y en la conexión automática *Wi-Fi*. Mejora el *runtime* ART para que funcione correctamente con más aplicaciones.

16. Android 5.0 Nivel de API 21 (noviembre 2014, Lollipop)

Incorpora *Material Design,* un diseño rápido y colorido. Con transiciones visuales muy naturales que hacen la navegación en el dispositivo intuitiva y atractiva al usuario. Nueva tipografía y gama de colores respecto a *KitKat* (Google, Android developer, 2014).

Manejo de notificaciones modificado. Incorpora nuevos métodos de control de notificaciones para controlar cuándo y cómo se reciben. Respuesta a notificaciones desde la pantalla de bloqueo, y configuración de ocultación de información sensible en notificaciones.

Las llamadas entrantes no interrumpirán nuestra actividad, podemos optar por ignorar la llamada y seguir con nuestras tareas mientras el teléfono suena.

Ilustración 25: Android 5.0

3. Aplicaciones

Las aplicaciones de un sistema operativo Android son el conjunto de aplicaciones instaladas en una máquina virtual Dalvik, las cuales han de ejecutarse sobre dicha máquina virtual (Gironés, 2012).

Normalmente las aplicaciones están escritas en Java, y son desarrolladas usando el Android SDK (*Software Development Toolkit*). Aunque también pueden estar desarrolladas en C/C++ usando el Android NDK (*Native Development Toolkit*).

Las aplicaciones están formados por una serie de componentes que en su conjunto y mediante su interrelación definen el aspecto y funcionamiento de cualquier aplicación Android (Gironés, 2012):

- Actividad *(Activity)*: Se trata de cada una de las "pantallas" de las que consta la aplicación. La ventana del *Activity* llena la pantalla aunque puede ser más pequeña y flotar sobre una actividad anterior. Su función principal es la de presentar al usuario los elementos visuales para que interactúe con ellos. Toda actividad ha de pertenecer a una clase descendiente de *Activity*. Toda aplicación está formadas de múltiples actividades débilmente unidas entre sí (Google, Android developer, 2014).
  La actividad principal es aquella que arranca cuando la aplicación es ejecutada, es decir, la primera pantalla de la aplicación. Cada vez que se inicia una nueva actividad, la actividad anterior se detiene, pero el sistema conserva la actividad en el *stack*. Cuando el usuario pulsa la tecla *back* el sistema ejecuta la actividad justo anterior. De este modo, podemos deducir que se trata de una pila LIFO (*Last In First Out*), la última actividad que entró será la primera en salir, en caso de necesitar recuperar una actividad.
  Cuando se detiene una actividad, esta es notificada de este cambio mediante los métodos de retrollamada del ciclo de vida de la actividad. Según al estado al que evolucione una actividad un método u otro pueden ser ejecutados. Estos métodos son usados principalmente para guardar datos de la actividad, por si esta volviese a ser

ejecutada, mantener la información que tenía sin perder nada. El ciclo de vida de una actividad y los métodos accesibles en cada transición son los siguientes:

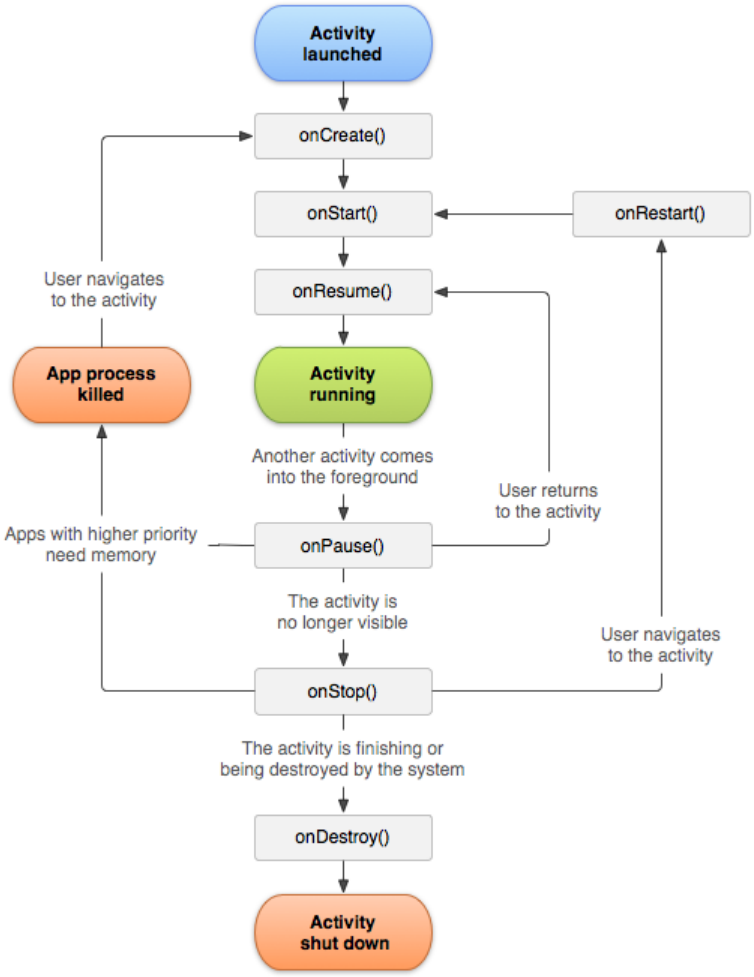

Ilustración 26: Ciclo de vida de una actividad en Android

En la ilustración vemos como una actividad pausada puede volver a ser llevada al frente tras llamar al método *onResume*, cuyo objetivo principal será restablecer la información de la actividad a partir de los datos guardados en *onPause* cuando ésta se pausó.

También vemos como una actividad parada puede volver a ser creada o ejecutada mediante los métodos *onCreate* y *onRestart* respectivamente.

Todas las actividades han de estar declaradas en un manifiesto llamado *AndroidManifest.xml* aquí tenemos un elemento padre llamado *application* que poseerá varios elementos hijos llamados *activities*. En este manifiesto definimos los permisos de la aplicación, y los permisos de acceso de otras aplicaciones a ésta. También el nombre de las actividades y si son actividades principales entre otras cosas.

- Servicio *(Service)*: Un servicio es un componente de la aplicación que puede realizar operaciones de larga ejecución en segundo plano y no proporciona una interfaz de usuario. La aplicación seguirá su curso normal, componentes y actividades se crearán y

destruirán y este servicio seguirá su funcionamiento. Existen dos tipos de servicios (Google, Android developer, 2014):

- Arrancado *Started*: El servicio es arrancado por un componente de la aplicación mediante el método *startService*. Este servicio seguirá ejecutándose, si es necesario, incluso si el componente que lo arrancó es destruido. Por lo general este tipo de servicios no devuelven ningún valor. Simplemente realizan su tarea y al finalizar se destruyen (Gironés, 2012).
- Asociado *Bound*: El servicio es enlazado cuando un componente de la aplicación se une a él llamando al método *bindService*. Este tipo de servicios ofrecen una interfaz mediante la cual los componentes interactúan con ellos. Envían solicitudes para obtener resultados. Una vez es creado este servicio múltiples componentes pueden enlazarse a él, aunque este servicio se destruirá cuando todos ellos se desconecten (Gironés, 2012).

El mismo servicio puede ser usado de las dos formas a la vez. Si la definición del servicio implementa el método de retrollamada *onStartCommand*, este servicio podría ser arrancado por otro componente. O si la definición del servicio implementa el método *onBind* el servicio puede ser asociado a un componente.

Cualquier servicio se ejecuta en el hilo principal del proceso que lo alberga, excepto si el servicio realiza operaciones muy pesadas. En ese caso se debe crear un nuevo hilo que contenga este servicio y así evitar la ralentización de la aplicación y la posibilidad de la aplicación se cuelgue.

La vida completa de un servicio transcurre entre el momento en el que este llama a *onCreate* donde se realizan las tareas iniciales y cuando retorna y llama a *onDestroy* donde se liberan los recursos en uso. Pero la vida activa de un servicio no se corresponde con la vida completa. De este modo la vida activa comienza con la llamada a *onStartCommand* en el caso de servicios arrancados u *onBind* en el caso de servicios enlazados. Ambos métodos reciben un *Intent* como parámetro. Si el servicio es arrancado la vida activa termina en el mismo momento que la vida completa. Pero si el servicio es asociado la vida activa finaliza con la llamada a *onUnbind*.

Ciertos métodos del ciclo de vida han de ser sobrescritos:

- *onStartCommand*: No es necesario si el servicio sólo será accedido mediante asociaciones. El sistema llama a este método cuando un componente quiere arrancar este servicio. El servicio se ejecutará indefinidamente y es responsabilidad del desarrollador pararlo mediante *stopSelf* o *stopService*.
- *onBind*: El sistema llama a este método cuando un componente quiere asociarse al servicio al llamar a *bindService*. En la implementación de servicios asociados se debe proporcionar al cliente una interfaz de acceso al servicio que devuelve un objeto del tipo *IBinder* como respuesta.

- *onCreate*: El sistema llama a este método cuando el servicio es creado por primera vez en tareas de inicialización y antes de realizar la llamada a *onStarCommand* u *onBind*.
- *onDestroy*: El sistema llama a este método cuando el servicio en cuestión no va a seguir en uso y se desea liberar los recursos de los que este está haciendo uso. Las sentencias de la liberación de recursos deben incluirse en este método.

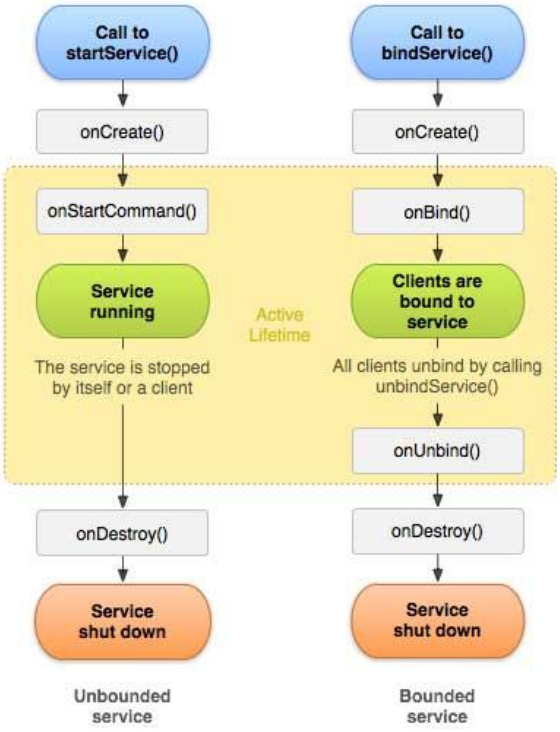

Ilustración 27: Ciclo de vida de un servicio

- Intención *(Intent)*: Es una descripción abstracta de una operación llevada a cabo. Puede ser realizada mediante *startActivity* para lanzar una actividad, mediante *broadcastIntent* para enviarlo a algún componente *BroadcastReceiver* interesado o mediante *bindService(Intent, ServiceConnection, int)* para comunicar con un servicio que se encuentra en *background* (Google, Android developer, 2014).
  Un *Intent* facilita la unión en tiempo de ejecución entre el código de diferentes aplicaciones. Aunque su uso más común es el de lanzar actividades, se le podría definir como el pegamento de las actividades. Existen dos tipos de *Intents*:
  - Explícitos: Se especifica qué clase va a ser ejecutada.
  - Implícitos: No especifican ningún componente. Simplemente incluyen información para que el sistema decida cuál es el mejor componente disponible para ejecutarlo.
- Receptor de anuncios *(BroadcastReceiver)*: Clase base que recibirá los *Intents* enviados por *sendBroadcast*. Lo que hace es recibir anuncios y reaccionar ante la información que éstos contienen. Estos anuncios han podido ser originados por eventos que se producen en el sistema del dispositivo como batería baja o una

llamada entrante. Pero también pueden ser anuncios lanzados por las aplicaciones del propio dispositivo. A pesar de no tener interfaz de usuario propia, pueden reaccionar a los anuncios recibidos lanzando una actividad, si procede (Google, Android developer, 2014).

- Proveedores de contenido *(ContentProvider)*: Es el mecanismo que se usa para que las aplicaciones puedan compartir datos entre ellas sin comprometer la seguridad de sus datos privados. Los datos que una aplicación presenta a las demás está en forma de tablas, con filas y columnas. Siendo la fila un tipo de datos y las columnas un valor dentro del tipo de dato (Google, Developer Android, 2014).
  Un proveedor de contenidos es un objeto de una subclase de *ContentProvider*. La aplicación accede a los datos de un proveedor usando un objeto cliente que pertenece a la clase *ContentResolver* sobre la cual invocamos los métodos de búsqueda o actualización de información.
  Un proveedor de contenidos puede especificar permisos que otras aplicaciones deben de poseer para acceder a los datos del proveedor. Así nos aseguramos que el usuario conoce a qué datos va a intentar acceder la aplicación, estos permisos serán aprobados por el usuario justo antes de la instalación de la misma. Para conseguir los permisos necesarios para acceder a un proveedor de contenidos la aplicación debe incluir un elemento <*uses-permission*> en su *AndroidManifest.xml* (Gironés, 2012).
  El proveedor de contenidos sólo tendrá un uso correcto si se necesita ofrecer datos a otras aplicaciones. No es necesario para acceder a una base de datos interna de la aplicación.

El entorno de la aplicación proporciona una plataforma de desarrollo libre. Esta capa se creó para la simplificación y reutilización de componentes. Una de las mayores fortalezas de Android es que se aprovecha del lenguaje de programación Java, en su mayor parte. Los servicios que incluye son:

- Vista *(Views)*: El conjunto de elementos visuales de la aplicación. Al usar nuestros dispositivos podemos apreciar la presencia de botones, campos de texto, imágenes… etc. Todos estos elementos son objetos de una superclase llamada *View*. Pueden estar declarados y presentados de forma estática o de forma dinámica (Google, Android developer, 2014):
  - Los elementos que son declarados de forma estática se encuentran definidos en un archivo XML, situado en la carpeta res/*layout* en el caso de la vista vertical, o res/*layout-land* en el caso de la vista horizontal o *portrait*. De este modo, todo lo definido en él será inamovible dentro de una vista. Un archivo de vista XML ha de estar definido por un tipo de *layout* que definirá cual será la relación de dependencia entre los elementos contenidos en él:
    - *LinearLayout*: Los elementos se distribuirán de forma linear unos debajo de otros según orden de definición del documento XML. Un

atributo útil en este tipo de *layouts* es el del peso o *weight* mediante el cual damos a cada elemento un valor numérico que ocupará la parte proporcional de pantalla entre su valor y la suma de los valores del resto de elementos (Nudelman, 2013).

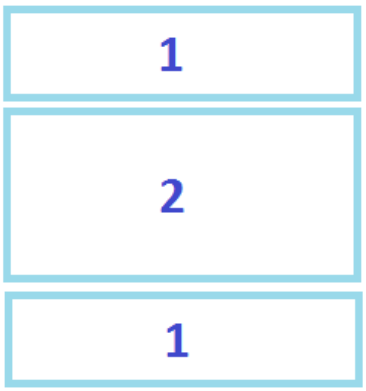

Ilustración 28: LinearLayout

- *RelativeLayout*: Los elementos se distribuirán siguiendo unos patrones de relación posicional que hay que definir entre ellos. Estos patrones se especifican definiendo el atributo *android:layout_...* a cada elemento. Este atributo nos permite posicionarnos arriba, debajo, a la derecha o izquierda, alinear parte superior o inferior… y en cada atributo se define respecto de qué otro elemento se posiciona ya sea un elemento a su mismo nivel, o un elemento padre que lo contiene (Nudelman, 2013).

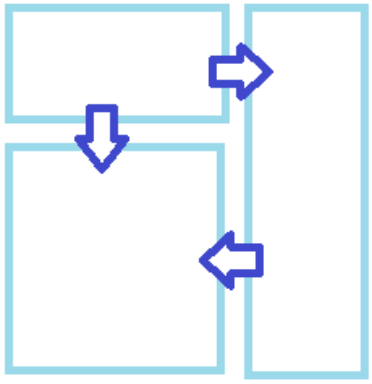

Ilustración 29: RelativeLayout

- *GridLayout*: Los elementos se distribuyen en una cuadrícula de la que se definen las filas y las columnas mediante los atributos del *layout*: *android:rowCount* y *android:columnCount*. Cada elemento que definamos dentro del *layout* tendremos que definir a qué columna y a qué fila pertenece, es decir, en qué cuadrícula estará ubicado (Nudelman, 2013).

Ilustración 30: GridLayout

- *ListView*: Es un conjunto de vistas que se presentan como un único elemento en el archivo XML principal y que posee elementos hijos desplazables en la pantalla. Estos elementos pueden ser definidos de forma estática directamente en el documento XML o también se pueden definir el *layout* propio de cada elemento de la lista definido en otro documento XML, y llenar de forma dinámica la lista en el código mediante un adaptador. Esto nos permite crear una lista de elementos en la que cada uno puede tener una disposición visual propia y que es totalmente dinámico (Nudelman, 2013).

Ilustración 31: ListView

- Los elementos que podemos definir dentro de cada uno de estos *layouts* contenedores de elemento son los siguientes (aunque también es posible insertar un *layout* dentro de otro para un fin concreto):
  - Campo de inserción de texto (*EditText)*: Representado por la clase *EditText* que hereda de la superclase *View*. Permite introducir texto para que posteriormente la aplicación obtenga un objeto del *EditText* y pueda hacer uso del texto. Mediante los atributo de definición en XML podemos adaptar el teclado que automáticamente aparece al clicar sobre el *EditText*, para que muestre una @ si es una dirección de correo (*textEmailAddress*) o una / y .com si es una página *Web* (*textUri)* o un teclado numérico para un número de teléfono (*phone)*. También podemos hacer que lo escrito no sea visible por seguridad

como en el caso de que tengamos que introducir una contraseña (*textPassword)* (Google, Android developer, 2014).

Ilustración 32: EditText

- Texto (*TextView)*: Representado por la clase *TextView* que hereda de la superclase *View*. Es un texto que se muestra por pantalla. El contenido del texto puede ser definido desde el fichero XML por texto plano, o también puede hacer referencia a un *string* contenido en *res/values/strings.xml* referenciado mediante su nombre. El texto del *TextView* también puede ser modificado y establecido desde el cogido de la aplicación obteniendo un objeto de tipo *TextView* y usando el método *setText* sobre él (Google, Android developer, 2014).

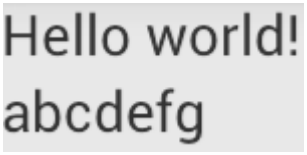

Ilustración 33: TextView

- Botón (*Button)*: Representado por la clase *Button* que hereda de la superclase *View*. Como su nombre indica se trata de un botón. Su principal función suele ser que al pulsarlo la aplicación realice una serie de operaciones programadas. Esta clase define un método de retrollamada para que al pulsar un *Button* este método sea ejecutado. Ha de ser definido desde el mismo fichero XML en el que se encuentra el *Button* y como atributo de este (*onClick*) definiendo el nombre del método que recibirá la llamada que por definición deberá devolver *void* y deberá tener un único argumento de entrada tipo *View* (Google, Android developer, 2014).

Ilustración 34: Button

- Caja de selección múltiple (*CheckBox)*: Elemento representado por la clase *CheckBox* que a su vez hereda de la superclase *View*. Permite la selección única o múltiple de una serie de elementos. Cada opción estará representado por un elemento *CheckBox*. Esta clase define un método de retrollamada para que al pulsar un *CheckBox* este método sea ejecutado, diferencie qué *CheckBox* ha sido marcado o

desmarcado y ejecute el código pertinente en cada caso. Este método puede ser definido desde el mismo fichero XML en el que se encuentra el *CheckBox* y como atributo de este (*onClick*) definiendo el nombre del método que recibirá la llamada que por definición deberá devolver *void* y deberá tener un único argumento de entrada tipo *View* (Google, Android developer, 2014).

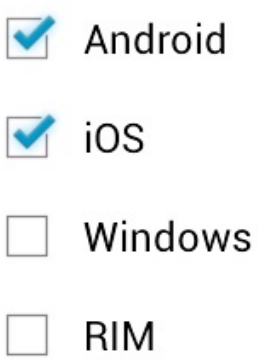

Ilustración 35: CheckBox

- Botones de selección única autoexcluyentes (*RadioButton)*: Cada botón está representado por un objeto de tipo *RadioButton* que hereda de la superclase *View*. Los *RadioButtons* se agrupan de modo que sólo podremos seleccionar un elemento del grupo. Así al seleccionar un elemento se deselecciona el que anteriormente estaba seleccionado si es que había alguno seleccionado (Gironés, 2012). De este modo nos aseguramos que el usuario seguro que seleccionará un elemento y no más de uno, y así se reducen las posibilidades de obtener un valor erróneo. Esta clase define un método de retrollamada para que al pulsar un *RadioButton* este método sea ejecutado, diferencie qué *RadioButton* ha sido marcado y ejecute el código pertinente en cada caso. Que desmarque el *RadioButton* anteriormente seleccionado, si procede. Este método puede ser definido desde el mismo fichero XML en el que se encuentra el *RadioButton* y como atributo de este (*onClick*) definiendo el nombre del método que recibirá la llamada que por definición deberá devolver *void* y deberá tener un único argumento de entrada tipo *View* (Google, Android developer, 2014).

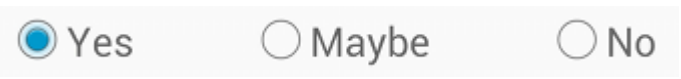

Ilustración 36: RadioButton

- Interruptor biestado (*ToggleButton)*: Cada botón está representado por un objeto del tipo *ToggleButton* que hereda de la superclase *View.* Permite ejecutar código en base a dos estados diferentes en los que se puede encontrar nuestro *ToggleButton*. Esta clase define un método

de retrollamada para que al pulsar un *ToggleButton* este método sea ejecutado, diferencie si el *ToggleButton* ha sido encendido o apagado y ejecute el código pertinente en cada caso. Este método puede ser definido desde el mismo fichero XML en el que se encuentra el *ToggleButton* y como atributo de este (*onClick*) definiendo el nombre del método que recibirá la llamada que por definición deberá devolver *void* y deberá tener un único argumento de entrada tipo *View* (Google, Android developer, 2014).

Ilustración 37: ToggleButton

- Menú de selección única desplegable (*Spinner)*: Un *Spinner* está representado por un objeto de la clase *Spinner* que hereda de la superclase *View*. Se trata de un menú desplegable que nos ayuda a seleccionar un valor de un conjunto de forma inequívoca, ya que los valores válidos a seleccionar ya se le proporcionan al usuario, y no puede seleccionar uno que sea no válido. De este modo se asegura la correcta verificación del formulario. Estos elementos pueden ser definidos de forma estática directamente en el documento XML o también se pueden definir el valor propio de cada elemento de la lista del *Spinner* desde el código, y llenar de forma dinámica la lista en el código mediante un adaptador. Esta clase también puede hacer uso del método *createFromResource* que permite crear un adaptador de tipo matriz a partir de la matriz de cadenas de caracteres. Cuando el usuario selecciona uno de los elementos de la lista, el *listener* ejecuta el método de retrollamada que diferencia qué elemento de la lista se ha seleccionado y ejecuta el código que proceda (Google, Developer Android, 2014).

Ilustración 38: Spinner

- Selector de fecha y/u hora (*Picker)*: Son controles que permiten al usuario seleccionar una fecha y/u hora de modo que no pueda introducir ningún formato incorrecto de ningún valor. Su uso es recomendado dentro de *DialogFragment*. El *DialogFragment* es una

ventana que se muestra por encima de nuestra pantalla y nos informa o nos pide información y en su parte inferior tiene botones normalmente de naturaleza positiva o negativa (*OK* y *Cancel*). Cuando el usuario selecciona uno de los elementos de la lista el *listener* ejecuta el método de retrollamada que procesa la información obtenida del *Picker* y la trata convenientemente (Google, Android developer, 2014).

Ilustración 39: Picker

- *Resource Manager*: Proporciona acceso a los recursos que no son código: animaciones, imágenes, sonidos, vídeos, textos prefijados.
- *Activity Manager*: Maneja el ciclo de vida de las aplicaciones y proporciona un sistema de navegación entre ellas.
- *Notification Manager*: Permite a las aplicaciones mostrar alertas personalizadas en la barra de estado.
- *Content Providers*: Mecanismo sencillo para acceder a datos de otras aplicaciones.

Otro de los elementos importantes en el desarrollo de una aplicación Android, y también importante en la que he desarrollado es el uso de *threads* o hilos:

- Hilos (*Threads*):

Cuando se inicia una aplicación, el sistema crea un hilo de ejecución para la aplicación, llamada hilo principal (*main thread)*. Este hilo es el encargado de enviar los eventos a los elementos de la interfaz de usuario, incluidos los eventos de dibujo. También es el hilo en el que nuestra aplicación se relaciona con los componentes del kit de herramientas de la interfaz de usuario de Android. Es por esta razón por la que el hilo principal es también llamado a veces el hilo de interfaz de usuario (*UI thread, User Interface*) (Goransson, 2014).

Android no crea un subproceso independiente para cada instancia de un componente. Todos los componentes que se ejecutan en el mismo proceso son instanciados en el hilo de interfaz de usuario y las llamadas del sistema para cada componente son realizadas desde ese hilo. En consecuencia, los métodos que responden a las llamadas del sistema son siempre ejecutados desde el hilo de interfaz de usuario del proceso.

Por ejemplo, cuando un usuario toca un botón en la pantalla, el hilo de interfaz de usuario de nuestra aplicación distribuye el evento táctil para el *widget,* éste a su vez establece su estado y mensajes enviando una solicitud con el método *invalidate()* a la cola de eventos. El hilo de interfaz de usuario retira de la cola de eventos la solicitud y notifica el *widget* que debería dibujarse.

Además, el kit de herramientas de la interfaz de usuario de Android no es seguro para realizar subprocesos. Por lo que no debemos manipular la interfaz de usuario de un hilo trabajador, nosotros debemos manipular la interfaz de usuario desde el subproceso de la interfaz de usuario. Por lo tanto, son simplemente dos reglas para el modelo de un solo hilo en Android (Meier, 2009):

- No bloquear el hilo de interfaz de usuario.
- No acceder al kit de herramientas de interfaz de usuario desde fuera del hilo de interfaz del usuario.

Debido al modelo de un solo hilo descrito anteriormente, es de vital importancia no bloquear el hilo de la interfaz de usuario de la aplicación para mantener intacta su capacidad de respuesta. Si tenemos operaciones que realizar que no son instantáneas, debemos asegurarnos de hacerlas en hilos separados llamados hilos de fondo (*background threads*).

El principal problema que se presenta en la gestión de la interfaz de usuarios es que ésta sólo puede ser modificada desde el *UI thread* y es muy común que los eventos desencadenados desde otros hilos en *background* quieran realizar modificaciones en la interfaz. Para solucionar éste problema, Android ofrece varias formas de acceder al hilo de la interfaz de usuario desde otros hilos. Estos son los métodos que nos permiten hacerlo:

- *Activity.runOnUiThread(Runnable)*
- *View.post(Runnable)*
- *View.postDelayed(Runnable, long)*

Un elemento fundamental en el desarrollo de una aplicación Android es la percepción del usuario sobre el funcionamiento. Es conveniente no introducir grandes cargas en el *UI thread* ya que el usuario percibirá el retraso asociado a estas operaciones. Para ello es muy recomendable el uso de *AsyncTask* que nos permite realizar el trabajo asíncrono en nuestra interfaz de usuario. Realiza las operaciones que bloquean nuestro hilo interfaz de usuario en un subproceso de trabajo y publica los resultados en el hilo de interfaz de usuario sin necesidad de manejar hilos.

Para usarlo, debemos usar la subclase *AsyncTask* y aplicar el método de devolución de llamada *doInBackGround(),* este método se ejecuta en un grupo de subprocesos de fondo.

Para actualizar la interfaz de usuario, se debe implementar el método *onPostExecute()* que proporciona el resultado de *doInBackGround()* y éste es ejecutado en el hilo de interfaz de usuario, por lo que puede actualizar de forma segura nuestra interfaz de usuario. A continuación, podemos ejecutar la tarea llamando al método *execute()* del hilo de interfaz de usuario (Goransson, 2014).

Cuando se ejecuta una tarea asíncrona, la tarea pasa a través de 4 pasos:

onPreExecute(): éste método es invocado en el subproceso de interfaz de usuario antes de ejecutar la tarea. Este paso se utiliza normalmente para configurar la tarea, por ejemplo, mostrando una barra de progreso en la interfaz de usuario.

doInBackground(Params…): éste método es invocado en el subproceso de fondo inmediatamente después de que el método *onPreExecute()* termina de ejecutarse. Este paso se utiliza para realizar los cálculos de fondo que pueden requerir demasiado tiempo. Los parámetros de la tarea asíncrona se pasan como parámetro de entrada en *doInBackground(Params…).* El resultado del cálculo debe ser devuelto por este paso y se pasará de nuevo a la última etapa. En éste paso también se puede utilizar el método *publisProgress(Progress…)* para que se publiquen una o más unidades de progreso. Estos valores se publicarán en el hilo de interfaz de usuario en el paso *onProgressUpdate(Progress…).*

onProgressUpdate(Progress…): este método es invocado en el subproceso de interfaz de usuario después de una llamada al método *publishProgress(Progress…).* El momento de la ejecución no está definido. Se utiliza para mostrar cualquier forma de progreso en la interfaz de usuario, mientras que las tareas de fondo se siguen ejecutando. Por ejemplo, se puede utilizar para animar una barra de progreso o mostrar los registros en un campo de texto.

onPostExecute(Result): este método es invocado en el subproceso de interfaz de usuario después de que finalice el cálculo o tareas de fondo. El resultado del cálculo de fondo se pasa a este paso como un parámetro.

Una tarea puede ser cancelada en cualquier momento mediante la invocación del método *cancel(boolean).* La invocación de este método hará que las llamadas posteriores al método *isCancelled()* devuelvan *true.* Después de invocar éste método, en lugar de *onPostExecute(Object)* será invocado*onCancelled(Object)* y justo después será invocado *doInBackgrounc(Object[]).* Para asegurarse de que una tarea se cancela lo más rápido posible, siempre se debe comprobar el valor de retorno de *inCancelled()* periódicamente desde el método *doInBackground(object[])* y si es posible dentro de un bucle.

6. Aplicaciones ya existentes

Existen multitud de aplicaciones con contenidos educativos en los mercados virtuales. Pero las podemos diferenciar en 3 grandes grupos (Gomis, 2012):

- Para niños: Hoy en día los niños han nacido inmersos en el boom tecnológico y ellos hacen uso de las nuevas tecnologías como cualquier adulto. Muchas aplicaciones van dirigidas a ellos con contenidos atractivos y a la vez educativos. La mayoría de las aplicaciones se centran en el ámbito de las matemáticas (Alexander Alonso Molinero, 2012).

- De idiomas (Google, Google Play, 2015): Una de las formas menos comunes de aprender un idioma es mediante una aplicación móvil. Resulta dificultoso aprender un idioma a través de una aplicación móvil. Estas aplicaciones están orientadas al aprendizaje de vocabulario por medio de la repetición. Obviamente diseñadas para un público adulto.

- De ejercicio mental: Este tipo de aplicaciones es usado con el fin de entretenerse jugando y a la vez aprendiendo y desarrollando la mente. Es un tipo de aplicación extendido, del que encontramos numerosas aplicaciones. No nos instruye en un campo concreto, si no en el ejercicio mental y va dirigido a todos los públicos.

- De exámenes: El tipo de aplicación más parecido a la desarrollada en este Trabajo de Fin de Grado. Existen numerosas aplicaciones con preguntas test de exámenes de conducir u otro tipo de exámenes como oposiciones (Marta Fernandes, 2012).

Es de mencionar, que no se han encontrado aplicaciones enfocadas a un campo muy concreto, como lo hace esta aplicación. Esto también alimenta la necesidad de esta aplicación por parte de los alumnos. Debido a que no se encontró ninguna aplicación de aprendizaje sobre programación o contenidos de telecomunicaciones.

7. Servicios Web

1. Definición

Un servicio *web* es un sistema de *software* diseñado para permitir la interoperabilidad máquina a máquina en una red (W3C, 2015). Se trata de APIs que son publicadas, localizadas e invocadas a través de la *web*. Es decir, estas APIs se instalan en un servidor de Internet y otras aplicaciones o servicios *web* pueden invocar uno de sus servicios.

Una de las principales motivaciones del uso de servicios *web* es la independencia de plataforma. Podemos utilizar un servicio *web* con independencia del sistema operativo o el lenguaje de programación en el que se haya desarrollado el dispositivo que invoca el servicio. Al estar apoyado sobre el protocolo HTTP hace uso de sus sistemas de seguridad (https).

Tipos de servicios *web* (Gironés, 2012):

- Basados en SOAP: es el protocolo más utilizado en la actualidad. Fue creado por Microsoft, IBM y otros y en la actualidad opera bajo el auspicio de W3C. Utiliza HTTP como protocolo de transporte y los mensajes usan un formato XML. Aunque SOAP está muy extendido, no es adecuado para su uso en Android debido a su complejidad que provoca un menor rendimiento que otras alternativas.
- Basados en REST. El término REST se refiere a una arquitectura y no a un protocolo (como en SOAP). Hace uso directo del protocolo HTTP. Este enfoque de comunicación implica el seguimiento de las normas WWW. Siendo los principios básicos de la aplicación WWW los siguientes:
    - El transporte de datos mediante HTTP, haciendo uso de las operaciones de este protocolo: GET, POST, PUT y DELETE.
    - La invocación de servicios mediante el formato de direcciones URI unificado, que identifica un recurso en Internet a través de un nombre. Es importante mencionar que el concepto de recurso en este caso no está limitada a elementos estáticos, también comprende toda la información que es requerida para acceder a estos recursos como documentos transaccionales u órdenes (Pautasso, 2011).
    - La codificación de datos identificada mediante tipos MIME.

  Las ventajas de REST para su uso en Android radican en su simplicidad. De este modo obtendremos un mejor tiempo de respuesta y una disminución de sobrecarga, tanto en cliente como en servidor. A esto hay que añadirle la simplicidad de desarrollo en cliente y la escalabilidad del sistema.
  Como inconveniente es importante mencionar la volatilidad del estado. Es decir, cada petición es tratada de forma independiente a las anteriores sin poder establecer una relación con ellas.

Capítulo 5: Descripción del método de resolución del problema.

Para desarrollar esta aplicación he utilizado el SDK de Android sobre el IDE Eclipse para la parte de programación. La plataforma *Eclipse* se estructura en subsistemas que se ejecutan en uno o más *plug-ins*. Los subsistemas se construyen en la parte superior de un pequeño motor *runtime*. La siguiente ilustración muestra una vista simplificada de esto (Eclipse, 2014):

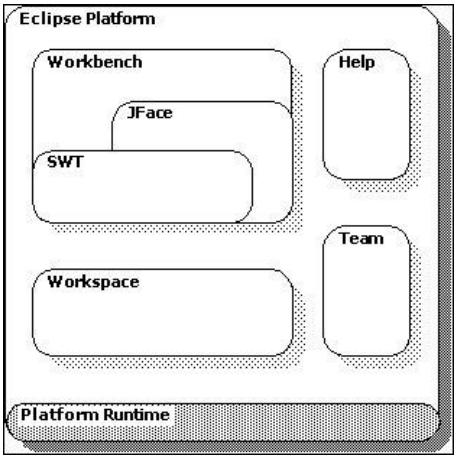

Ilustración 40: Estructura de la IDE Eclipse

El término *workbench* se refiere al entorno de desarrollo del escritorio. El *workbench* tiene como objetivo lograr la integración transparente de herramientas y la apertura controlada proporcionando un paradigma común para la creación, gestión y navegación de los recursos en el entorno de trabajo. Cada ventana del *workbench* tiene una o más perspectivas que contienen una forma de mostrar los datos, o una serie de vistas abiertas, de modo que cada ventana del *workbench* tenga una función específica.

Para este IDE, Google posee una herramienta de desarrollo de Android (ADT) que confiere al sistema de desarrollo un entorno potente e integrado en el desarrollo de aplicaciones Android. Google recomienda altamente el uso de *Eclipse* para el desarrollo de aplicaciones Android, por ser la forma más rápida y sencilla de comenzar tras *Android Studio*. Con la configuración guiada del proyecto que ofrece y la integración de herramientas, editores de XML personalizados ADT le da un gran impulso al desarrollo de aplicaciones Android.

Debido a que el módulo de *moodle eLiza* usa como servidor la plataforma XAMPP para su desarrollo he elegido también XAMPP como servidor local sobre el que ejecutar la aplicación Android de *eLiza*.

XAMPP es un servidor independiente de plataforma, software libre, que consiste principalmente en la base de datos *MySQL*, el servidor web Apache y los intérpretes para lenguajes de script: PHP y Perl. El nombre proviene del acrónimo de X (para cualquiera de los diferentes sistemas operativos), Apache, *MySQL*, PHP, Perl. El programa está liberado bajo la licencia GNU y actúa como un servidor web libre, fácil de usar y capaz de interpretar páginas

dinámicas. Actualmente XAMPP está disponible para *Microsoft Windows*, GNU/Linux, Solaris y Mac OS X, y el desarrollo ha sido sobre *Microsoft Windows 8* que es el sistema operativo del equipo que disponía (Friends).

Este servidor independiente sobre el que instalé el módulo de *moodle eLiza* me ha servido para constatar que ambas versiones de *moodle* conviven sin problemas, y que el mismo usuario puede hacer uso de ambas plataformas indistintamente. Ya que el objetivo de este trabajo era ampliar las plataformas de uso de *eLiza* sin restringir las ya existentes.

XAMPP posee la herramienta de gestión de bases de datos de *phpMyAdmin*. Esta herramienta ya la conocía por lo que fue razón de más para elegir XAMPP. *PhpMyAdmin* es una herramienta escrita en lenguaje de programación *web* php para la administración de bases de datos *MySQL* a través de Internet. Dentro de esta herramienta tenemos la posibilidad de crear, eliminar, hacer búsquedas, insertar datos o exportar e importar bases de datos.

Para comenzar la creación de la aplicación debía de trabajar sobre la base de datos actual de *eLiza* que actualmente se está usando. Por lo que importé dicha base de datos proporcionada por mi tutora a mi base de datos local mediante *phpMyAdmin*.

Para realizar el desarrollo de los documentos que acceden directamente a la base de datos y sirven de nexo de comunicación entre la aplicación y ésta he usado el programa de desarrollo de código *Notepad++*. La principal característica de este programa es su sencillez y la facilidad que da a la edición de archivos. La depuración de estos ficheros la he realizado desde la IDE *Eclipse*, a partir de los valores devueltos por estos ficheros y que son captados por la aplicación.

Estos ficheros están escritos en lenguaje php y su función se limita a recoger los parámetros que se pasan desde la aplicación mediante *POST* y que se usan para la realización de consultas en la base de datos o para la inserción o actualización de filas en ésta. Y posteriormente de la realización de estas funciones devuelven valores de confirmación de que el proceso se ha realizado correctamente o, se devuelven los datos accedidos de la base de datos que son requeridos desde la aplicación. Como por ejemplo, la actualización de una máxima puntuación en la clasificación o la obtención de las preguntas y respuestas de una asignatura.

Para realizar dichos accesos a la base de datos se hace uso de *MySQLi* (*Improved*), que es la versión mejorada de *MySQL*. Posee una estructura orientada a objetos que logra que la integración de *MySQL* se ajuste mejor a las características de la versión 5 PHP que es la que he usado (Corporation, 2012).

La mayor ventaja del uso de *MySQLi* para el programador es la existencia de instrucciones preparadas para el escape de cada cadena usada en una consulta para prevenir ataques de inyección. También presenta mejoras en la eficiencia debido a las instrucciones preparadas, de modo que al hacer un *INSERT* la sobrecarga del mismo, tendría que repetirse con cada consulta a la base de datos. Con la extensión *MySQLi* puedes agrupar un conjunto de operaciones de datos en una transacción y ejecutar las operaciones en conjunto como una sola

transacción. Si una operación de la transacción falla, la operación completa falla y todos los cambios que se hayan efectuado se revierten.

"Los ordenadores son inútiles. Sólo pueden darte respuestas"

Pablo Picasso

eLiza

2. Descripción técnica

Este apartado se dividirá en funcionalidades que la aplicación podrá realizar. En cada uno de ellos se analizará el funcionamiento de la aplicación y las actividades que están involucradas en ellos.

1. Registro y acceso de usuarios

1. Usuario no registrado en base de datos *moodle*

Al arrancar la aplicación se ejecuta como primera actividad la actividad *SplashScreen.* Esta actividad muestra al usuario una imagen con el rótulo de *eLiza* y los logos de la Universidad de Valladolid y el GTI. Mientras la aplicación realiza una serie de operaciones.

El tiempo que la aplicación muestra este *splash* viene marcado por la variable de tipo *long* llamada *splashDelay*, expresada en milisegundos. Un tiempo razonable para esto pueden ser 2 o 3 segundos. Ya que la paciencia del usuario es limitada y en ese tiempo la aplicación, en condiciones normales, ha tenido tiempo de sobra de realizar sus operaciones.

En el caso de que el usuario no esté registrado en la base de datos de *moodle* tampoco lo estará en nuestra base de datos de la aplicación. La aplicación verifica esto mediante una clase auxiliar creada únicamente para las interacciones con la base de datos interna de la aplicación llamada *DBHelper*, la cual contiene todos los métodos de los que la aplicación hará uso para interaccionar con la base de datos interna de la aplicación. En este caso hará uso del método *getLastUser*, que no hace otra cosa que una búsqueda de una fila con el campo *lastUser* a 1 en la tabla *user* y nos devuelve un dato de tipo *Cursor* con todos los datos de dicho usuario. Si el *Cursor* recibido tiene tamaño 0 es que no existe ningún usuario registrado en nuestra aplicación como sería este caso.

La aplicación entonces pone en marcha el procedimiento de registro para el usuario. Mediante la aplicación el usuario se podrá registrar como jugador en el terminal, pero también como usuario de *moodle* en la base de datos del GTI. De este modo se lanza una nueva actividad llamada *MainActivity*, en ella el usuario tiene 2 modos de acceso a la aplicación dependiendo de su condición de registrado o no registrado. En la parte superior puede introducir usuario y contraseña si es usuario registrado de *moodle*, y pasar a la parte final del registro. O si no está registrado en la base de datos de *moodle* pulsará el botón de la parte inferior de la pantalla, como es el caso de la funcionalidad que estamos analizando, y pasará a realizar el registro completo.

Esto se desarrolla de un modo bien sencillo, en la definición de la interfaz gráfica de la actividad se define el botón que pulsaremos, como atributo de este elemento incluimos *onClick* y le agregamos como valor el nombre del método que se ejecutará al pulsar dicho botón, en este caso el método tiene por nombre *firstEntryOnClick*. Por definición, para que el evento de pulsación del botón dispare una llamada, este método deberá ser de tipo *public* devolver *void* y tener como único parámetro de entrada una variable de tipo *View*. Dentro de ese método es desde donde lanzaremos la siguiente actividad llamada *RegisterActivity*, el primer paso del proceso de registro del usuario.

En el momento en que el usuario pulsa el botón de registro de un nuevo usuario, se llama al método declarado en el *layout* de la actividad donde se fija el atributo *onClick* del botón. Previamente al lanzamiento de la actividad de registro, la aplicación comprueba si el dispositivo móvil posee acceso a Internet. Esto se debe a que sin Internet de ningún modo podremos acceder a la base de datos *Moodle* alojada en el servidor. Esta comprobación se realizará en lo posterior cada vez que pasemos de una actividad a otra, hasta que finalmente completemos el registro y se realice una inserción de los datos del usuario de forma atómica. En el caso de que el dispositivo no tenga acceso a Internet se denegará el registro y se informará de ello a usuario a través de un *Toast*.

Al acceder a la actividad *RegisterActivity* se muestra un formulario en el que deberemos rellenar todos los campos de los que consta, a saber:

- Nombre de usuario
- Contraseña
- Nombre
- Apellidos
- Ciudad
- Correo electrónico

También se muestra un botón en la parte inferior de la pantalla que deberemos pulsar cuando hayamos completado el formulario anteriormente mencionado. La aplicación realiza una serie de comprobaciones de los datos introducidos por el usuario tras pulsar el botón de "Siguiente" que confirma que ha sido rellenado el formulario. Estas son las comprobaciones que realiza:

- Se asegura de que en la base de datos de *moodle* no exista otro usuario con el mismo *nick*. No es necesario mirar en la base de datos interna del dispositivo, ya que si no existe en la base de datos de *moodle* de ninguna manera podrá existir en la base de datos del dispositivo. Para ello, la aplicación hace uso del servicio *Web* REST. Crea un objeto de la clase *JSONParser* que realizará todos los accesos al servicio *Web*.
  La clase *JSONParser* consta de un único método llamado *getJSONFromUrl*, el cual devuelve un objeto de tipo *JSONObject* y recibe como argumentos de entrada:
  - el nombre de la URL a la que ha de acceder para comunicarse con la base de datos, de tipo *String.* Contendrá la URL donde estará alojado el fichero que accede a la base de datos de *moodle*, en este caso el fichero esta nombrado como *form.php*.

- una variable de tipo *ArrayList* formada por elementos del tipo *NameValuePair* que tiene un valor asociado a un nombre clave para cada elemento del *List.* Contendrá los argumentos de entrada del servicio *Web,* en este caso, el par valor-nombre hará referencia al *nick* del usuario que queremos verificar si existe en la base de datos.

Este archivo *form.php* recibe el parámetro del *nick* que el método *getJSONFromUrl* le pasa. Conecta con la base de datos y realiza una búsqueda en la tabla *mdl_user* en la que se busca una fila cuyo campo *username* coincida con el *nick*. Si la búsqueda concluye con 0 resultados, significa que el nombre de usuario elegido esta libre, y devuelve mediante un *echo* una variable de tipo *JSON* en el cual según el identificador *status* se indica si el nombre está disponible mediante *ok,* o al contrario se encuentra ocupado *nook*. La aplicación recibe esta información y procesa los datos recibidos.

Si el nombre de usuario elegido estuviese ocupado la aplicación lo comunicaría al usuario mediante un *Toast*. Debido a que esta consulta se ha debido realizar en un hilo adicional al principal de la aplicación (ya que estamos accediendo a una base de datos) no podemos hacer cambios en la interfaz del usuario. Es necesario en este caso la creación de un *Handler* declarado como método *private* que recibe un parámetro de tipo *Message* que en este caso hemos llenado con el mensaje que queremos mostrar en el *Toast*. Este pequeño método *Handler* únicamente toma el valor incluido en el *Message* y lo muestra por pantalla mediante un *Toast*.

- Se asegura que el formato del correo electrónico sea correcto. Es decir, verificar que al menos hay dos caracteres al que siga una "@", que después al menos haya otros dos caracteres al que siga un "." y que como último elemento al menos haya 2 letras correspondientes al dominio. En caso contrario, se muestra un *Toast* al usuario informado de esto, mediante el método *private Handler*.
- Se asegura de que la contraseña sea válida, es decir, que tenga al menos 6 caracteres de los que uno al menos ha de ser un número, exista al menos una letra minúscula y otra mayúscula y que haya al menos un carácter especial de estos: "` ~ ! @ # $ % ^ & * ( ) _ - + = { } [ ] \ | : ; " ' < > , . ? /". En caso contrario mediante el *Handler* anteriormente mencionado mostramos un *Toast* por pantalla indicándole al usuario el formato incorrecto de la contraseña.
- Obviamente se ha de asegurar de que todos los campos han sido rellenados, es decir, de que no haya ninguno que se encuentre vacío. En caso contrario mediante el *Handler* anteriormente mencionado mostramos un *Toast* por pantalla indicándole al usuario que existen aún campos sin rellenar.

Tras verificar que todos los campos son correctos, el hilo principal de la aplicación procede a la encriptación de la contraseña introducida por el usuario. La encriptación se realiza mediante el algoritmo MD5. Para la realización de la aplicación no se ha elegido el algoritmo de encriptación, si no que se ha utilizado el mismo que usa *Moodle* para encriptar las contraseñas de sus usuarios, y que así de esta manera, se pueda compartir la tabla de usuarios además de compatibilizar el acceso a la misma plataforma desde diferentes dispositivos. De no haber sido así, probablemente se hubiese elegido un algoritmo de cifrado más complejo que proporcionara mayor resistencia ante ataques como por ejemplo *Blowfish* que ya se usa en versiones más actuales de *moodle*.

Una vez se haya comprobado que todos los campos están completos y con el formato correcto pasamos a la siguiente actividad del registro. Estos valores recogidos se los pasamos mediante el *Intent* de la siguiente clase, y cuando el registro haya finalizado los introduciremos en la base de datos de forma atómica. De este modo nos aseguramos de que si el registro se aborta no se produzcan problemas en la gestión de las bases de datos.

En la siguiente actividad llamada *RegisterCourseActivity* se muestra un *ListView* con los cursos que poseen un módulo de *moodle*, pulsar el botón de la parte inferior de la pantalla para continuar completando el registro. En el caso de que el usuario pulsase el botón de siguiente sin haber seleccionado ningún curso, la aplicación bloqueará el proceso y mostrará un Toast. Al iniciarse la actividad ésta lanza un hilo en *background* que accede mediante *getJSONFromURL* de la clase *JSONParser* al fichero *course.php* del servicio *Web.* Éste realiza una búsqueda de los curso que tienen un módulo de *eLiza* en la base de datos, y sólo mostrará como cursos elegibles. Para ello recoge los identificadores de curso de la tabla *mdl_eliza_categories* que a su vez, contiene a las categorías de juego. Posteriormente realiza una búsqueda de curso en la tabla *mdl_course_categories* en la cual se pone como condición que el identificador del curso coincida con el de las categorías anteriormente accedidas y este a su vez con los identificadores de curso de la tabla que contiene todas las asignaturas, *mdl_course*. De este modo nos aseguramos de que los cursos que mostraremos están presentes como módulo activo de *eLiza* y también en las tablas relacionadas. Esto es necesario ya que actualmente en la base de datos de *Moodle* hay cursos sin asignaturas y asignaturas que pertenecen a cursos pero no tienen módulo de *eLiza.*

Para llenar este *ListView* se ha creado un adaptador que es a su vez completado mediante un array de la clase *CourseModel*, esta clase contiene variables globales que identifican a cada uno de los cursos en sus objetos, y también posee métodos de modificación y obtención de cada una de las variables del curso.

Este proceso de acceso a base de datos y llenado de la interfaz puede tardar unos segundos en los que el usuario, no debería tocar ningún elemento de la interfaz hasta que esta esté completada y la aplicación haya obtenido la información necesaria para que el usuario realice la selección. Por esto, se ha incluido al inicio del método *onCreate,* que crea la interfaz visual de la actividad al ser lanzada, un *progress dialog*, que no es más que una pequeña ventana que bloquea la interacción del usuario con la aplicación y a su vez le informa de que hay operaciones llevándose a cabo y debe esperar unos segundos. Al bloquear la interacción del usuario con la aplicación, mientras ésta realiza operaciones, se evitan actuaciones no deseadas por parte del usuario que puedan llevar a la aplicación a un estado de inestabilidad.

Tras estos procesos el usuario seleccionará su curso y pulsará el botón de continuar para seleccionar sus asignaturas en la siguiente actividad llamada *RegisterSubjectsActivity.* El proceso seguido en esta actividad es muy similar al seguido en la anterior, pero con algunas diferencias. Al igual que en la anterior actividad debido a que se requiere un acceso a la base de datos que puede llevar unos segundos, se lanza un *progress dialog* para informar al usuario. También se lanza el *thread* de *background* para el acceso a la base de datos. En este caso, la aplicación envía al servicio *Web*, mediante el método *getJSONFromUrl* de la clase *JSONParser*, el identificador del curso seleccionado en la actividad anterior. El fichero que recibe este

argumento de entrada es *subject.php*, que realiza una búsqueda en la tabla *mdl_course*, que contiene la información relativa a las asignaturas, donde el identificador del curso coincida con el que se le ha pasado. Mediante un *echo* devuelve un objeto de tipo *JSON* que contiene en cada posición el nombre y el identificador de cada asignatura perteneciente al curso seleccionado. En la aplicación se recibe esta información y se rellenan los arrays de datos pertenecientes a las asignaturas que al igual que se hizo en el caso de los cursos serán usados para llenar el *ListView*. La única diferencia que presenta el *ListView* de asignaturas con el de cursos es que podemos seleccionar más de una asignatura, por lo que incluir *RadioButtons* no es una elección adecuada. Los *CheckBoxes* en cambio nos permiten realizar una selección múltiple no excluyente, este elemento sustituye a los anteriores *RadioButtons.*

Tras realizar esta última selección, se introducen todos los datos recabados durante el registro en la base de datos. En este momento se hace una búsqueda en la base de datos de *moodle* para ver si es necesario registrarlo. Si es así, se envían todos los datos al servicio *Web* accedido desde el fichero *register.php* mediante el método *getJSONFromUrl* de la clase *JSONParser*. Desde el fichero *register.php* se insertarán los datos en la base de datos, en concreto en la tabla que recoge los datos de los usuarios *mdl_user*. Esta operación se realiza de forma atómica. La aplicación se ha cerciorado de que todos los datos que han llegado hasta este punto son válidos, y que las condiciones para realizar el registro son correctas. De modo que si cualquier mecanismo de la aplicación fallase, la aplicación no registraría al usuario.

El siguiente paso es ver si se necesita introducir los datos del usuario en la base de datos del terminal, en este caso sí es necesario. La aplicación insertará datos de usuario como identificador, *nick,* contraseña cifrada en la tabla *user*, y también insertará en la tabla *subject* una correspondencia entre el usuario y las asignaturas de los cursos que ha seleccionado, de modo que, cada usuario quede identificado con qué asignaturas posee.

Una vez concluido el registro, la aplicación dirige al usuario a la pantalla *home*. Desde aquí el usuario podrá hacer un uso normal de la aplicación y sus tareas de registro ya habrán finalizado.

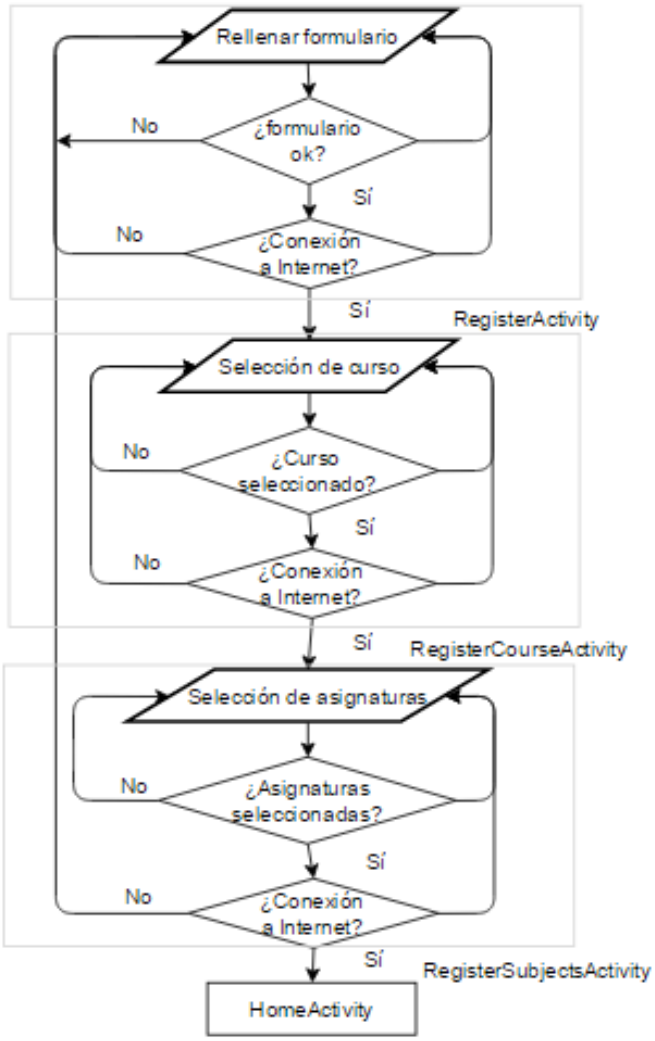

Diagrama 1: Proceso de registro

2. Usuario registrado en la base de datos

Este proceso es muy similar al anterior con algunas excepciones, debido a que hay pasos que en este caso nos saltaremos ya que no son necesarios pues el usuario ya está registrado en la base de datos. Este podría ser el caso de cualquier alumno que ya ha utilizado el módulo de *moodle* de *eLiza* desde su plataforma *Web* y *a posteriori* descarga la aplicación en su móvil.

Al arrancar la aplicación se ejecuta como primera actividad la actividad *SplashScreen.* Esta actividad muestra al usuario una imagen con el rótulo de *eLiza* y los logos de la Universidad de Valladolid y el GTI. Mientras la aplicación realiza una serie de operaciones.

El tiempo que la aplicación muestra este *splash* viene marcado por la variable de tipo *long* llamada *splashDelay*, expresada en milisegundos. Un tiempo razonable para esto pueden ser 2 o 3 segundos. Ya que la paciencia del usuario es limitada y en ese tiempo la aplicación, en condiciones normales, ha tenido tiempo de sobra de realizar sus operaciones.

En el momento en que el usuario pulsa el botón de registro de un nuevo usuario, se llama al método declarado en el *layout* de la actividad donde se fija el atributo *onClick* del botón. Previamente al lanzamiento de la actividad de registro, la aplicación comprueba si el dispositivo móvil posee acceso a Internet. Esto se debe a que sin Internet de ningún modo podremos acceder a la base de datos *Moodle* alojada en el servidor. Esta comprobación se realizará en lo posterior cada vez que pasemos de una actividad a otra, hasta que finalmente completemos el registro y se realice una inserción de los datos del usuario de forma atómica. En el caso de que el dispositivo no tenga acceso a Internet se denegará el registro y se informará de ello a usuario a través de un *Toast*.

En este caso el usuario está registrado en la base de datos de *moodle* pero no está registrado como usuario en la base de datos local del terminal. La aplicación verifica esto mediante una clase auxiliar creada únicamente para las interacciones con la base de datos interna de la aplicación llamada *DBHelper*, la cual contiene todos los métodos de los que la aplicación hará uso para interaccionar con la base de datos interna de la aplicación. En este caso hará uso del método *getLastUser*, que no hace otra cosa que una búsqueda de una fila con el campo *lastUser* a 1 en la tabla *user* y nos devuelve un dato de tipo *Cursor* con todos los datos de dicho usuario. Si el *Cursor* recibido tiene tamaño 0 es que no existe ningún usuario registrado en nuestra aplicación como sería este caso.

Debido a que el usuario ya está registrado en la base de datos de *moodle* pasamos directamente a la actividad de selección del curso. En esta actividad llamada *RegisterCourseActivity* se muestra un *ListView* con los cursos que poseen un módulo de *moodle*, pulsar el botón de la parte inferior de la pantalla para continuar completando el registro. En el caso de que el usuario pulsase el botón de siguiente sin haber seleccionado ningún curso, la aplicación bloqueará el proceso y mostrará un *Toast*. Al iniciarse la actividad ésta lanza un hilo en *background* que accede mediante *getJSONFromURL* de la clase *JSONParser* al fichero *course.php* del servicio *Web.* Éste realiza una búsqueda de los cursos que tienen un módulo de *eLiza* en la base de datos, y sólo mostrará como cursos elegibles aquellos que posean asignaturas asociadas a categorías de juego y que a su vez posean preguntas y

respuestas ya definidas. Para ello recoge los identificadores de curso de la tabla *mdl_eliza_categories* que a su vez, contiene a las categorías de juego. Posteriormente realiza una búsqueda de identificador de curso en la tabla *mdl_course_categories* en la cual se pone como condición que el identificador del curso coincida con el de las categorías anteriormente accedidas y este a su vez con los identificadores de curso de la tabla que contiene todas las asignaturas, *mdl_course*. Así nos aseguramos de que los cursos que mostraremos están presentes como módulo activo de *eLiza* y también en las tablas que se relacionadas. Esto es necesario ya que hay cursos sin asignaturas y asignaturas que pertenecen a cursos pero no tienen módulo de *eLiza.*

Para llenar este *ListView* se ha creado un adaptador que es a su vez completado mediante un array de la clase *CourseModel*, esta clase contiene variables globales que identifican a cada uno de los cursos en sus objetos, y también posee métodos de modificación y obtención de cada una de las variables del curso.

Este proceso de acceso a base de datos y llenado de la interfaz puede tardar unos segundos durante los que el usuario, no debería tocar ningún elemento de la interfaz hasta que esta esté completada y la aplicación haya obtenido la información necesaria para que el usuario realice la selección. Por esto, se ha incluido al inicio del método *onCreate,* que crea la interfaz visual de la actividad al ser lanzada, un *progress dialog*, que no es más que una pequeña ventana que bloquea la interacción del usuario con la aplicación y a su vez le informa de que hay operaciones llevándose a cabo y debe esperar unos segundos.

Tras estos procesos el usuario seleccionará su curso y pulsará el botón de continuar para seleccionar sus asignaturas en la siguiente actividad llamada *RegisterSubjectsActivity.* El proceso seguido en esta actividad es muy similar al de la actividad anterior, pero con algunas diferencias. Al igual que en la anterior actividad debido a que se requiere un acceso a la base de datos que puede llevar unos segundos, se lanza un *progress dialog* para informar al usuario y prevenir a la aplicación de interacciones no deseadas que falseen el comportamiento de la misma. También se lanza el *thread* de *background* para el acceso a la base de datos. En este caso, la aplicación envía al servicio *Web*, mediante el método *getJSONFromUrl* de la clase *JSONParser*, el identificador del curso seleccionado en la actividad anterior. El fichero que recibe este argumento de entrada es *subject.php*, que realiza una búsqueda en la tabla *mdl_course*, la cual contiene la información relativa a las asignaturas, donde el identificador del curso coincida con el que se le ha pasado. Mediante un *echo* devuelve un objeto de tipo *JSON* que contiene en cada posición el nombre y el identificador de cada asignatura perteneciente al curso seleccionado. En la aplicación se recibe esta información y se rellenan los arrays de datos pertenecientes a las asignaturas que, al igual que se hizo en el caso de los cursos, serán usados para llenar el *ListView*. La única diferencia que presenta el *ListView* de asignaturas con el de cursos es que podemos seleccionar más de una asignatura, por lo que incluir *RadioButtons* no es adecuado. Los *CheckBoxes* en cambio nos permiten realizar una selección múltiple, este elemento sustituye a los anteriores *RadioButtons.*

Tras realizar esta última selección, se introducen todos los datos recabados durante el registro en la base de datos. La aplicación insertará datos de usuario como identificador, *nick,* contraseña cifrada en la tabla *user*, y también insertará en la tabla *subject* una

correspondencia entre el usuario y las asignaturas de los cursos que ha seleccionado, de modo que, cada usuario quede identificado con qué asignaturas posee.

Una vez concluido el registro, la aplicación dirige al usuario a la pantalla *home*. Desde aquí el usuario podrá hacer un uso normal de la aplicación y sus tareas de registro ya habrán finalizado.

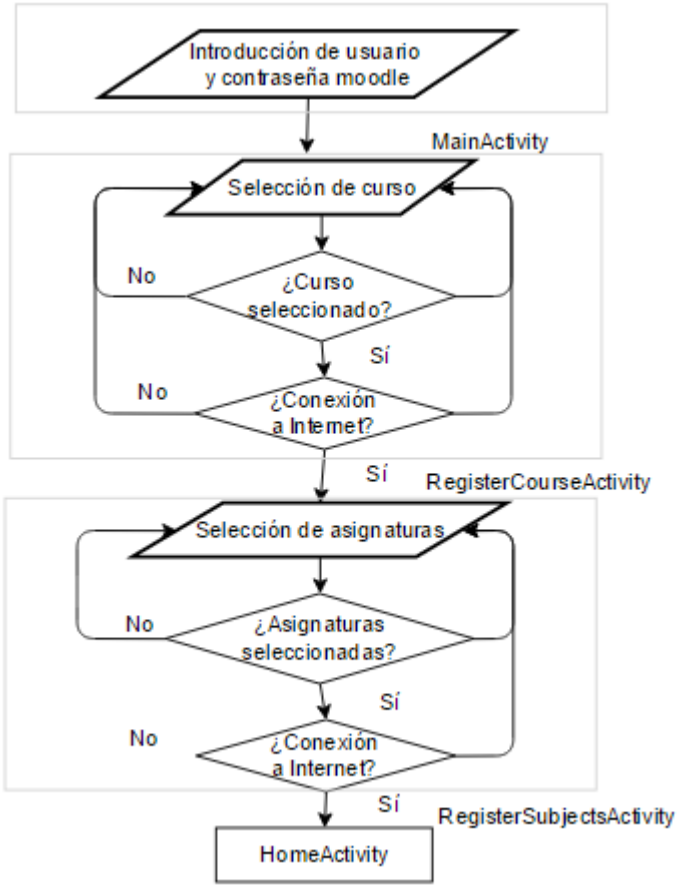

Diagrama 2: Proceso de registro de usuario existente

3. Posteriores accesos

Debido a que un terminal móvil es algo personal, presuponemos que el la mayoría de los accesos a la aplicación serán realizados por el propietario del terminal. De este modo se ha intentado maximizar la comodidad en el acceso a la aplicación.

Un usuario que ya haya sido registrado en ambas bases de datos y además sea el último usuario que se registró en la aplicación del terminal entrará, tras pasar por la actividad *SplashScreenActivity*, directamente a la pantalla *home*. Pensando en facilitar y acelerar al máximo el acceso. Pero internamente la aplicación realiza una serie de operaciones que vamos a describir.

Al arrancar la aplicación se ejecuta como primera actividad la actividad *SplashScreen.* Esta actividad muestra al usuario una imagen con el rótulo de *eLiza* y los logos de la Universidad de Valladolid y el GTI. Mientras la aplicación realiza una serie de operaciones.

El tiempo que la aplicación muestra este *splash* viene marcado por la variable de tipo *long* llamada *splashDelay*, expresada en milisegundos. Un tiempo razonable para esto pueden ser 2 o 3 segundos. Ya que la paciencia del usuario es limitada y en ese tiempo la aplicación, en condiciones normales, ha tenido tiempo de sobra de realizar sus operaciones.

En este caso el usuario está registrado en la base de datos de *moodle* pero y también lo está en la base de datos local del terminal. La aplicación verifica esto mediante una clase auxiliar creada únicamente para las interacciones con la base de datos interna de la aplicación llamada *DBHelper*, la cual contiene todos los métodos de los que la aplicación hará uso para interaccionar con la base de datos interna de la aplicación. En este caso hará uso del método *getLastUser*, que no hace otra cosa que una búsqueda de una fila con el campo *lastUser* a 1 en la tabla *user* y nos devuelve un dato de tipo *Cursor* con todos los datos de dicho usuario. Si el *Cursor* recibido tiene tamaño 1 es que existe un usuario registrado en nuestra aplicación como sería este caso y que además es el último que la usó.

La aplicación aprovechando los segundos que le da el *splash* para realizar operaciones de volcado de datos, entra en la base de datos de *moodle* mediante el método *getJSONFromUrl* de la clase *JSONParser* para obtener las preguntas y respuestas actualizadas de la base de datos. Estas preguntas accedidas se encuentran en la tabla *mdl_eliza_preguntas* y las respuestas en *mdl_eliza_respuestas*. El fichero que accede a la base de datos recibe de la aplicación una llamada para cada una de las asignaturas que ese usuario posea, de este modo actualizaremos por completo la base de datos de preguntas y respuestas del usuario que está accediendo. El fichero llamado *question.php* recibe como parámetros de entrada los identificadores de las categorías de los módulos de preguntas de las asignaturas. Posteriormente realiza la búsqueda de las preguntas asociadas a dicho identificador. Con el identificador de cada pregunta, en un bucle *for*, realiza una búsqueda recursiva de las respuestas de cada pregunta y las almacena en un array contenido como elemento de array de preguntas. Este array incluye los campos del texto de la respuesta y un campo numérico que indica cual es la respuesta correcta. Esta información es devuelta hacia la aplicación mediante un *echo* de los datos introducidos en un objeto de tipo *JSON*. La aplicación recoge estos datos y los envía, como parámetro de entrada, a un método de la clase *DBHelper*, que gestiona la base de datos local del dispositivo, llamado *insertQuestAnsw*. Este método recibe el objeto *JSON* del que extrae la información, la compara con la existente en la base de datos, y si no coincide, la actualiza.

La principal razón de este procedimiento es reducir el número de accesos a base de datos durante el juego, debido a que lastra a la aplicación introduciendo muchos tiempos de espera para el usuario. De este modo, en cada acceso se realizará una actualización de la información del juego y así durante el juego la aplicación correrá fluidamente.

En el caso de que la aplicación no dispusiese de acceso a Internet en el momento en el que el usuario, ya registrado, abre la aplicación: las operaciones de actualización de contenidos

anteriormente mencionadas no se realizan. Pese a esto, el usuario podrá acceder normalmente a la aplicación ya que los datos de verificación de identidad se encuentran en la base de datos local de la aplicación. De modo que se verifica la identidad del usuario mediante una clase auxiliar creada únicamente para las interacciones con la base de datos interna de la aplicación llamada *DBHelper*. Se hará uso del método *getLastUser*, que realiza una búsqueda de una fila con el campo *lastUser* a 1 en la tabla *user* y nos devuelve un dato de tipo *Cursor* con todos los datos de dicho usuario. Si el *Cursor* recibido tiene tamaño 1 es que existe un usuario registrado en nuestra aplicación como sería este caso y que además es el último que la usó. Y posteriormente el usuario accederá a la actividad *home*.

4. Cambio de usuario

Para realizar un cambio de usuario desde la pantalla *home, classification, statistic* o *achievements* hemos de pulsar la tecla atrás. En ese momento el método *onKeyDown* es ejecutado por el sistema y recibe 2 parámetros de entrada identificando el tipo de evento y qué tecla se pulsó. De este modo podemos identificar que se pulsó la tecla atrás. En ese momento la aplicación muestra un *dialog* que pregunta al usuario si quiere permanecer en la aplicación, salir de la aplicación, o cerrar la sesión. Pulsamos cerrar sesión. Entonces mediante el método *quitLastUser* de la clase *DBHelper* se pone a 0 el campo del usuario de la tabla *user* que tuviese el campo *lastUser* marcado a 1, es decir, el correspondiente al usuario que cierra la sesión. E inmediatamente somos dirigidos a la actividad *MainActivity* donde podremos registrarnos de la manera que deseemos.

2. Comenzar a jugar

Nos encontramos en la pantalla *home* de la actividad *HomeActivity*, en la que visualizamos un *Spinner* que nos permite seleccionar una asignatura. Debajo de él se encuentra un *ToggleButton* que activa o desactiva la visualización de un tutorial rápido de la aplicación. Este tutorial ha sido diseñado para que con un vistazo cualquier usuario sea capaz de comprender el funcionamiento del juego antes de incluso usarla por primera vez. Al pulsar sobre él en la parte inferior aparecerán unas instrucciones rápidas que nos dan una idea de cómo funciona la mecánica del juego. La aplicación internamente dentro del método de retrollamada del *ToggleButton* distingue si este es activado o desactivado y actúa en consecuencia. De modo que mediante el método *setVisibility,* aplicable a objetos descendientes de la clase *View,* podemos ocultar o hacer visible el tutorial en cada caso. Si volvemos a pulsar el *ToggleButton* este breve tutorial desaparece.

Para acceder a jugar el usuario debe seleccionar una asignatura del *Spinner* y pulsar el botón que se encuentra debajo de él. En el caso de que el usuario pulse jugar sin haber seleccionado asignatura alguna, se mostrará un *Toast* informando de esto, y bloqueará el acceso.

Una vez seleccionada la asignatura del *Spinner* y pulsado el botón inferior se accede a la actividad *QuesIntroActivity*. Esta actividad muestra en un primer momento, las categorías que posee la asignatura seleccionada que se identifican con cada una de las columnas que aparecen. En la parte central de la pantalla columnas de botones del 100 al 500 que significan los 5 niveles de dificultad dentro de cada categoría del juego. De este modo veremos tantas columnas como categorías de juego tenga la asignatura con la que estemos jugando.

En este momento el juego aún no ha comenzado, de modo que si el usuario pulsa atrás y salir, no se computará la partida. En el momento que el usuario selecciona una pregunta de las categorías, el juego comenzará, y se mostrará en la parte superior el tiempo restante y la puntuación que se inicia en 0. Para implementar la cuenta atrás del tiempo restante de juego se ha creado un objeto de la clase *CountDownTimer* que recibe como argumentos de entrada el valor temporal en milisegundos desde el que se comienza la cuenta atrás y el frecuencia de tiempo en milisegundos con la que se ejecutará el método constantemente hasta que el objeto *CountDownTimer* sea cancelado. Dentro de este objeto se declaran dos métodos. Uno de ellos se ejecuta cuando la cuenta atrás finaliza, y que se detallará más adelante. El otro, llamado *onTick* es ejecutado en cada interacción del periodo de tiempo definido, en el caso de esta aplicación, cada segundo. En él lo que se hace es actualizar la interfaz temporal de modo que se vaya mostrando la cuenta atrás segundo a segundo.

Volviendo al juego, el fondo del botón pulsado pasa a ser de color gris, determinándose así como botón marcado. Internamente, la aplicación ejecuta el método de retrollamada asignado para los botones de nombre *pointOnClick*. Una vez dentro de ese método distinguiremos de que categoría procede la llamada gracias a la etiqueta del botón, que obtendremos mediante el método *getTag,* y el nivel de dificultad del botón pulsado, que obtendremos mediante el método *getText* el cual devuelve el texto del botón: 100…500. El método de retrollamada *pointOnClick*, una vez ha sido reconocido el botón realiza en la base

de datos del dispositivo una búsqueda aleatoria de una pregunta que se ajuste a la asignatura, categoría y dificultad seleccionada. Este proceso de búsqueda es ejecutado invocando un método sobre un objeto creado de la clase *DBHelper*, este método es *getQuestion*. *GetQuestion* recibe como parámetros de entrada el identificador de la categoría y la dificultad y realiza la búsqueda en la tabla *question*. Una vez es seleccionada una pregunta que coincide con los criterios se accede a la tabla de respuestas llamada *answer* y se obtienen las respuestas correspondientes a la pregunta seleccionada. El método devuelve un *String[]* que contiene el identificador de la pregunta, su descripción, el texto de la pregunta, cada posible respuesta y el entero que indica si la respuesta es correcta o no.

La aplicación de vuelta con los datos requeridos muestra al usuario, debajo de las columnas de categorías la descripción de la pregunta, y debajo de esta, 2 botones: contestar o pasar. La descripción mostrada debe influir al alumno a decidir si contesta la pregunta o por contrario prefiere pasarla, ya que los fallos penalizan el marcador restando la cantidad que corresponda a su dificultad. En este punto no se puede pulsar otro botón que no sea contestar o pasar, de hacerlo el usuario será informado mediante un *Toast.* Si el usuario decide pasar este procedimiento se volvería a repetir.

Si el usuario decide contestar la aplicación pasa a la siguiente actividad llamada *QuestActivity,* a esta actividad se le pasan los datos de la pregunta accedida en la base de datos para que los muestre al usuario. En ella, en la parte superior visualizaremos el tiempo restante y la puntuación igual que en la actividad anterior pero a la derecha de estos se encuentra un botón que al pulsarlo nos muestra el contenido de la pregunta en pantalla completa. Esto nos será útil cuando la pregunta sea muy extensa, como por ejemplo cuando consta de una porción de código. Debajo todo de esto, está el enunciado de la pregunta. Debajo de la pregunta se encuentra un *ListView* rellenado con las posibles respuestas. Para llenar este *ListView* se ha creado un adaptador que es a su vez completado mediante un array de la clase *AnswerModel*, esta clase contiene variables globales que identifican a cada una de las respuestas en sus objetos, y también posee métodos de modificación y obtención de cada una de las variables del la respuestas, a saber: posición de la respuesta, texto de la respuesta, el entero que indica si la respuesta es la correcta o no y el *booleano* que indica si la respuesta ha sido seleccionada por el usuario. Para cada respuesta que forma el ListView, se ha creado un método de retrollamada lanzado al realizar una pulsación prolongada en una de las respuestas. La función de este método llamado *fullScreenAnswerOnClick* es lanzar la actividad *FullScreenActivity* a la que pasa como parámetro el contenido de la pregunta que será mostrada en esta actividad a pantalla completa. Lo que motiva el desarrollo de esta actividad es acomodar el uso de la actividad a las necesidades del usuario. Debajo de la lista de respuestas se encuentra el botón que deberemos pulsar para confirmar la respuesta que hayamos seleccionado. Al pulsarlo, el elemento de la lista se pondrá de color verde al acertar o rojo al fallar y oiremos un sonido identificativo en cada uno de los casos.

Tras mostrar la información por pantalla la aplicación internamente tiene 2 *listener* esperando detectar la interacción del usuario con las respuestas mediante el *RadioButton* o mediante el resto de la superficie del elemento del *ListView.* Estos *listeners* actualizarán los objetos de la clase *AnswerModel* que contienen la información de cada respuesta,

concretamente el *booleano* que contiene la información de si la respuesta está seleccionada o no.

El botón de confirmación de la respuesta también está asociado a un *listener* que es activado al arrancar la actividad mediante la llamada al método *checkButtonClick.* De este modo, ese método es ejecutado al confirmar nuestra respuesta. Se encarga de ver si la respuesta seleccionada es la correcta y manejar la interfaz en cada caso, mostrando la respuesta del color que proceda, lanzando el audio que corresponda y actualizando el valor de la puntuación. También se encarga del mantenimiento de los valores estadísticos y logros, mediante el método *insertAnswerStatistic* definido en la clase *DBHelper*. De este modo aumentará o reseteará los contadores de preguntas acertadas o falladas que proceda. A saber:

- Número de preguntas respondidas de una asignatura y dificultad concretas. Aumenta este valor en una unidad por cada pregunta respondida.
- Número de preguntas respondidas correctamente consecutivamente. Aumenta este valor en una unidad por cada pregunta acertada y lo resetea a 0 cuando fallamos una pregunta.
- Actualiza el valor de la variable que registra si el logro se ha conseguido para el caso del n preguntas acertadas consecutivamente.

Este proceso termina cuando el tiempo termine o cuando el usuario desee terminar de forma manual el juego. En ambas actividades el tiempo viene gestionado por un objeto de la clase *CountDownTimer*, el cual, como argumentos de entrada requiere el tiempo restante en formato *long* y como segundo parámetro, cada cuánto tiempo en milisegundos se accede de nuevo al método del objeto, en este caso el valor es 1000 para que cada segundo que corre se refleje en el *TextView* de la parte superior de la pantalla. En el caso de que el tiempo finalice, se realiza una retrollamada al método *onFinish* declarado en la clase *CountDownTimer*. Se encarga de lanzar la actividad *GameFinishedActivity* que hace recuento de la puntuación y ejecuta operaciones de almacenamiento de datos recopilados durante el juego y que se detalla más adelante.

Para finalizar la partida en curso desde cualquiera de las 2 actividades mencionadas hemos de pulsar la tecla atrás. En ese momento el método *onKeyDown* es ejecutado por el sistema y recibe 2 parámetros de entrada identificando el tipo de evento y qué tecla se pulsó. De este modo podemos identificar que se pulsó la tecla atrás. En ese momento la aplicación muestra un *dialog* que nos pregunta si queremos permanecer en el juego o por contrario finalizarlo con la puntuación alcanzada hasta ese momento.

Tras haber finalizado el juego por cualquiera de las 2 formas de finalizarlo (al haber acabado el tiempo o haber salido manualmente) la aplicación nos conduce a la actividad *GameFinishedActivity*. Visualmente lo que el usuario percibe es un recuento de puntuación. En el centro de la pantalla el número va contando desde 0 hasta la puntuación alcanzada. Pero internamente son numerosas las operaciones que realiza.

Al iniciarse la actividad se lanza un *thread* en *background*, que se encarga de verificar si la puntuación obtenida es la más alta conseguida por ese usuario en esa asignatura. Esto lo hace invocando el método *isMaxScore* sobre un objeto de la clase *DBHelper*. Lo que hace es realizar una búsqueda en la tabla *statistic* de la base de datos interna del dispositivo. Busca la fila que sea de ese usuario y de esa asignatura y compara el campo *max_score* con la puntuación obtenida en la partida recientemente finalizada. Devuelve un *booleano* que indica si es la máxima puntuación, y también actualiza la máxima puntuación de la base de datos interna del dispositivo si procede. Si es la puntuación más alta alcanzada la aplicación accede al servicio *Web* para actualizar el valor de máxima puntuación de la tabla *mdl_eliza_classification* perteneciente a la base de datos de *moodle.* La aplicación accede al fichero *classification.php* mediante el método *getJSONFromUrl* de la clase *JSONParser*. A este método la aplicación le pasa como argumentos de entrada el identificador de usuario, el identificador de la asignatura y la puntuación obtenida. Así en recepción mediante la instrucción *UPDATE* actualiza el valor de puntuación máxima en la base de datos.

Tras este procedimiento la aplicación realiza modificaciones en la base de datos el dispositivo en términos de logros. A saber:

- Número máximo de preguntas consecutivas respondidas correctamente. Actualiza el logro si fuese necesario (oro, plata y bronce) y resetea el contador de aciertos a 0.

- Número de preguntas pasadas. Actualiza el logro si fuese necesario (oro, plata y bronce) y resetea el contador de aciertos a 0.

- Número de fallos en una partida. Actualiza el logro si fuese necesario (oro, plata y bronce) y resetea el contador de aciertos a 0.

El contador de puntuación es llevado a cabo por un *runOnUiThread*, que es un hilo a parte del principal de la aplicación con la excepcionalidad de que puede modificar la interfaz de la aplicación. De este modo, este hilo es invocado sobre un objeto de tipo *Timer*, que simplemente en cada instante es accedido y actualizamos la interfaz en cada acceso según programemos, en este caso incrementando una unidad.

En el caso de que el dispositivo no tuviese conexión a Internet y tuviese descargada una versión de los datos del juego aunque desactualizados, se informará al usuario de que los resultados obtenidos durante el juego no computarán en las estadísticas, y se procederá a jugar tal y como se ha descrito. En el caso también posible de que tuviese acceso a Internet pero no tuviese la información necesaria descargada no se iniciará el juego y se informará al usuario mediante *Toast*.

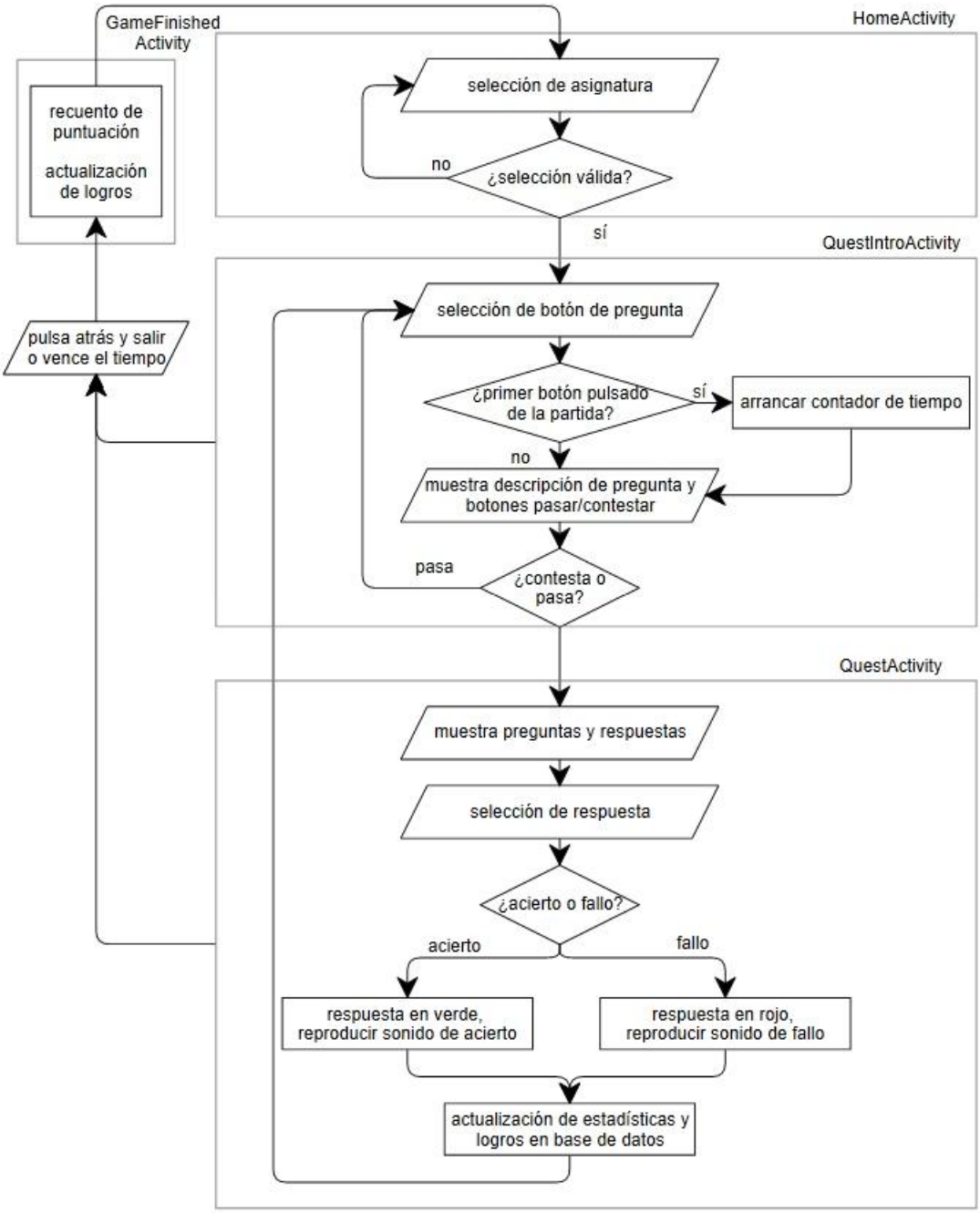

Diagrama 3: Comenzar a jugar

3. Acceder a las estadísticas

Para acceder a las estadísticas hemos de situarnos en la pantalla *home* y seleccionar la pestaña que se sitúa inmediatamente a la derecha. Esto se puede hace pulsando sobre ella en la barra de pestañas o mediante un gesto de arrastre sobre la pantalla hacia la izquierda.

Mencionar que esta no es una actividad a parte, es un *Fragment* de la actividad *HomeActivity*, la cual contiene todas las pestañas. Es decir, la clase *StatisticTab* hereda de *Fragment* y está incluida en la actividad *HomeActivity*. En ella visualizamos un *Spinner* que contiene todas nuestras asignaturas. Al seleccionar una de ellas se muestran las estadísticas correspondientes.

El método de creación de la interfaz visual que en una actividad común se conoce por *onCreate*, en el caso de los *Fragment* se llama *onCreateView* y a diferencia de *onCreate,* además de un parámetro tipo *Bundle* que contiene el estado de la actividad de algún uso previo de la misma, también obtiene como argumentos de entrada un objeto de tipo *LayoutInflater* sobre el que podemos crear cualquier tipo de vista en el *Fragment* y un objeto de tipo *ViewGroup* que contiene la vista padre a la que el *Fragment* está conectado. Dentro del método *onCreateView*, se crea una lista de *Strings* que contienen el encabezado del *Spinner* que no puede ser seleccionado como asignatura y contiene un mensaje que nos indica que hemos de seleccionar una asignatura, y posteriormente los nombres de las asignaturas. Tras esto, creamos un adaptador de *Strings* al que pasamos el estado de la actividad, una constante predefinida de Android que defina el modo de visualización de la lista del *Spinner* y la lista de *Strings* anteriormente llenada. Ahora obtenemos un objeto referencia a nuestro *Spinner* declarado en el XML, y sobre este objeto invocamos el método *setAdapter* al que le pasamos como argumento de entrada el adaptador que hemos creado.

El método de creación de la interfaz visual desactiva de la vista todos los elementos que contienen la información o los títulos de las estadísticas (de tipo *TextView)* ya que cuando la vista es creada por el método *onCreateView,* ninguna asignatura se ha seleccionado por el momento, por tanto, ninguna estadística es mostrada aún. Estas operaciones se vuelven a realizar cuando tras seleccionar una asignatura el usuario vuelve a seleccionar la primera opción del *Spinner* que indica que se seleccione una asignatura.

Seguidamente activamos el *listener* que será llamado por la aplicación cuando el usuario seleccione una asignatura. Para ello, sobre el objeto del *Spinner* ejecutamos el método *setOnItemSelectedListener* que recibe como argumento de entrada un objeto de tipo *OnItemSelectedListener* creado junto con un método *OnItemSelected* en su interior. Este último método, al seleccionar el usuario una asignatura, recibe como parámetros de entrada:

- Un objeto de tipo *AdapterView* que contiene el adaptador de la vista donde sucedió la selección.
- Un objeto de tipo *View* que contiene la vista con la cual el *AdapterView* fue seleccionado.
- Un entero que contiene la posición del elemento seleccionado sobre la lista del *Spinner.*

- Un entero que contiene el identificador del elemento de seleccionado de la lista del *Spinner.*

En su interior, mediante la posición del elemento seleccionado verificamos si se ha seleccionado una asignatura o si por contrario se ha seleccionado el primer elemento del *Spinner* que nos insta a seleccionar una asignatura. Si se ha seleccionado una asignatura, primeramente se rellenan los elementos de tipo *TextView* que van a mostrar la información mediante la información accedida a la base de datos local. Esta información es accedida mediante un objeto de la clase *DBHelper*, invocando al método *getStatistic* que recibe como parámetros de entrada el nombre de usuario y el identificador de la asignatura de la que deseamos visualizar las estadísticas y devuelve un array de *Strings* con los datos estadísticos de la asignatura requerida, a saber:

- Número de preguntas contestadas en total.
- Número de preguntas contestadas / número de preguntas falladas.
- Para cada uno de los 5 niveles de dificultad, preguntas contestadas y preguntas acertadas de cada uno.

Posteriormente a esto, hace visible todos los elementos de tipo *TextView* que se encontraban invisibles mientras ninguna asignatura había sido seleccionada. Esto se hace invocando al método *setVisibility* sobre cada uno de los objetos *TextView* de los que consta este *Fragment*. Junto a cada estadística porcentual se muestra una animación circular que va llenando un sector de la animación hasta llegar al porcentaje correspondiente. Esto se realiza de la siguiente forma: se hace una llamada al método *setCircle* que recibe como parámetros el *View* correspondiente a la animación, el *View* general de la actividad y el porcentaje que se desea mostrar. Este método, de forma resumida, fija los valores recibidos sobre otros métodos declarados dentro de la clase *CircleDisplay.* Una vez fijados los parámetros que determinan la animación, ejecuta el método *show* que a su vez inicia la animación sobre un objeto *ObjectAnimator* del que anteriormente hemos fijado los parámetros.

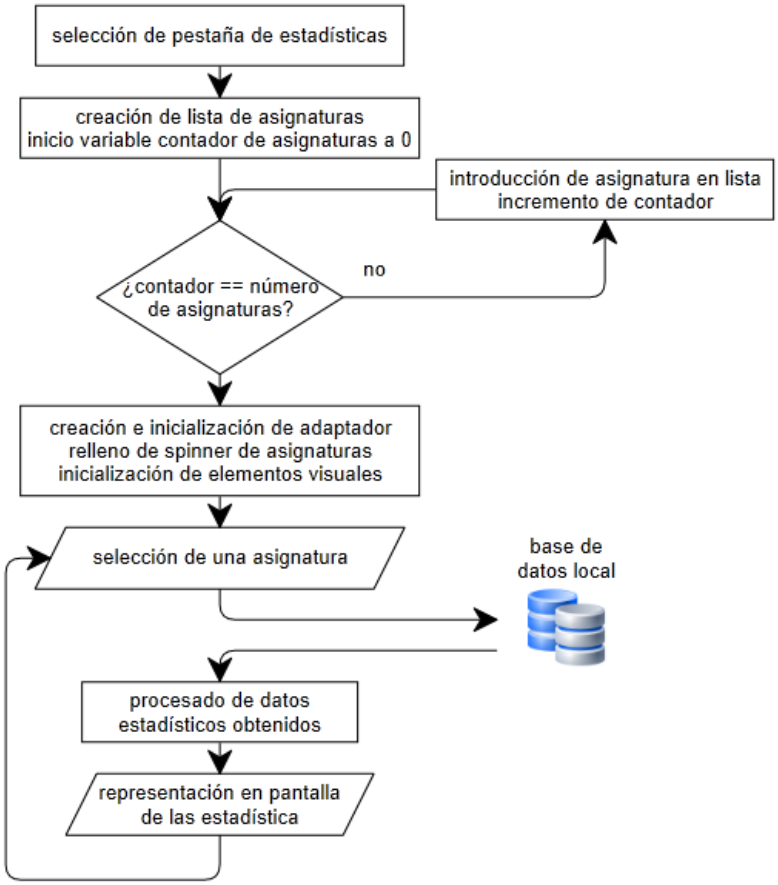

Diagrama 4: Actividad de estadísticas

4. Acceder a la clasificación

Para acceder a la clasificación hemos de situarnos en la pantalla *statistic* y seleccionar la pestaña que se sitúa inmediatamente a la derecha. Esto se puede hace pulsando sobre ella en la barra de pestañas o mediante un gesto de arrastre sobre la pantalla hacia la izquierda.

Mencionar que esta no es una actividad a parte, es un *Fragment* de la actividad *HomeActivity*, la cual contiene todas las pestañas. Es decir, la clase *ClassificationTab* hereda de *Fragment* y está incluida en la actividad *HomeActivity*. En ella visualizamos un *Spinner* que contiene todas nuestras asignaturas. Al seleccionar una de ellas se muestran la clasificación correspondiente a dicha asignatura.

El método de creación de la interfaz visual que en una actividad común se conoce por *onCreate*, en el caso de los *Fragment* se llama *onCreateView* y a diferencia de *onCreate,* además de un parámetro tipo *Bundle* que contiene el estado de la actividad de algún uso previo de la misma, también obtiene como argumentos de entrada un objeto de tipo *LayoutInflater* sobre el que podemos crear cualquier tipo de vista en el *Fragment* y un objeto de tipo *ViewGroup* que contiene la vista padre a la que el *Fragment* está conectado. Dentro del método *onCreateView*, se crea una lista de *Strings* que contienen el encabezado del *Spinner*, el cual no puede ser seleccionado como asignatura y contiene un mensaje que nos indica que hemos de seleccionar una asignatura, posteriormente los nombres de las asignaturas. Tras esto, creamos un adaptador de *Strings* al que pasamos el estado de la actividad, una constante predefinida de Android que defina el modo de visualización de la lista del *Spinner* y la lista de *Strings* anteriormente llenada. Ahora obtenemos un objeto referencia a nuestro *Spinner* declarado en el XML, y sobre este objeto invocamos el método *setAdapter* al que le pasamos como argumento de entrada el adaptador que hemos creado.

Seguidamente activamos el *listener* que será llamado por la aplicación cuando el usuario seleccione una asignatura. Para ello, sobre el objeto del *Spinner* ejecutamos el método *setOnItemSelectedListener* que recibe como argumento de entrada un objeto de tipo *OnItemSelectedListener* creado junto con un método *OnItemSelected* en su interior. Este último método, al seleccionar el usuario una asignatura, recibe como parámetros de entrada:

- Un objeto de tipo *AdapterView* que contiene el adaptador de la vista donde sucedió la selección.
- Un objeto de tipo *View* que contiene la vista con la cual el *AdapterView* fue seleccionado.
- Un entero que contiene la posición del elemento seleccionado sobre la lista del *Spinner.*
- Un entero que contiene el identificador del elemento de seleccionado de la lista del *Spinner.*

En su interior, mediante la posición del elemento seleccionado verificamos si se ha seleccionado una asignatura o si por contrario se ha seleccionado el primer elemento del *Spinner* que nos insta a seleccionar una asignatura. Si se ha seleccionado una asignatura, trabajaremos con una variable del tipo *String[][]* llamada *classification* que contiene las clasificaciones de las asignaturas.

Las clasificaciones han sido obtenidas por un *thread* creado desde *onCreateView* que ejecuta las operaciones en *background*. Lo que hace este método de *background* es acceder al servicio *Web get_classification.php* al que le pasa como argumento de entrada el identificador de la asignatura de la que quiere obtener la clasificación. El servicio *Web* realiza una búsqueda en la base de datos de *moodle* en la tabla *mdl_eliza_classification,* con el criterio de búsqueda del identificador de la asignatura y obteniendo los datos de modo descendente según el campo *max_score*. De este modo, ya recibiremos la clasificación ordenada, teniendo en la primera posición al usuario con más puntuación. Este servicio *Web* devuelve un array codificado como un objeto de tipo *JSON*, que en cada posición del array tiene un campo para el nombre de usuario y su puntuación. En recepción, mediante un bucle *for* se recorre todas las posiciones de cada clasificación nada más recibirla. Lo que se hace es introducir la información obtenida en la variable *String classification[][]*, con el siguiente formato. La primera dimensión del array divide las asignaturas, así en *classification[0]* se almacena la clasificación de la primera asignatura. Dentro de cada clasificación se distribuye la segunda dimensión del array en posiciones de dicha clasificación, así en *classification[0][0]* tendremos el contenido "nombre de usuario, puntuación" del usuario con más puntuación de la primera asignatura. También en cada iteración del bucle, se busca el nombre del usuario que está jugando con el fin de obtener qué puntuación tiene y actualizar el logro de mejor posición en la clasificación. Esto se hace mediante el método *maxClassificationPosition* de la clase *DBHelper,* la cual compara el valor obtenido de la clasificación con el que se tiene en la base de datos local, y actualiza la tabla *achievement* en consecuencia según proceda. Decir que en cada acceso al *Fragment* de la clasificación, este proceso se realizará para todas las asignaturas del usuario. De este modo aseguramos que la aplicación funcionará de forma fluida al visualizar las clasificaciones de las demás asignaturas, y sólo introducirá un pequeño retraso al principio.

En el mismo momento en el que se recibe la clasificación del servicio *Web* REST, ésta es almacenada en la base de datos local del dispositivo. Esta operación es llevada a cabo mediante un método de la clase *DBHelper* llamado *setLocalClassification* en el cual en cada llamada a la función se introducirá la información referente al nombre del usuario, su puntuación y su posición en la clasificación. Para ello, este método recibirá los parámetros anteriormente mencionados además del nombre y el identificador de la asignatura, ya que si un usuario tiene más de una asignatura la información debe ser distinguida para cada una de ellas.

Una vez analizada la obtención de la información referente a la variable *classification,* proseguimos con la explicación del *listener* asociado a la selección de una asignatura en el *Spinner*. Primeramente se verifica que exista algún usuario registrado en la clasificación. Posteriormente se procede a la creación de un de una lista de objetos de tipo *ClassificationModel*. La clase *ClassificationModel* posee variables globales en las que almacenar la información relativa a cada posición de cada clasificación, a saber:

- Entero que identifica la posición dentro de la clasificación. Esta variable puede ser modificada o accedida mediante los métodos de la clase *setPosition* y *getPosition*, que reciben y devuelven, respectivamente, un entero con la posición.

- *String* que identifica el nombre del usuario en la clasificación. Esta variable puede ser modificada o accedida mediante los métodos de la clase *setName* y *getName*, que reciben y devuelven, respectivamente, un *String* con el nombre del usuario.
- Entero que identifica la puntuación en la clasificación. Esta variable puede ser modificada o accedida mediante los métodos de la clase *setScore* y *getScore*, que reciben y devuelven, respectivamente, un entero con la puntuación.
- *Booleano* que identifica si al usuario de la aplicación le corresponde dicho objeto *ClassificationModel*. Esta variable puede ser modificada o accedida mediante los métodos de la clase *isMe* y *setIsMe*, que reciben y devuelven, respectivamente, un *booleano* con la información de si corresponde o no al usuario actual. Es usada para destacar de otro color nuestra posición en la clasificación.

Tras rellenar la información de un usuario de la clasificación mediante estos métodos, el objeto perteneciente al usuario se introduce como un elemento más de la lista mediante el método *add,* invocado sobre el objeto de dicha lista, y al que le se pasa como argumento de entrada el objeto *ClassificationModel* recientemente llenado.

Tras haber realizado este proceso tantas veces como posiciones haya en la clasificación de la asignatura que el usuario ha seleccionado, se define el adaptador. Al adaptador se le pasa con argumento de entrada un objeto del tipo *Context* que contiene el estado actual de la actividad, un entero que representa a un identificador interno de Android como indicación de qué tipo de vista predefinida de lista queremos mostrar y la lista anteriormente llenada. A continuación, obtenemos la referencia al objeto de tipo *listView* que queremos llenar para representar la clasificación sobre la pantalla y posteriormente sobre él invocamos al método *setAdapter* que recibe como argumento de entrada el adaptador creado, para mostrar la lista con los usuarios. En ese momento, la clase *MyCustomAdapter* es accedida mediante su método principal *getView*. Este método se encarga de la representación gráfica de la clasificación sobre la pantalla. Sobre todo de aspectos visuales como, indicar que un usuario no ha jugado cuando su puntuación es la creada por defecto “-99999” o marcar con color amarillo el nombre de usuario y la puntuación del usuario de la aplicación.

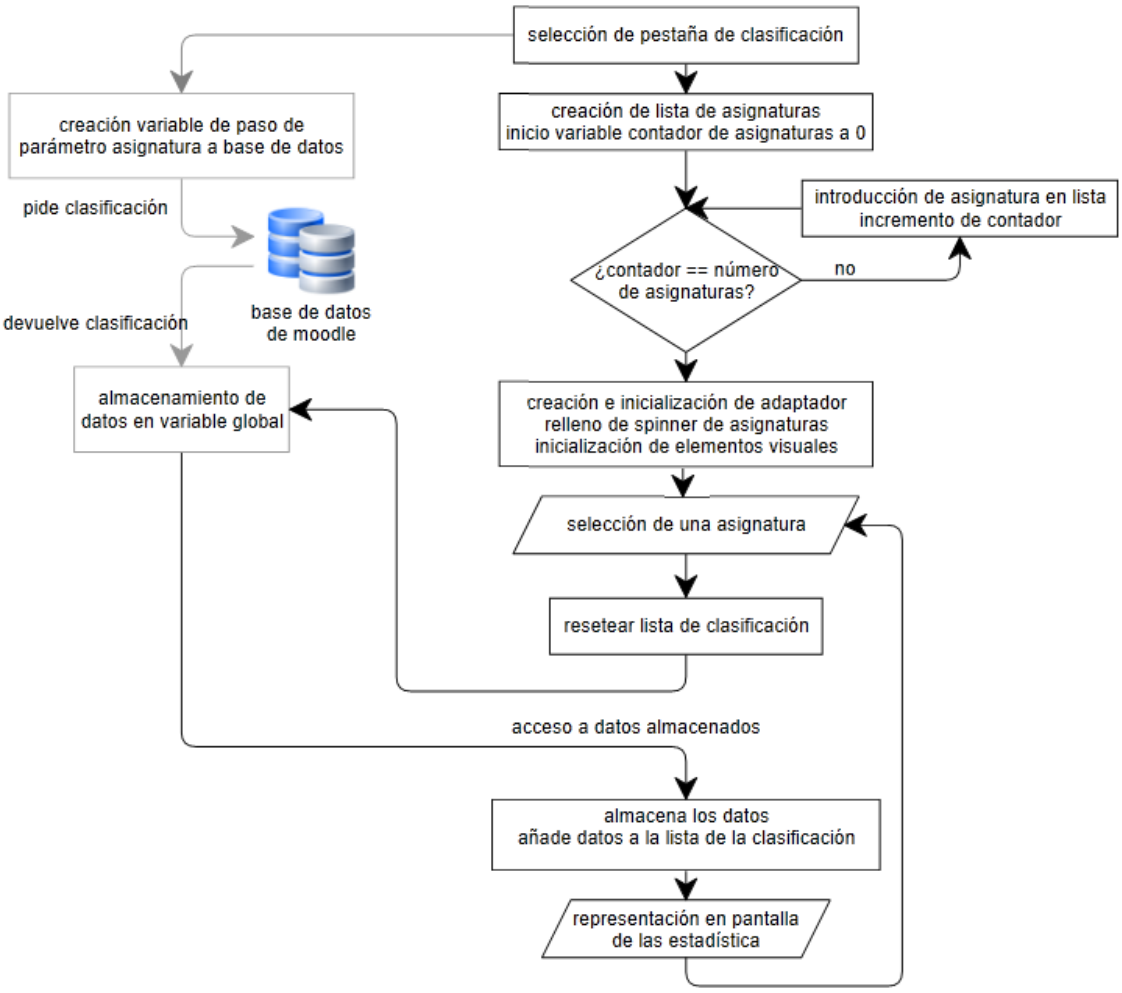

Diagrama 5: Actividad de clasificación

5. Acceder a los logros

Para acceder a la los logros hemos de situarnos en la pantalla *classification* y seleccionar la pestaña que se sitúa inmediatamente a la derecha. Esto se puede hace pulsando sobre ella en la barra de pestañas o mediante un gesto de arrastre sobre la pantalla hacia la izquierda.

Mencionar que esta no es una actividad a parte, es un *Fragment* de la actividad *HomeActivity*, la cual contiene todas las pestañas. Es decir, la clase *AchievementTab* hereda de *Fragment* y está incluida en la actividad *HomeActivity*. En ella visualizamos un *Spinner* que contiene todas nuestras asignaturas. Al seleccionar una de ellas se muestran los logros correspondientes.

El método de creación de la interfaz visual que en una actividad común se conoce por *onCreate*, en el caso de los *Fragment* se llama *onCreateView* y a diferencia de *onCreate,* además de un parámetro tipo *Bundle* que contiene el estado de la actividad de algún uso previo de la misma, también obtiene como argumentos de entrada un objeto de tipo *LayoutInflater* sobre el que podemos crear cualquier tipo de vista en el *Fragment* y un objeto de tipo *ViewGroup* que contiene la vista padre a la que el *Fragment* está conectado. Dentro del método *onCreateView*, se crea una lista de *Strings* que contienen el encabezado del *Spinner*, el cual no puede ser seleccionado como asignatura y contiene un mensaje que nos indica que hemos de seleccionar una asignatura, posteriormente los nombres de las asignaturas. Tras esto, creamos un adaptador de *Strings* al que pasamos el estado de la actividad, una constante predefinida de Android que defina el modo de visualización de la lista del *Spinner* y la lista de *Strings* anteriormente llenada. Ahora obtenemos un objeto referencia a nuestro *Spinner* declarado en el XML, y sobre este objeto invocamos el método *setAdapter* al que le pasamos como argumento de entrada el adaptador que hemos creado.

Seguidamente activamos el *listener* que será llamado por la aplicación cuando el usuario seleccione una asignatura. Para ello, sobre el objeto del *Spinner* ejecutamos el método *setOnItemSelectedListener* que recibe como argumento de entrada un objeto de tipo *OnItemSelectedListener* creado junto con un método *OnItemSelected* en su interior. Este último método, al seleccionar el usuario una asignatura, recibe como parámetros de entrada:

- Un objeto de tipo *AdapterView* que contiene el adaptador de la vista donde sucedió la selección.
- Un objeto de tipo *View* que contiene la vista con la cual el *AdapterView* fue seleccionado.
- Un entero que contiene la posición del elemento seleccionado sobre la lista del *Spinner.*
- Un entero que contiene el identificador del elemento de seleccionado de la lista del *Spinner.*

En su interior, mediante la posición del elemento seleccionado verificamos si se ha seleccionado una asignatura o si por contrario se ha seleccionado el primer elemento del *Spinner* que nos insta a seleccionar una asignatura. Si se ha seleccionado una asignatura, la aplicación llamará al método *getAchievement* desde un objeto de la clase *DBHelper*, al que le

pasará como argumento de entrada el nombre de usuario y el entero que identifica a la asignatura que se ha seleccionado. Este método realiza una búsqueda de los campos de la tabla *achievement* que contienen la información relativa a la consecución de logros del usuario. En ellas está la información que nos dice si el usuario ha logrado oro, plata, bronce o aún no tiene medalla, en cada uno de los logros. Esto es representado en cada una de las variables con un 4 si aún no se logró medalla, un 3 si se logró la de bronce, un 2 si se logró la de plata o un 1 si se logró la de oro. Los logros que se acceden son los siguientes:

- Número de preguntas acertadas consecutivamente. Representado por el campo *nRightConsecutive* de la tabla *achievement.* Las medallas se distribuyen de la siguiente manera:
    - Oro: 15 preguntas acertadas consecutivamente
    - Plata: 10 preguntas acertadas consecutivamente
    - Bronce: 5 preguntas acertadas consecutivamente
- Puntuación alcanzada. Representado por el campo *scoreReached* de la tabla *achievement.* Las medallas se distribuyen de la siguiente manera:
    - Oro: 2000 puntos logrados en una partida
    - Plata: 1500 puntos logrados en una partida
    - Bronce: 1000 puntos logrados en una partida
- Partida limpia. Representado por el campo *cleanGame* de la tabla *achievement.* Las medallas se distribuyen de la siguiente manera:
    - Oro: 0 preguntas falladas en una partida
    - Plata: 1 preguntas falladas en una partida
    - Bronce: 2 preguntas falladas en una partida
- Partida de riesgo. Representado por el campo *riskGame* de la tabla *achievement.* Las medallas se distribuyen de la siguiente manera:
    - Oro: 0 preguntas pasadas en una partida
    - Plata: 1 preguntas pasadas en una partida
    - Bronce: 2 preguntas pasadas en una partida
- Superación de máxima puntuación. Representado por el campo *overcomingScore* de la tabla *achievement.* Las medallas se distribuyen de la siguiente manera:
    - Oro: 5 veces se ha superado la máxima puntuación
    - Plata: 3 veces se ha superado la máxima puntuación
    - Bronce: 2 veces se ha superado la máxima puntuación
- Clasificación. Representado por el campo *classificationPodium* de la tabla *achievement.* Las medallas se distribuyen de la siguiente manera:
    - Oro: ser primero en la clasificación de la asignatura
    - Plata: ser segundo en la clasificación de la asignatura
    - Bronce: ser tercero en la clasificación de la asignatura

Al contener estas variables enteros, se almacenan en una variable *int[]* y esta variable es devuelta por el método *getAchievement.* Posteriormente, se recoge en un array los nombres de los retos, y la explicación de cada uno de ellos. Estos datos se encuentran almacenados como arrays de *Strings* en el fichero *strings.xml* de los *resources* de la aplicación.

De este modo, mediante *getResources().getStringArray()* y pasándole al último método el identificador, obtenemos estos datos.

Seguidamente se procede a la creación de un de una lista de objetos de tipo *AchievementModel*. La clase *AchievementModel* posee variables globales en las que almacenar la información relativa a cada posición de cada clasificación, a saber:

- Entero que identifica qué grado de consecución del logro hemos alcanzado. De este modo:
    - Contiene un 1 si hemos logrado el oro.
    - Contiene un 2 si hemos logrado la plata.
    - Contiene un 3 si hemos logrado el bronce.
    - Contiene un 4 si aún o hemos logrado medalla.

  Esta variable puede ser modificada o accedida mediante los métodos de la clase *setPosition* y *getPosition*, que reciben y devuelven, respectivamente, un *String* con el nombre del logro.
- *String* que identifica el nombre del logro. Es un nombre genérico con el que a veces es probable que no identifiquemos exactamente de qué trata ese logro, por ello, se ha incluido una descripción del mismo. Esta variable puede ser modificada o accedida mediante los métodos de la clase *setName* y *getName*, que reciben y devuelven, respectivamente, un *String* con el nombre del logro.
- *String* que describe el logro y explica los requisitos que ha de alcanzar el usuario para la consecución de cada una de las medallas. Se especifica perfectamente de qué consiste el logro, por tanto concreta el nombre, que en muchas ocasiones podría ser ambiguo. Esta variable puede ser modificada o accedida mediante los métodos de la clase *setDescription* y *getDescription*, que reciben y devuelven, respectivamente, un *String* con la descripción del logro.
- Imagen que contiene la imagen de la medalla que se mostrará en cada logro según se haya conseguido alguna medalla. En la clase *AchievementModel* no posee métodos que la usen, pero posteriormente, cuando se prepare la interfaz de usuario para ser mostrada, será accedida. Estas imágenes están almacenadas en la carpeta *drawable*.

Tras rellenar la información de los logros de un usuario mediante estos métodos, el objeto perteneciente al usuario se introduce como un elemento más de la lista mediante el método *add.* Este método es invocado sobre el objeto de dicha lista, y al que le se pasa como argumento de entrada el objeto *AchievementModel* recientemente llenado.

Tras haber realizado este proceso tantas veces como logros haya, se define el adaptador. Al adaptador se le pasa con argumento de entrada un objeto del tipo *Context* que contiene el estado actual de la actividad, un entero que representa a un identificador interno de Android como indicación de qué tipo de vista predefinida de lista queremos mostrar y la lista anteriormente llenada.

A continuación, obtenemos la referencia al objeto de tipo *listView* que queremos llenar para representar la clasificación sobre la pantalla y posteriormente sobre él invocamos al método *setAdapter* que recibe como argumento de entrada el adaptador creado, para

mostrar la lista con los usuarios. En ese momento, la clase *MyCustomAchievementAdapter* es accedida mediante su método principal *getView*. Este método se encarga de la representación gráfica de la lista de logros sobre la pantalla. Primeramente obtiene una referencia a todos los elementos de la interfaz de la pantalla que van a ser modificados y que están declarados en los ficheros XML. Véase, el nombre y la descripción del logro y la imagen de la medalla. Obtenemos de la lista de logros el situado en la posición que vamos a representar y en función de su posición, accedida mediante el método *getPosition*, fijamos el contenido de la imagen. También fijamos el contenido de los *TextViews* del nombre y la descripción del logro, obteniendo dicha información invocando sobre el elemento de la lista el método *getName* y *getDescription* respectivamente. Este último proceso descrito se repite para todos los logros que tengamos.

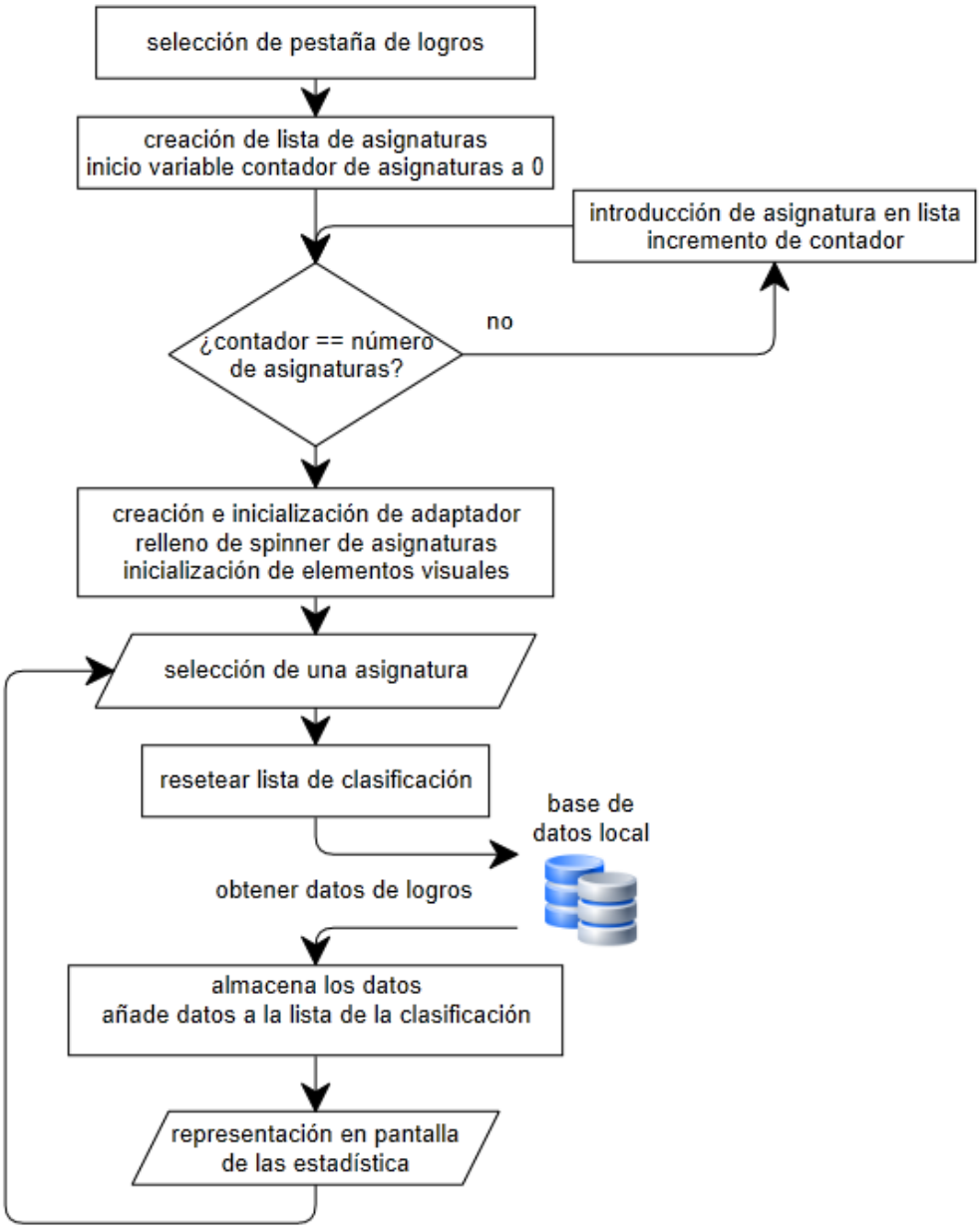

Diagrama 6: Actividad de logros

6. Visión general de la interacción entre actividades

Como resumen de la interacción entre actividades mostradas mencionamos que al arrancar la aplicación siempre se muestra la actividad *SplashScreen*. En ella se verifica si en el dispositivo hay un usuario marcado como *lastUser* que haya usado anteriormente el juego, en tal caso se pasa a *home* y en caso contrario a la actividad *MainActivity*.

En *MainActivity* podremos realizar un registro de cero, o registrarnos con nuestro usuario y contraseña. En el caso de que no estemos registrados pasaremos por el formulario de registro y en caso contrario nos saltaremos este paso y seleccionaremos únicamente nuestro curso y asignaturas.

Una vez en *home* podemos movernos entre pestañas usando las funcionalidades de cada una. Debido a la extensión del apartado jugar se hace referencia en el diagrama al diagrama representado anteriormente.

Diagrama 7: Resumen relación entre actividades

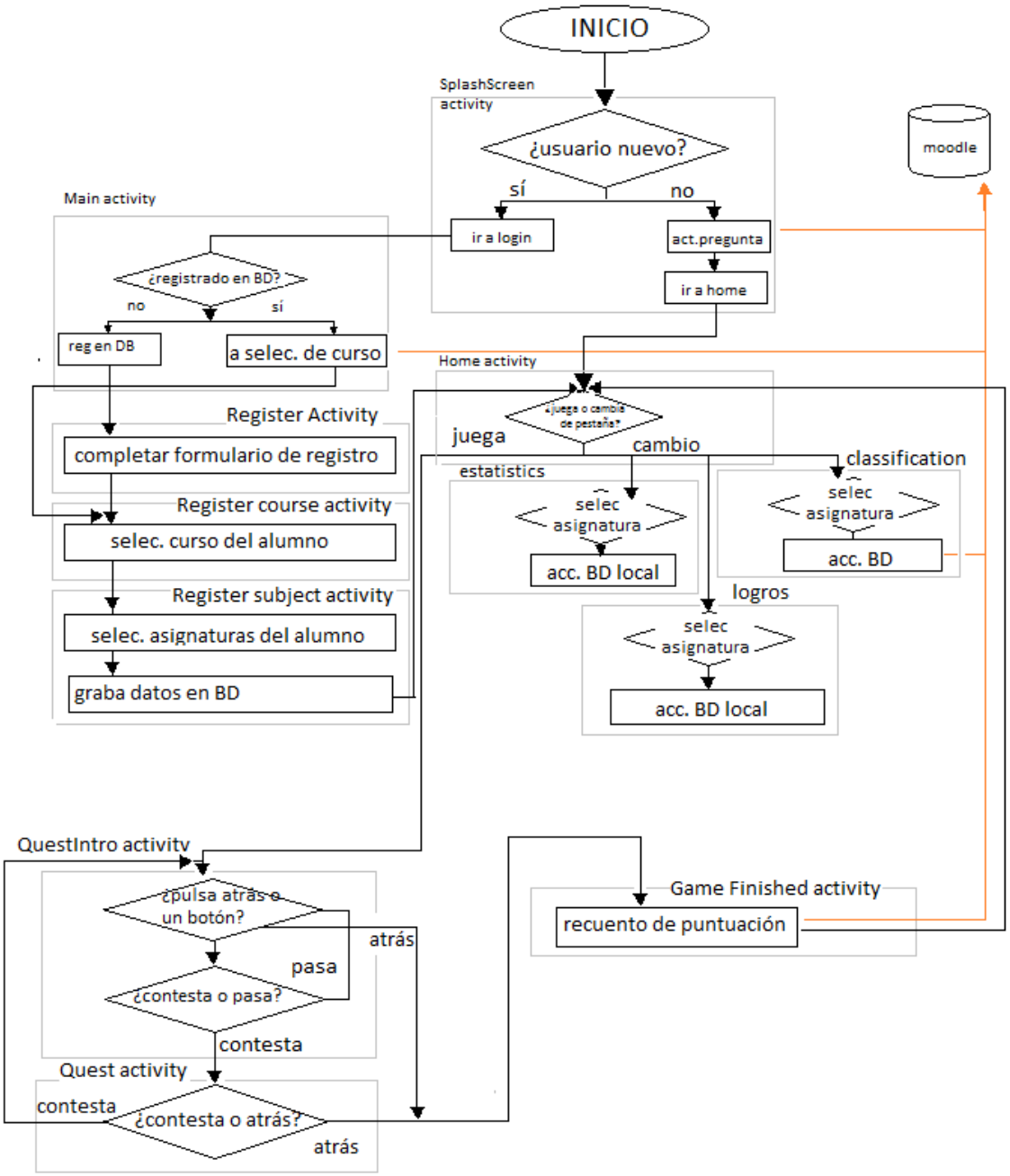

Diagrama 8: Interacción detallada entre actividades

7. Salir de la aplicación

Para salir de la aplicación hemos de estar fuera de dos procesos concretos: fuera de un juego y fuera de un proceso de registro. Así, si estuviésemos en un proceso de registro y pulsásemos la tecla atrás, lo único que conseguiríamos sería volver a la actividad anterior. Esto sí nos puede servir, debido a que podemos pulsar atrás y movernos entre actividades hasta

llegar a *MainActivity*. En esa actividad, al pulsar atrás, el método *onKeyDown* es ejecutado por el sistema y recibe 2 parámetros de entrada identificando el tipo de evento y qué tecla se pulsó. De este modo podemos identificar que se pulsó la tecla atrás. En ese momento la aplicación muestra un *dialog* que nos pregunta si queremos permanecer en la aplicación o salir de ella. Al pulsar salir, la aplicación ejecuta *finish* y salimos de la aplicación.

En el caso de que estuviésemos en medio de un juego, primero deberemos de salir de dicho juego del mismo modo que salimos de la aplicación. En este caso, nos podemos encontrar en la actividad *QuestIntroActivity* o en *QuestActivity,* para salir de aquí hemos de finalizar la partida. Si nos encontramos en *GameFinishedActivity* nos veremos obligados a esperar a que termine el proceso de recuento que se ejecuta en él que apenas dura unos segundos, e inmediatamente seremos redireccionados a *HomeActivity,* proceso que se detalla más adelante. Para finalizar la partida en curso desde cualquiera de las 2 actividades mencionadas hemos de pulsar la tecla atrás. En ese momento el método *onKeyDown* es ejecutado por el sistema y recibe dos parámetros de entrada identificando el tipo de evento y qué tecla se pulsó. De este modo podemos identificar que se pulsó la tecla atrás. En ese momento la aplicación muestra un *dialog* que nos pregunta si queremos permanecer en el juego o por contrario finalizarlo con la puntuación alcanzada hasta ese momento. Tras haber finalizado el juego la aplicación nos conduce a la actividad *GameFinishedActivity*. Visualmente lo que el usuario percibe es un recuento de puntuación. En el centro de la pantalla el número va contando desde 0 hasta la puntuación alcanzada.

Al finalizar el recuento de la puntuación somos dirigidos a la pantalla *home*, aquí al pulsar atrás, el método *onKeyDown* es ejecutado por el sistema y recibe dos parámetros de entrada identificando el tipo de evento y qué tecla se pulsó. De este modo podemos identificar que se pulsó la tecla atrás. En ese momento la aplicación muestra un *dialog* que nos pregunta si queremos permanecer en la aplicación, salir de ella o cerrar la sesión. Al pulsar salir, la aplicación ejecuta *finish* y salimos de la aplicación.

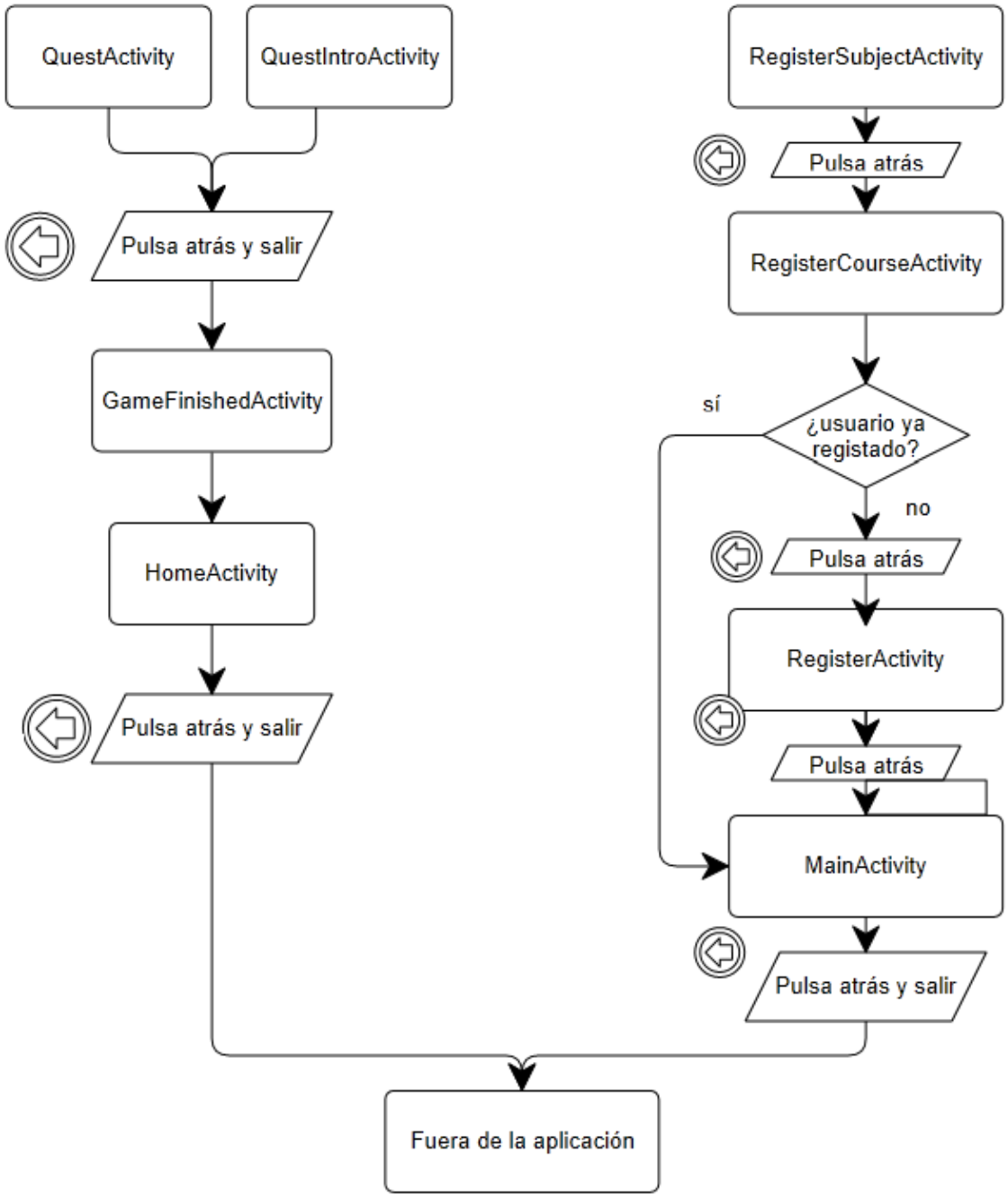

Diagrama 9: Salir de la aplicación

Manual de uso de la aplicación

Al arrancar la aplicación se muestra un *splash* unos segundos en el que aparece el nombre de la aplicación así como los logotipos de la Universidad de Valladolid y el Grupo de Telemática Industrial, para el que he desarrollado la aplicación.

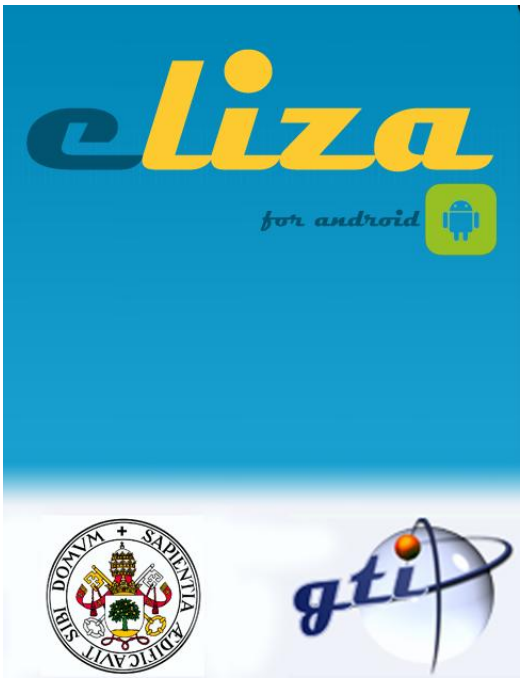

Ilustración 41: Pantalla SplashScreenActivity

Durante estos segundos la aplicación realiza una serie de operaciones internas. Si no existe un usuario marcado como *lastUser*, tras unos segundos el *splash* desaparece y entramos en una pantalla en la que se muestra:

- Parte superior de la pantalla:
  - Campo de introducción de texto de nombre de usuario.
  - Campo de introducción de texto de contraseña.
  - Botón de procesado de los datos introducidos.
- Parte inferior de la pantalla:
  - Botón de acceso para usuarios no registrados en la plataforma de *moodle* del GTI.

Usaremos el método de acceso de la parte superior de la pantalla si somos un usuario registrado en el sistema de *moodle* del GTI. En ese caso, la aplicación distinguirá si somos un usuario registrado en la aplicación instalada en el dispositivo y de lo contrario, obtiene los datos necesarios de la base de datos para posteriormente introducirlos en la base de datos local de la aplicación. También se depura la veracidad de los datos: si el usuario existe en el sistema *moodle* y si la contraseña introducida es correcta, de lo contrario se le informará al

usuario por pantalla mediante *toasts*. Cuando el acceso sea verificado como correcto el usuario será dirigido a la pantalla *home*.

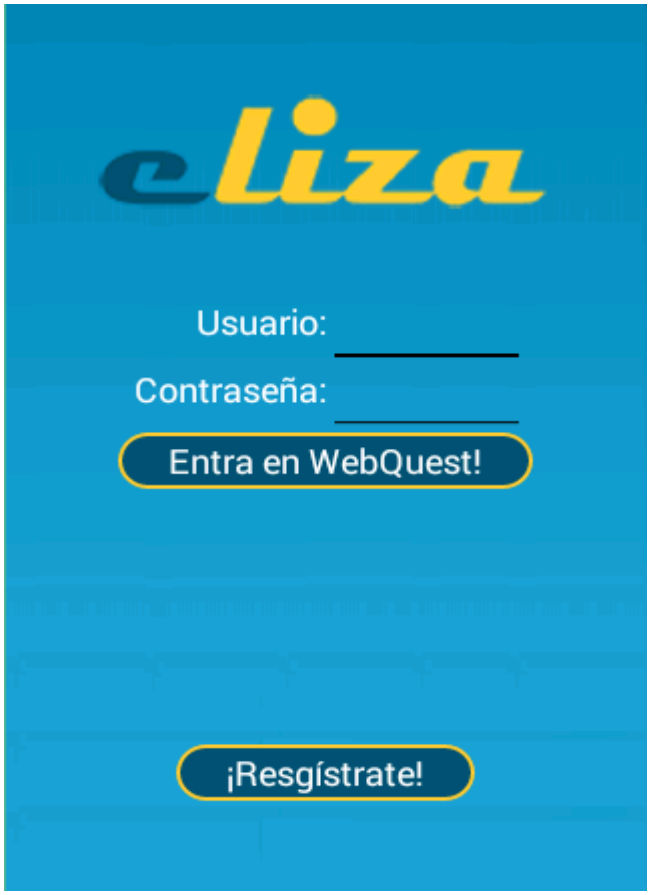

Ilustración 42: Pantalla MainActivity

Si no somos un usuario registrado en la plataforma *Moodle* del GTI tendremos que hacer click en el botón inferior de la pantalla. Que nos dirigirá a la pantalla de registro para nuevos usuarios.

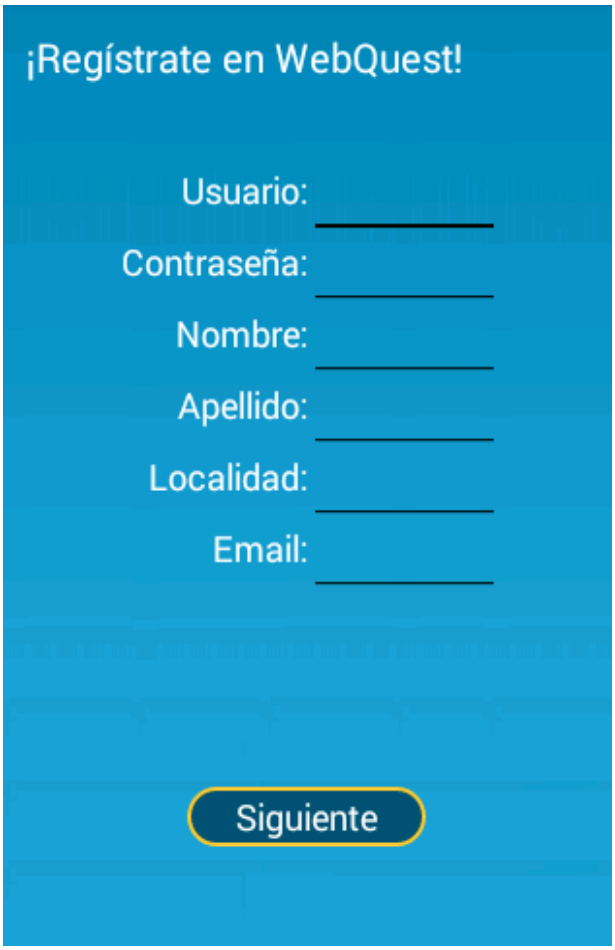

Ilustración 43: Pantalla RegisterActivity

Dentro del registro se nos solicitan los datos:

- Nombre de usuario
- Contraseña
- Nombre
- Apellidos
- Ciudad
- Correo electrónico

Los datos serán verificados y todos han de ser completados para continuar nuestro registro. Entre ellos la longitud de la contraseña y la verificación de existencia de mayúsculas y caracteres especiales en ella. Los requerimientos de contraseña son los siguientes:

- 6 caracteres de longitud mínima
- Al menos 1 carácter especial
- Al menos 1 número
- Al menos 1 letra mayúscula

Estos datos aún no serán almacenados en la base de datos, se espera a completar del todo el proceso de registro para poder hacer el registro en base de datos de forma atómica. Por si por cualquier motivo, el registro no fuese completado totalmente.

Una vez hayamos rellenado los datos requeridos pulsamos el botón inferior que nos conduce a otra pantalla en la que se nos muestran las titulaciones que poseen un módulo *eLiza* con preguntas. Hemos de elegir uno y darle al botón inferior para continuar con el registro.

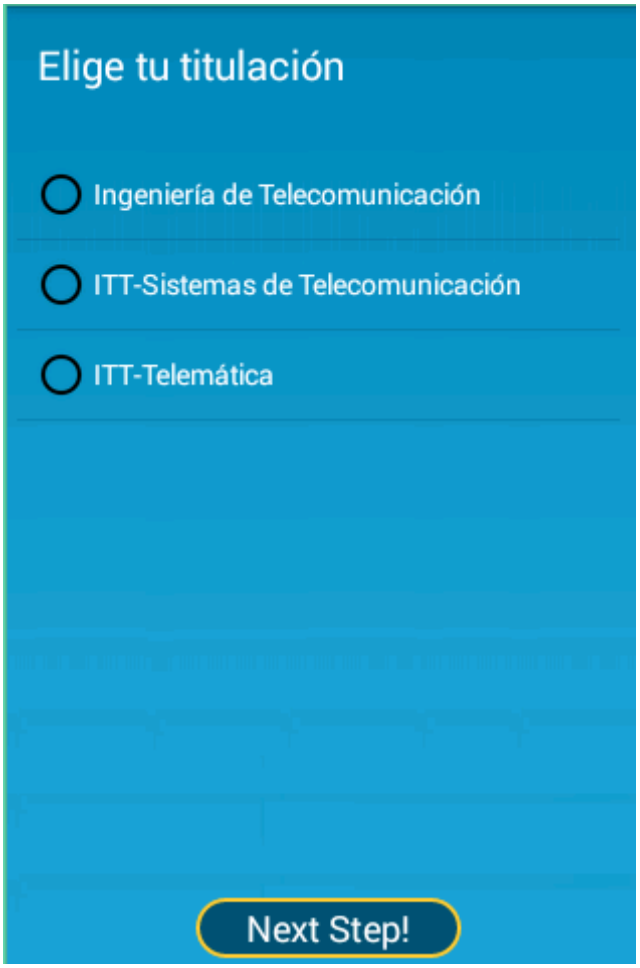

Ilustración 44: Pantalla RegisterCourseActivity

En la siguiente pantalla se muestran las asignaturas que poseen un módulo *eLiza*. Hemos de elegir nuestras asignaturas y darle a finalizar. De la misma forma que se procedió anteriormente en el caso de los cursos, esta vez se ha hecho en el caso de las asignaturas del curso seleccionado.

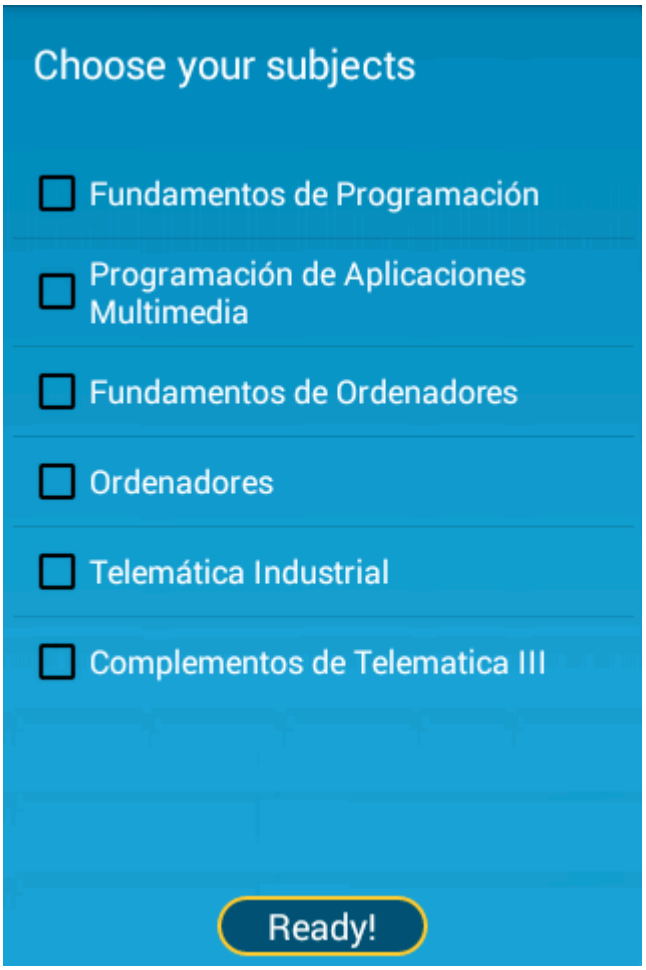

Ilustración 45: Pantalla RegisterSubjectsActivity

Ahora nos encontramos en el distribuidor de la aplicación, en el que tenemos 4 pestañas entre las que nos podemos mover:

- *Home*: Contiene el acceso al juego y un breve tutorial desplegable.
- Estadísticas: Contiene las estadísticas del alumno en cada asignatura.
- Clasificación: Contiene la clasificación de cada una de las asignaturas del alumno.
- Logros: Contiene la lista de logros del juego y muestra el progreso del alumno en cada uno de ellos.

Ilustración 46: Pestañas de la pantalla HomeActivity

La pestaña Home: En ella podemos seleccionar una asignatura y proceder a jugar con ella. Muestra un *Spinner* que ha sido llenado de forma dinámica con los nombres de las asignaturas en las que el alumno está matriculado. Esta ventana también posee un tutorial rápido que podemos mostrar y ocultar y que de una forma muy visual nos enseña la dinámica del juego. De este modo nos aseguramos que cualquiera que instale la aplicación sin tener conocimientos previos, pueda hacer un uso correcto del juego.

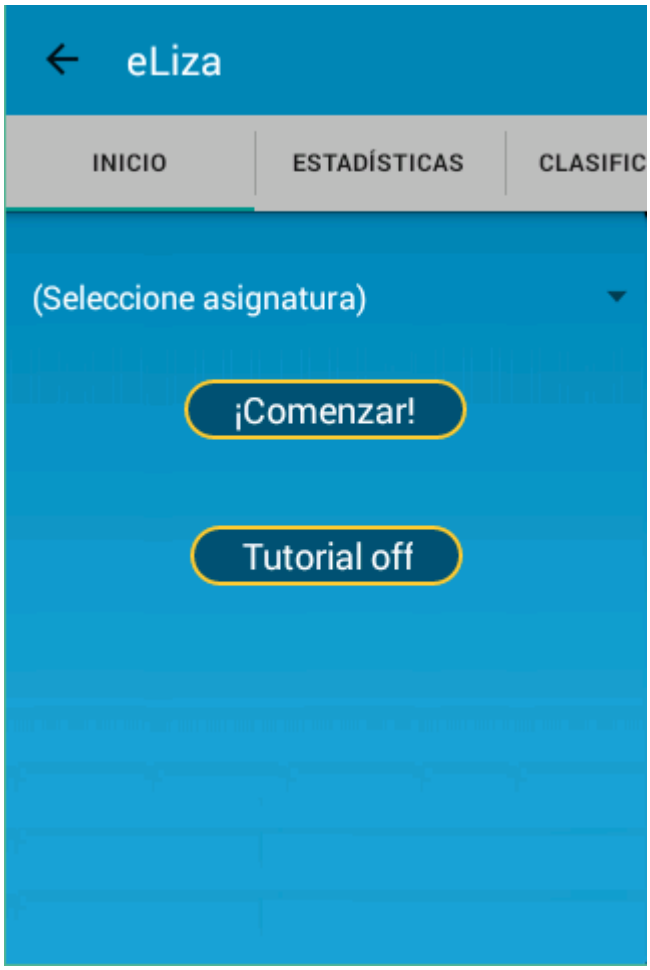

Ilustración 47: Pantalla HomeActivity: Inicio

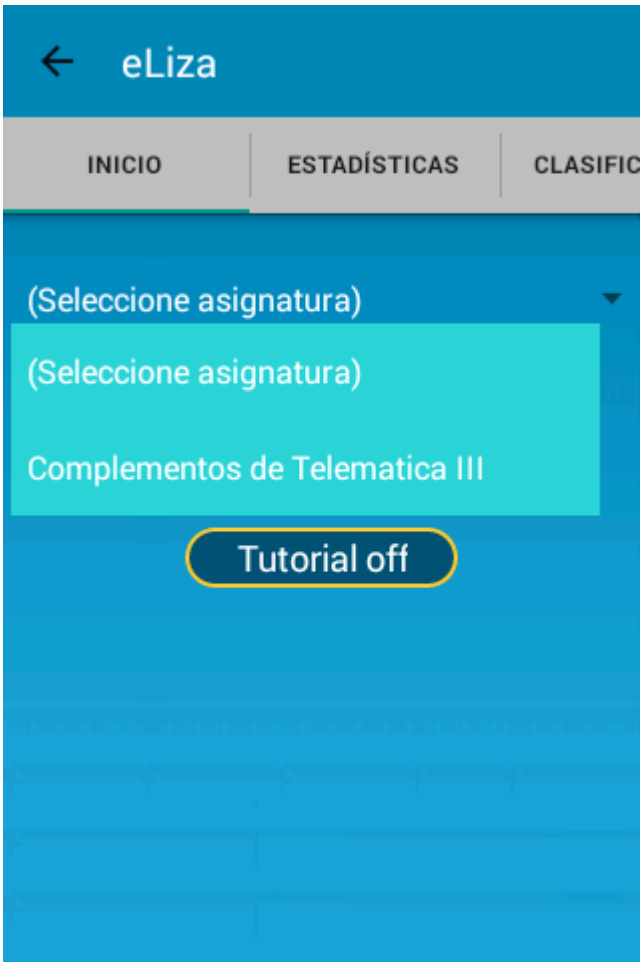

Ilustración 48: Pantalla HomeActivity: Despliegue de Spinner

La pestaña estadísticas: En ella vemos los porcentajes genéricos o agrupados por dificultad de nuestras preguntas acertadas o falladas. Primeramente hemos de seleccionar de qué asignatura queremos visualizar las estadísticas. Los datos que nos muestra la aplicación sobre nuestras estadísticas para cada una de las asignaturas son los siguientes:

- Número de preguntas contestadas en total.
- Número de preguntas contestadas / número de preguntas falladas.
- Para cada uno de los 5 niveles de dificultad, el porcentaje de acierto de preguntas.

Ilustración 49: Pantalla HomeActivity: Estadísticas

La pestaña clasificación: Al igual que la pestaña estadísticas está dividida por asignaturas. Seleccionando cualquieras de ellas veremos la clasificación que muestra dos columnas: el nombre de usuario y la puntuación en un *ranking*.

Ilustración 50: Pantalla HomeActivity: Clasificación

La pestaña de logros: Como las dos anteriores está dividida por asignaturas. Se trata de conseguir la medalla de oro en todos los retos que nos propone la aplicación. Estos retos son los siguientes:

- Número de preguntas acertadas consecutivamente. Las medallas se distribuyen de la siguiente manera:
  - Oro: 15 preguntas acertadas consecutivamente
  - Plata: 10 preguntas acertadas consecutivamente
  - Bronce: 5 preguntas acertadas consecutivamente
- Puntuación alcanzada: Las medallas se distribuyen de la siguiente manera:
  - Oro: 2000 puntos logrados en una partida
  - Plata: 1500 puntos logrados en una partida
  - Bronce: 1000 puntos logrados en una partida
- Partida limpia: Las medallas se distribuyen de la siguiente manera:
  - Oro: 0 preguntas falladas en una partida
  - Plata: 1 preguntas falladas en una partida
  - Bronce: 2 preguntas falladas en una partida
- Partida de riesgo: Las medallas se distribuyen de la siguiente manera:

    - Oro: 0 preguntas pasadas en una partida
    - Plata: 1 preguntas pasadas en una partida
    - Bronce: 2 preguntas pasadas en una partida
- Superación de máxima puntuación: Las medallas se distribuyen de la siguiente manera:
    - Oro: 5 veces se ha superado la máxima puntuación
    - Plata: 3 veces se ha superado la máxima puntuación
    - Bronce: 2 veces se ha superado la máxima puntuación
- Clasificación: Las medallas se distribuyen de la siguiente manera:
    - Oro: ser primero en la clasificación de la asignatura
    - Plata: ser segundo en la clasificación de la asignatura
    - Bronce: ser tercero en la clasificación de la asignatura

Sobre la pantalla se muestra el *spinner* de selección de asignaturas en la parte superior de la pantalla. Debajo se muestra la lista de los logros. En cada elemento de la lista se muestra el nombre del reto, en el centro el dibujo de la medalla obtenida y en la parte inferior de cada elemento la explicación de cada una de las medallas del logro. En este caso, la información relativa a los logros se procesa de forma local. Conforme se juega, la información relativa a los logros es actualizada.

Ilustración 51: Pantalla HomeActivity: Logros

Para acceder a jugar seleccionamos una asignatura en la pantalla *home* y procedemos a jugar. Inmediatamente acedemos a una pantalla en la que visualizamos en la parte superior el tiempo de juego restante (que hasta que no pulsemos sobre una pregunta no empezará) y la puntuación conseguida (inicialmente a 0). En la parte central veremos una columna por cada categoría de preguntas. Las categorías de juego indican los temas sobre los que tratarán las preguntas. De este modo el usuario puede valorar sus puntos fuertes o débiles en cada una de las categorías que se evalúen y elegir cuál contestar en consecuencia.

Cada botón de los 5 que posee esta columna (cada una de las categorías de juego) está titulado con un número que identifica la puntuación que posee esa pregunta. De modo que si la pregunta es acertada esa puntuación se sumará a nuestro marcador, pero si es fallada se restará.

Al pulsar sobre una dificultad de una categoría (un botón), en la parte inferior aparece la descripción de la pregunta, es decir, el tema del que trata. Y dos botones: para pasar de pregunta o para contestarla. Antes de seleccionar cualquier otro botón estamos obligados a decidir si contestamos o pasamos la pregunta seleccionada. Si pasamos podemos repetir el proceso.

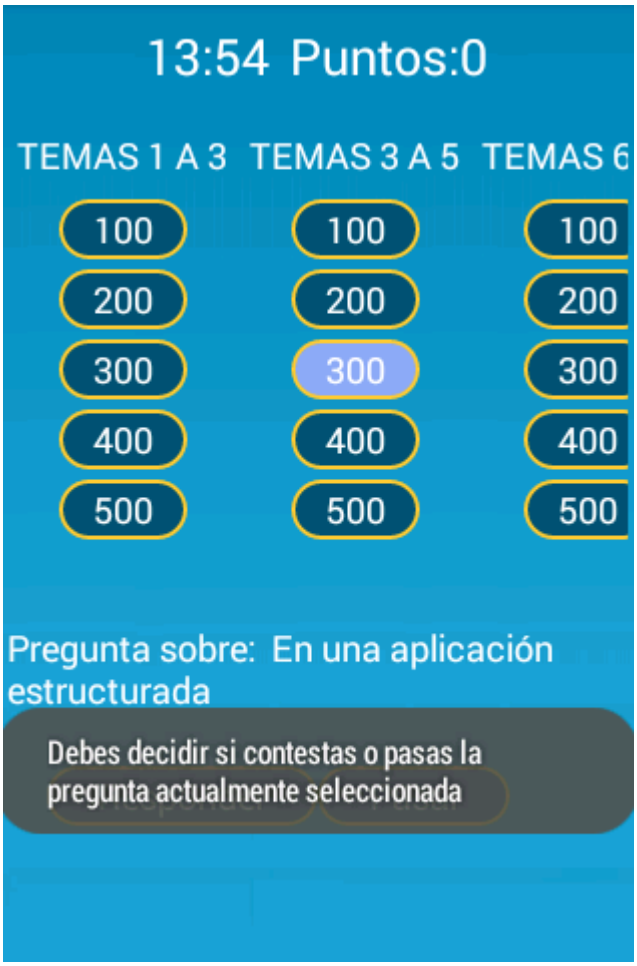

Ilustración 52: Pantalla QuestIntroActivity: Contestar o pasar

Si pulsamos el botón de contestar la pregunta entraremos a otra pantalla en la que se mantiene en la parte superior el tiempo restante y la puntuación. A la derecha de ésta aparece un botón que al pulsarlo nos muestra el enunciado de la pregunta en pantalla completa mediante un *Dialog*. Debajo de esto, vemos la pregunta, y debajo de esta las posibles respuestas. Al seleccionar una pulsamos el botón inferior para verificar si esta es correcta o no.

En cualquier momento podemos salir del juego pulsando el botón atrás. Se nos abrirá un diálogo en el que deberemos decidir si seguimos jugando o finalizamos el juego con la puntuación obtenida hasta ese momento.

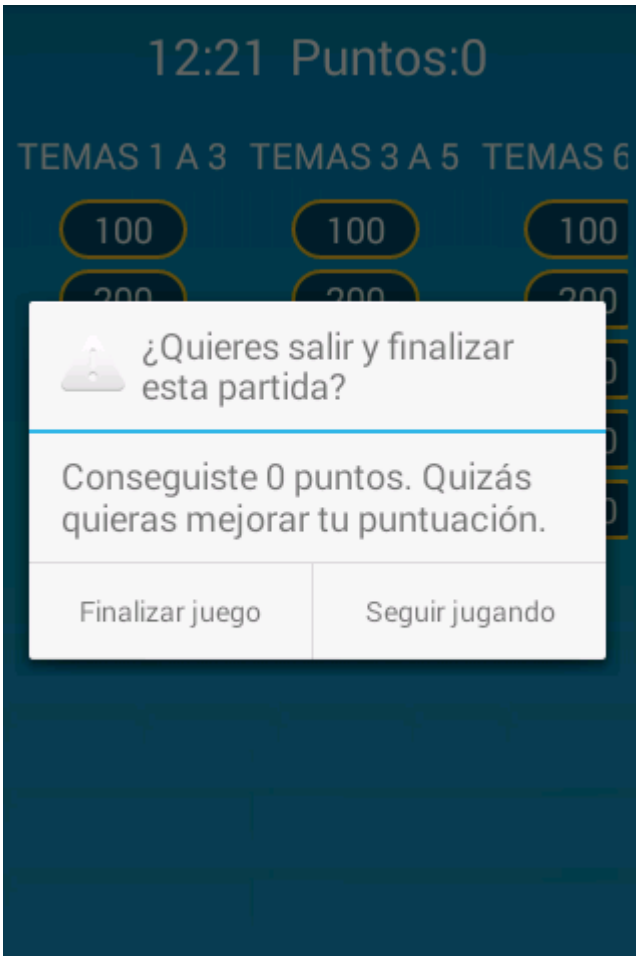

Ilustración 53: Diálogo de salida de la partida

Al finalizar el juego ya sea porque lo hemos finalizado manualmente, o porque el tiempo se ha terminado, entramos en una pantalla en la que se hace recuento de la puntuación que hemos obtenido. Segundos después la aplicación nos devuelve a la pantalla *home*.

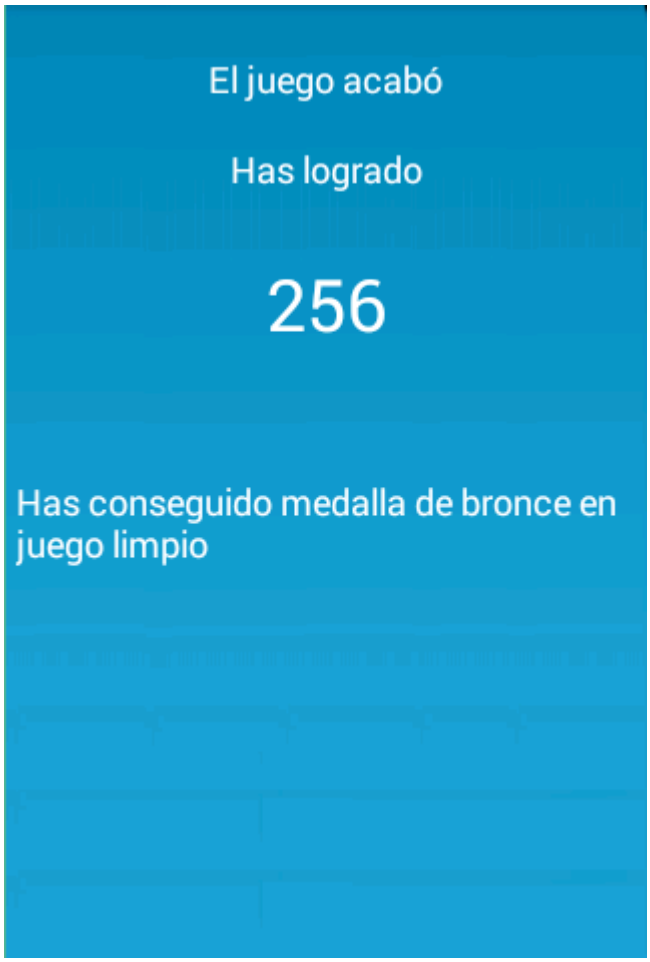

Ilustración 54: Pantalla GameFinishedActivity

Como resumen de lo anteriormente mencionado en relación con el flujo del programa a través de las actividades y su interacción con la base de datos se detalla el siguiente diagrama:

Parte 3: Estudio económico

Para realizar el estudio económico de este trabajo de fin de grado hemos de diferenciar entre los costes fijos y variables que puede general la realización de este proyecto.

Como costes fijos podemos desglosar el equipo utilizado para la programación de la aplicación. Para ello sirve cualquier ordenador con un mínimo de 3GB de memoria RAM y al menos 40GB de disco duro. Esto puede costar 400 euros como mínimo, pero ya que se ha realizado de forma doméstica no lo incluiremos en el cálculo. Otros costes que no incluiremos en el cálculo son el alquiler y gastos asociados al local donde se desarrolla la aplicación. En el caso de las empresas estos gastos se aplican multiplicando por 3 el coste de los recursos empleados para la realización del trabajo. Debido a que queremos evitar la computación de estos gastos, multiplicaremos por 2 los recursos empleados.

Otro de los aspectos a tener en cuenta es el tiempo que se dispone para la realización del proyecto ya que realizar el mismo trabajo en menos tiempo requiere una mayor disposición de recursos que en este caso no computaremos en el cálculo ya que he trabajado de forma individual y sin una meta temporal fijada para la entrega de la aplicación.

La experiencia del programador que realiza la aplicación encarece o abarata el presupuesto económico del trabajo. En mi caso, dada la escasa experiencia que poseía en el inicio del proyecto puedo valorar las horas trabajadas en 6 euros. A pesar de que en este momento valoraría más al alza la hora trabajada.

El número de horas empleadas en la realización del proyecto la aproximo en 300. Es difícil realizar un cálculo exacto, pero creo que es la cifra que más se ajusta a las horas efectivas invertidas en la implementación de la aplicación. Es importante definir que excluyo de estas horas, todas aquellas que he invertido en el aprendizaje. Solamente computo las horas que hubiese necesitado de haber sabido cómo desarrollar la aplicación en cada momento.

Por tanto valoro el coste económico del desarrollo de la aplicación (únicamente cubriendo gastos asociados al tiempo de realización de la misma) en 1200 euros.

"El logro más impresionante de la industria del software es su continua anulación de los constantes y asombrosos logros de la industria del hardware"

Henry Petroski - Especialista en análisis de fallos

Parte 4: Conclusiones y Líneas futuras

1. Conclusiones

Este trabajo de fin de grado nace de la actual necesidad de portar las plataformas *Web* a aplicaciones adaptadas a los dispositivos móviles. Esta migración de las plataformas *Web* ha sido llevada a cabo debido a la gran cantidad de personas que hoy en día tienen un *smartphone* como su dispositivo móvil. No solamente esto ha influido a la creación de aplicaciones móviles, si no que la mentalidad de la población también ha cambiado. Actualmente todo el mundo precisa un acceso a la información y servicios de Internet fácil y rápido. De este modo el desarrollo de aplicaciones móviles responde a la demanda de los usuarios satisfaciendo sus necesidades (Marta Fernandes, 2012).

La aplicación móvil que se ha desarrollado se basa en la idea de GBL (*Game Base Learning*) cuyo objetivo es el de promover el uso de juegos didácticos, como recursos formativos, de apoyo al personal docente.

Los sistemas *e-learning* ponen a disposición, tanto del alumno como del profesor, una herramienta muy eficaz para el apoyo de la parte no presencial de la asignatura. De esta manera el juego eLiza invita al alumno a comprobar sus conocimientos sobre una asignatura desde su dispositivo móvil Android a través de Internet, o de forma local, de un modo dinámico y divertido (Gomis, 2012).

Lo que se ha hecho ha sido trasladar la plataforma *Web* de eLiza ya existente en el módulo de aprendizaje *Moodle* a una aplicación Android. Conservando el método de juego y los recursos accedidos. De esta manera un mismo usuario podrá usar ambas plataformas indistintamente dotando a eLiza de un mayor acceso y flexibilidad de uso.

Con la realización de la aplicación se ha conseguido que ambas plataformas convivan, compartiendo los recursos de las bases de datos que *Moodle* ya usaba. Y también se han ampliado las funcionalidades del juego añadiendo un apartado de estadísticas, logros y clasificación personalizado para cada usuario y asignatura. Completando detalles que hacen al juego más accesible, vistoso y versátil. De este modo la aplicación eLiza para Android queda lista para ser utilizada por el personal docente de la Universidad de Valladolid.

2. Líneas Futuras

Una de las mejoras de las que se podría dotar a la aplicación es la implementación de animación en las transiciones. Esto haría que la interfaz de la aplicación fuese más acorde con el nuevo lenguaje de diseño gráfico de Android 5.0 llamada *Material Design* que se caracteriza por un diseño limpio en el que abundan las transiciones de respuesta, efectos de relleno y los efectos de profundidad. En esta línea hay aspectos a mejorar, que hagan que la aplicación sea más atractiva para el usuario sin perder de vista su motivación principal que es servir de apoyo docente.

Otra de las mejoras que se han quedado fuera de esta versión es la posibilidad de realizar retos entre los usuarios en los que se realicen una serie de preguntas (las mismas a ambos usuarios) y que posteriormente se computen los resultados nombrando a uno de los dos jugadores como ganador del reto o simplemente dejando el reto en empate. Esta idea desarrolla un tipo de interacción distinta de la que la aplicación implementa actualmente. Así, la aplicación actualmente se comunica únicamente con la base de datos del servidor, sin embargo, con la implementación de retos usuario-usuario realizaría una comunicación dispositivo-dispositivo. Una vez más, el único objetivo de esta implementación sería la de promover el uso de la aplicación entre los usuarios.

Una forma de realizar una promoción activa de la aplicación mediante las vías más actuales, que encajaría perfectamente en una línea futura de desarrollo, es la de publicar datos de la aplicación en las redes sociales. Esto significaría que un usuario pudiese publicar que ha desbloqueado un logro, alcanzado el primer puesto o logrado una puntuación concreta, en Facebook o Twitter. Para su realización sería necesario hacer uso de las APIs que estas redes sociales ponen a disposición de los programadores. También sería interesante la creación de *hashtags* que agrupasen los mensajes que estos usuarios publicaran.

La traducción de la aplicación a más idiomas que el castellano o el inglés expandiría el rango de usuarios que podría hacer uso de esta plataforma que puede ser exportada para todo tipo de contenidos y no únicamente a contenidos relacionados con la programación o las telecomunicaciones.

También sería interesante la introducción en la aplicación de eLiza para Android de las partidas en grupo que ya existen en la versión *Web.* Y de este modo, calcaría las funcionalidades que la plataforma *Web* ofrece.

3. Experiencia personal

Como experiencia personal la realización de este trabajo de fin de grado ha sido muy positiva en muchos aspectos que paso a detallar.

El inicio de la aplicación no fue todo lo complicado que podía haberme resultado ya que partía de una buena base de programación en Android gracias a los conocimientos que adquirí durante el curso pasado en la asignatura de Desarrollo de Aplicaciones para Dispositivos móviles. Esto no ha sido únicamente ayuda para la realización del trabajo de fin de grado si no que fue el detonante de mi afición por la programación de aplicaciones móviles además de acrecentar mi gusto por la programación en general.

Durante el desarrollo de este trabajo de fin de grado han sido numerosos los errores que he corregido sobre mi propio código por lo que también he mejorado mis habilidades de auto aprendizaje. Como programador he aprendido a diseñar estructuras de aplicaciones antes de comenzar a desarrollar el código. Esta es una de las costumbres que no tenía antes de comenzar a desarrollar este trabajo y que hoy en día me resulta imprescindible en mi trabajo.

En definitiva creo que he madurado como programador tanto técnicamente como en labores de planificación. He aprendido a dividir una tarea compleja en muchas tareas sencillas implementadas por pequeños módulos de aplicación relacionados que se ejecutan en distintos hilos.

Espero que la lectura de esta memoria sirva para futuros desarrollos del juego eLiza y que los alumnos que hagan uso de ella disfruten de su uso tanto como he disfrutado yo desarrollándola.

Bibliografía

Ableson, F. (2011). *Android: Guía para desarrolladores.* Anaya.

Ahson, M. I. (2012). *Smartphones: Research report.*

Alexander Alonso Molinero, F. J. (2012). Technological Solution for Improving Time. En H.-s. S. Seung-hwan Ju, *Study on Analysis Android Applications for Smartwork.*

Androidentity. (s.f.). *Androidentity - Firmar aplicaciones Android.* Recuperado el 18 de Noviembre de 2014, de http://androideity.com/2011/08/25/%C2%BFcomo-firmar-aplicaciones-android/

Apple. (2015). *Apple Developers.* Recuperado el 2015, de https://developer.apple.com/.

Barbera, E. (2008). *Aprender e learning.* Paidós.

Blackberry. (2014). *Blackberry.* Recuperado el Diciembre de 2014, de http://developer.blackberry.com.

BlackBerry. (2015). *BlackBerry Devices*. Recuperado el Abril de 2015, de http://global.blackberry.com/es/smartphones/blackberry-classic/specifications.html.

Corporation, O. (2012). *MySQL: The world's most popular open source database*. Recuperado el Abril de 2015, de https://www.mysql.com/

Cummings, I. K. (2013). Android on x86. In I. K. Cummings, *An Introduction to Optimizing for Intel Architecture.* Paperback.

Date, T. A. (2014). Recuperado el 2014, de http://www.theappdate.com/

Eclipse. (2014). *http://help.eclipse.org/luna/index.jsp?nav=%2F0.* Recuperado el Diciembre de 2014, de eclipse org.

Emmanuel Bertin y Noel Crespi, T. M. (2013). *Evolution of Telecommunication Services: The Convergence of Telecom and Ecosystems.*

Friends, A. (s.f.). *XAMPP*. Recuperado el Abril de 2015, de https://www.apachefriends.org/es/index.html.

Gironés, J. T. (2012). *El gran libro de Android.* Marcombo.

Gomis, J. M. (2012). *Mobile-Learning: Estrategias para el uso de aplicaciones, smartphones y tablets en educación.*

Google. (s.f.). *ADT Plugin.* Obtenido de http://developer.android.com/tools/sdk/eclipse-adt.html.

Google. (2014). *Android developer.* Recuperado el Diciembre de 2014, de http://developer.android.com.

Google. (2014). *Developer Android.* Recuperado el Noviembre de 2014, de Empezando a publicar aplicaciones: http://developer.android.com/distribute/googleplay/start.html

Google. (2015). *Google Play*. Recuperado el Abril de 2015, de http://play.google.es

Klemens, G. (2010). *The cellphone, the history and technology of the gadget that changed the world.* Paperback.

Marta Fernandes, C. M. (2012). Adaptation Model for PCMAT – Mathematics Collaborative Learning. En M. E. R. Berjón, *Mobile Virtual Platform to Develop Multiplatform Mobile Applications.*

Microsoft. (2014). *Windows phone developer center.* Recuperado el Diciembre de 2014, de hhtp://dev.windowsphone.com/en-us/develop.

Morrissey, S. (2010). *iOS Operating and File System Analysis.* Apress.

Nudelman, G. (2013). *Android Design Patterns: Interaction Design Solutions for Developers.* Wiley.

Play, A. d. (2014). *Google Play.* Recuperado el 18 de Noviembre de 2014, de https://play.google.com/intl/ALL_es/about/developer-distribution-agreement.html

Sevilla, U. d. *e-Learning. Definición y Características.* Sevilla: CFP, Centro de Formación Permanente.

Soft, A. (s.f.). *AJPD Soft.* Recuperado el 18 de Noviembre de 2014, de http://www.ajpdsoft.com/modules.php?name=News&file=article&sid=561

Soft, A. (18 de Noviembre de 2014). *AJPD Soft.* Obtenido de http://www.ajpdsoft.com/modules.php?name=News&file=article&sid=561

Anexo I: Configuración del entorno de desarrollo en Android

1. Introducción

Para el desarrollo de aplicaciones Android es importante trabajar en un entorno adecuado. Trabajar en un entorno apropiado con las mejores herramientas posibles nos facilita mucho el trabajo. Podemos disponer de las mejores herramientas de desarrollo de forma gratuita que son compatibles con cualquier equipo. Estas herramientas son las siguientes (Google, Android developer, 2014):

- *Java Runtime Environment,* versión 8 actualización 25
- Eclipse (Eclipse IDE *for Java Developers*)
- Android (SDK Google)
- Eclipse *Plug-in* (*Android Development Tools* – ADT)

A continuación se describe el proceso de instalación del *software* anteriormente mencionado:

2. Instalación de la máquina virtual Java

La instalación de este *software* permitirá la ejecución de código Java en el equipo. A esta máquina virtual también se le conoce como entorno de ejecución Java, *Java Runtime Environment* (JRE) o *Java Virtual Machine* (JVM).

Para instalar la máquina virtual de Java hemos de abrir el enlace http://www.java.com/es/download .

3. Instalación de Eclipse

Eclipse es el entorno de desarrollo de aplicaciones Android más recomendado. Es libre y soportado por Google. Los desarrolladores de Google lo usaron para desarrollar Android. La versión que se ha usado para desarrollar este trabajo es la 4.2.1. Los pasos son los siguientes (Eclipse, 2014):

1. Acceder a la página http://www.eclipse.org/downloads/ y descargar la última versión de Eclipse IDE *for Java Developers*. Es necesario reparar en este momento en la arquitectura de nuestro equipo y en nuestro sistema operativo para elegir qué paquete de archivos descargamos. También mencionar que las arquitecturas de 64 bits son representadas como x64 y las de 32 bits como x86. En mi caso, el equipo con el que desarrollé el trabajo corría con un sistema operativo Windows 8 de 64 bits.
2. Este entorno de desarrollo no requiere una instalación al uso. El ejecutable se descarga comprimido en un archivo. Por tanto, descomprimimos y ya tenemos Eclipse listo para ejecutarse.

3. Al arrancar Eclipse nos preguntará que donde queremos alojar nuestro directorio *workspace*. Lo primero mencionar que este escritorio contendrá como subcarpetas los proyectos que creemos mediante Eclipse. Es necesario conocer perfectamente donde se encuentra ubicado para hacer las copias de seguridad que creamos oportunas.
4. En la primera ejecución aparecerá una ventana de bienvenida, al cerrarla accedemos al entorno común de Eclipse que será lo que veamos en futuras ejecuciones.

4. Instalar Android SDK de Google (Eclipse, 2014)

Los pasos para la instalación del Android SDK de Google se enumeran seguidamente:

1. Acceder a la siguiente página http://developer.android.com/sdk/ y descarga el archivo que corresponda a tu sistema operativo. Al igual que en el caso de Eclipse, el SDK no requiere una instalación, si no que lo único necesario es descomprimir los archivos en el directorio que elijamos.
2. Al ejecutar el programa *SDK Manager* hemos de seleccionar los paquetes que deseemos instalar. En mi caso en un primer momento desarrollé la aplicación para la versión de Android que creí más extendida al comenzar: la 4.4.2 KitKat. Por ello únicamente descargué las librerías de la Android 4.4.2 API 19. Esta instalación puede durar varios minutos.

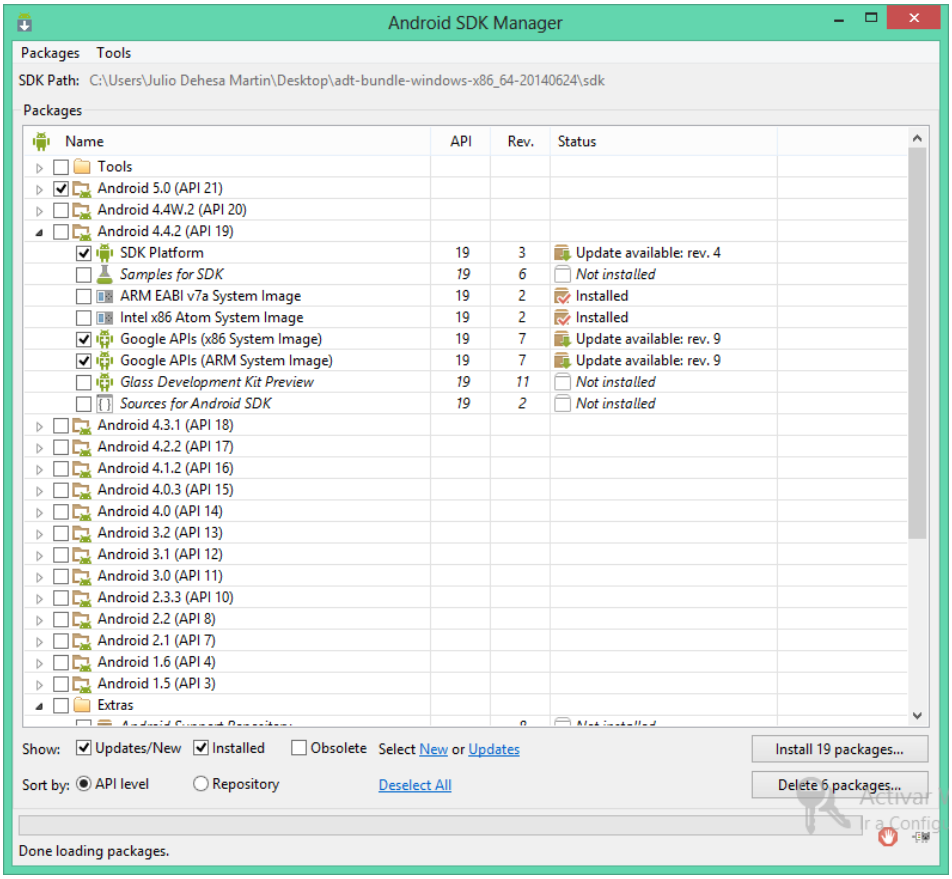

Ilustración 55: Android SDK Manager

El *SDK Manager* se puede ejecutar desde Eclipse pulsando un icono de la barra superior:

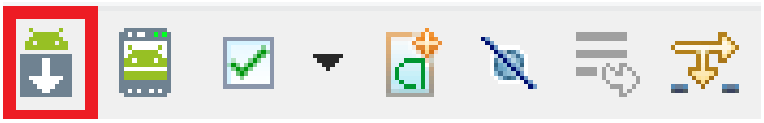

Ilustración 56: Icono Android SDK Manager en Eclipse

En lo sucesivo es interesante mantener actualizado nuestro SDK. Ya que las actualizaciones de las librerías Android se suceden constantemente, y siempre es recomendable trabajar con lo último.

3. Instalación del *Plug-in* de Android para Eclipse

Este *Plug-in* también es conocido como ADT. Este *software*, desarrollado por Google, instala complementos en Eclipse para que el entorno de desarrollo se adapte a las necesidades de un desarrollador de aplicaciones Android. De este modo el desarrollo será más cómodo y rápido ya que las herramientas han sido creadas específicamente para este fin.

1. Arrancar Eclipse y seleccionar *help -> Install new software*

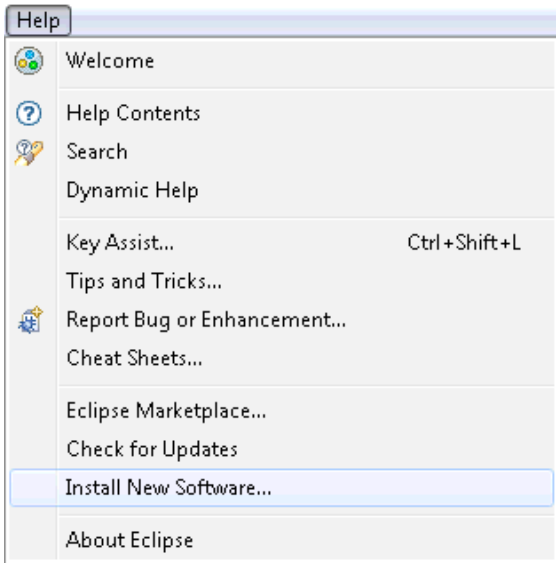

Ilustración 57: Instalación ADT – Instalar software

2. En el cuadro de diálogo *Add Site* que sale, introducir un nombre para el sitio remoto. En el campo *location* introducir la siguiente URL http://dl-ddl.google.com/android/eclipse/ .

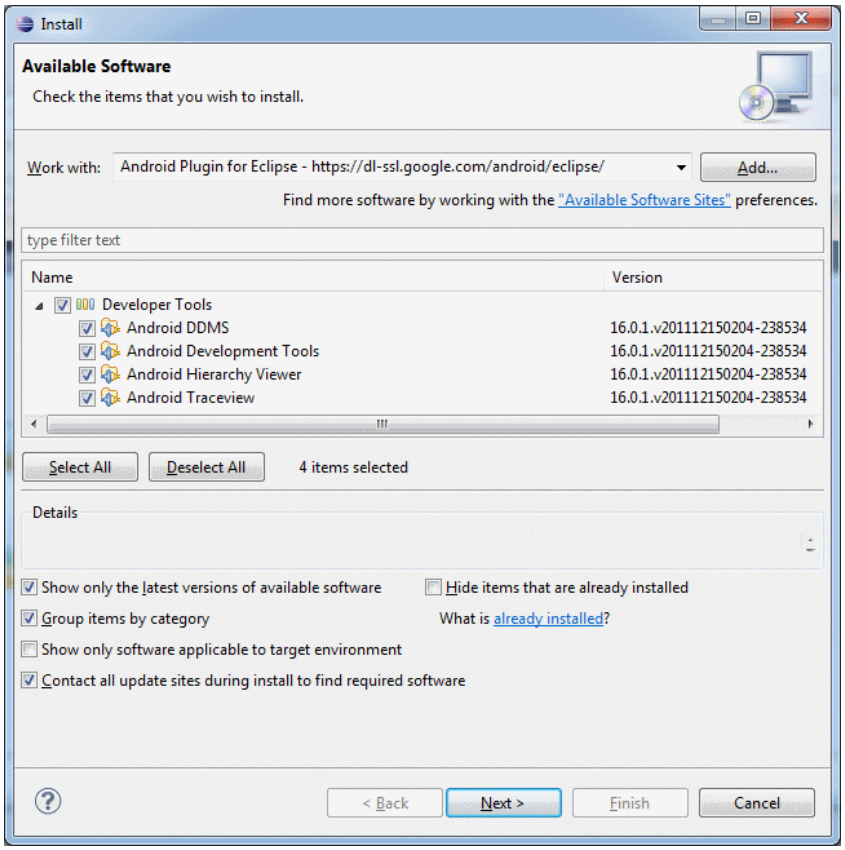

Ilustración 58: Instalación ADT - AddLocation

3. En este momento en el cuadro *Available software* debe aparecer *Developer Tools.*
4. Seleccionar los paquetes a instalar y pulsar *next*. Y ahora aparecen listadas las características *Android DDMS Android DDMS* y *Android Development Tools*. Pulsar *next* para aceptar la licencia y *finish* para finalizar.
5. Reiniciar Eclipse.
6. Configurar Eclipse para que sepa donde se ha instalado el Android SDK. Para ello entrar en *Window –> Preferences -> Android ->Browse* y examinamos la ruta en la que esté alojado el SDK en *SDK Location*. Pulsar *OK* para finalizar.

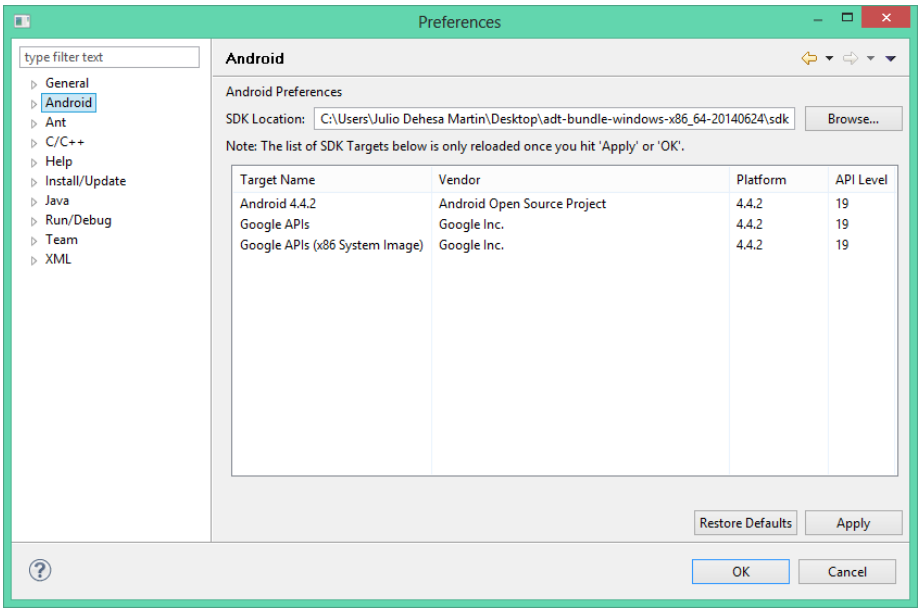

Ilustración 59: Instalación ADT – Configuración de localización de SDK

4. Creación de un dispositivo virtual de Android AVD

Un AVD o dispositivo virtual de Android permite emular en el ordenador el funcionamiento de un dispositivo móvil con la ventaja de que está ligado a Eclipse. De este modo podemos aprovechar las funciones de depuración de código que nos ofrece Eclipse conjuntamente con la emulación de la aplicación en nuestro propio equipo. También nos permite probar la aplicación en numerosos dispositivos sin necesidad de tenerlos físicamente lo que sería muy costoso. Así veremos cómo nuestra aplicación se adapta a cada uno de ellos y podremos modificar nuestra aplicación para que sea compatible con los terminales que deseemos. Para crear un AVD seguiremos los siguientes pasos:

1. En Eclipse pulsar el botón de la ilustración situado en la barra superior. Para abrir el AVD *Manager* que gestiona la creación, eliminación y modificación de AVDs.

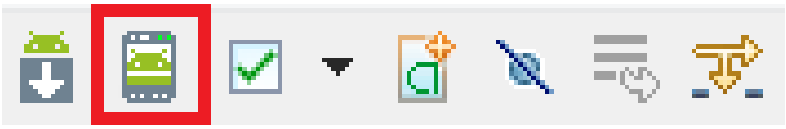

Ilustración 60: Creación de AVD – Abrir AVD Manager

2. Pulsar el botón *New* para crear un nuevo AVD. En la ventana emergente hemos de rellenar estos campos:

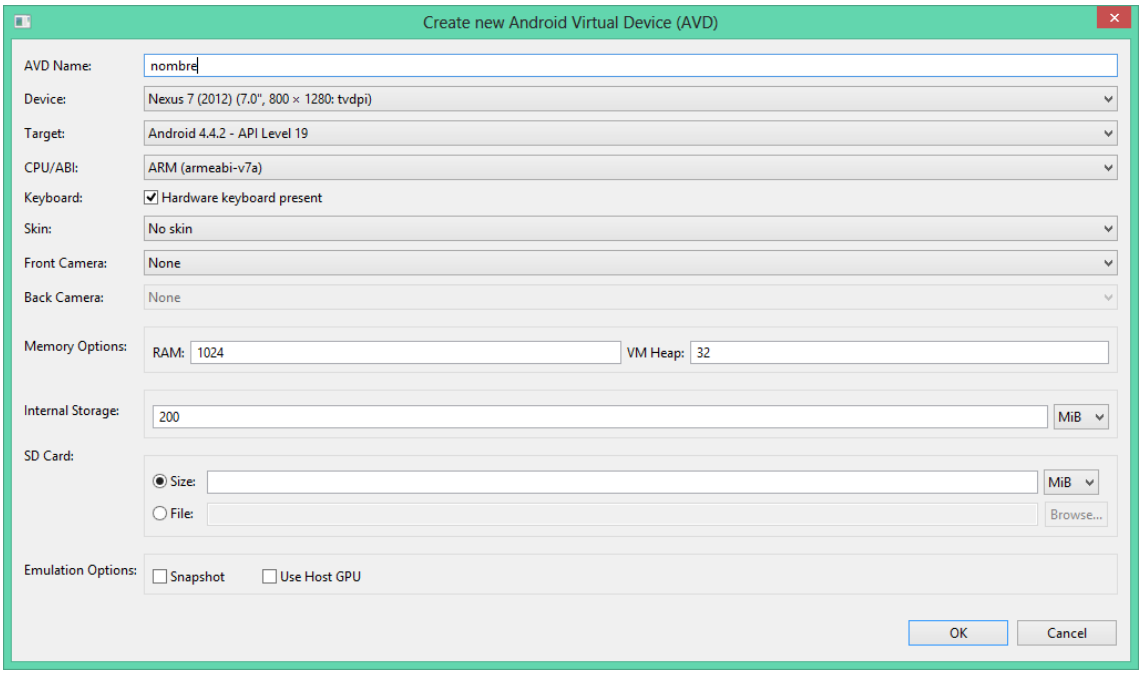

Ilustración 61: AVD - Creación

a. *AVD Name*: un nombre cualquier con el que distinguir nuestro nuevo dispositivo virtual.
b. *Device*: es el dispositivo que emularemos. Puede ser un terminal real, como se ofrecen los dispositivos Nexus de Google. O un terminal genérico con ciertas características configurables como las pulgadas o la resolución de pantalla.
c. *Target*: versión SDK que soportará el dispositivo. Sólo aparecen las versiones que hayamos elegido instalar anteriormente desde el *Android SDK Manager*.
d. CPU/ABI: tipo de CPU y arquitectura que se van a emular. En necesario tener en cuenta si se trabajará en código nativo, aunque la opción más habitual es ARM.
e. *KeyBoard*: al seleccionarla el emulador supondrá que el equipo tiene un teclado físico que actuará como teclado en el terminal. En caso contrario, el teclado de la pantalla del dispositivo emulado hará la función.
f. *Skin*: al seleccionarlo se mostrarán a la derecha del dispositivo emulado una serie de botones con volumen, *standby*, atrás o menú.
g. *Front/Back camera*: para activar la emulación de la cámara delantera o trasera.
h. *Memory options*: memoria que se dedicará al emulador. RAM: memoria total en MB. *VM Heap*: Memoria dinámica asignada a la máquina virtual.
i. *Internal storage*: memoria interna del dispositivo. Determinará el número de aplicaciones y datos que podrás instalar. Esta memoria será reservada del disco duro del equipo por lo que no conviene excederse demasiado.
j. *SD Card*: memoria externa del dispositivo emulado.

k. *Snapshot*: al seleccionarlo tendremos la posibilidad de congelar la ejecución del dispositivo en un determinado instante. Más tarde, podrás retomar la ejecución en ese instante, sin tener que esperar a que se reinicie el dispositivo. Conviene marcarlo para conseguir una carga más rápida.

l. *Use Host GPU*: se habilita la emulación de *hardware* para gráficos *OpenGLES*. Así la navegación entre ventanas será más fluida.

3. Aparecerá el dispositivo creado en la lista de AVDs. Para arrancarlo, se selecciona y se hace clic en *Start*.

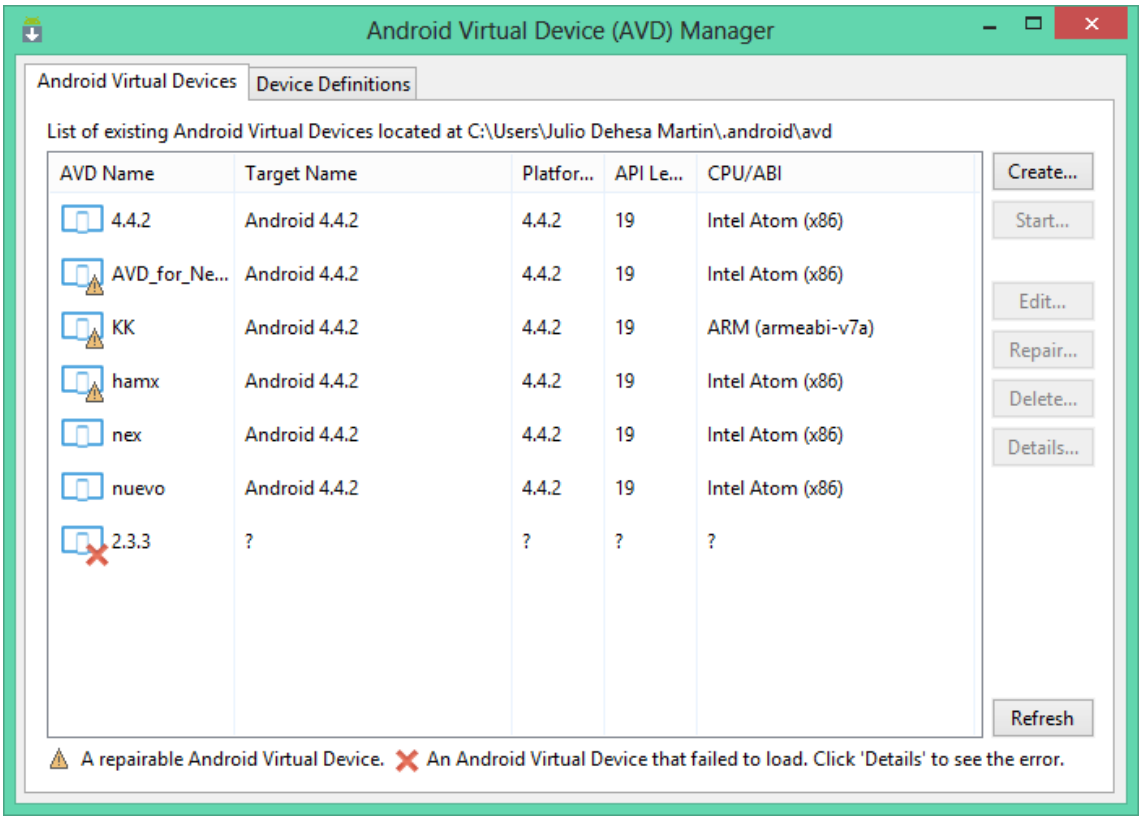

Ilustración 62: AVD Manager

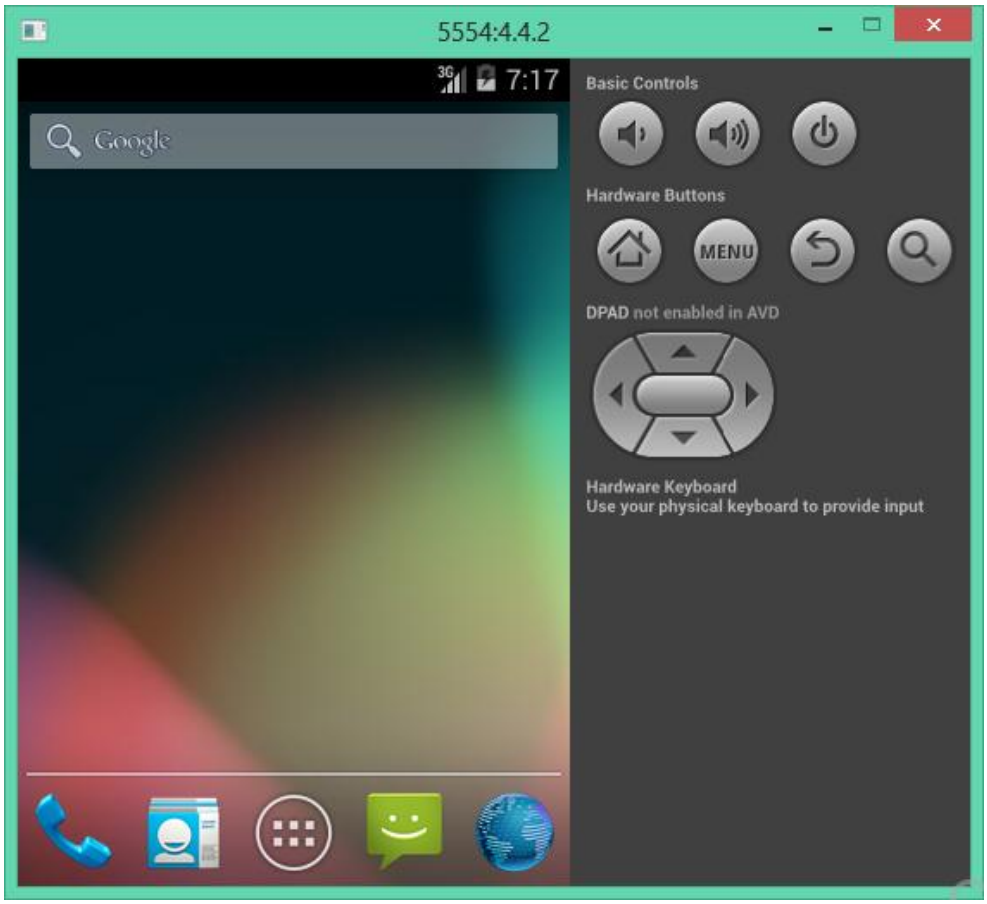

Ilustración 63: AVD

5. Creación de un proyecto Android (Play, 2014)

Utilizar un entorno de desarrollo adaptado a Android gracias al *plug-in* nos facilita mucho la creación de proyectos. Esto importa especialmente ya que tendremos que manejar una gran variedad de ficheros para concluir el desarrollo de nuestra aplicación.

1. Seleccionar *File-> New -> Android Project*
2. En la ventana emergente seleccionamos *Android Project* y pulsamos *next*.
3. Rellenar los detalles del proyecto con los siguientes valores:

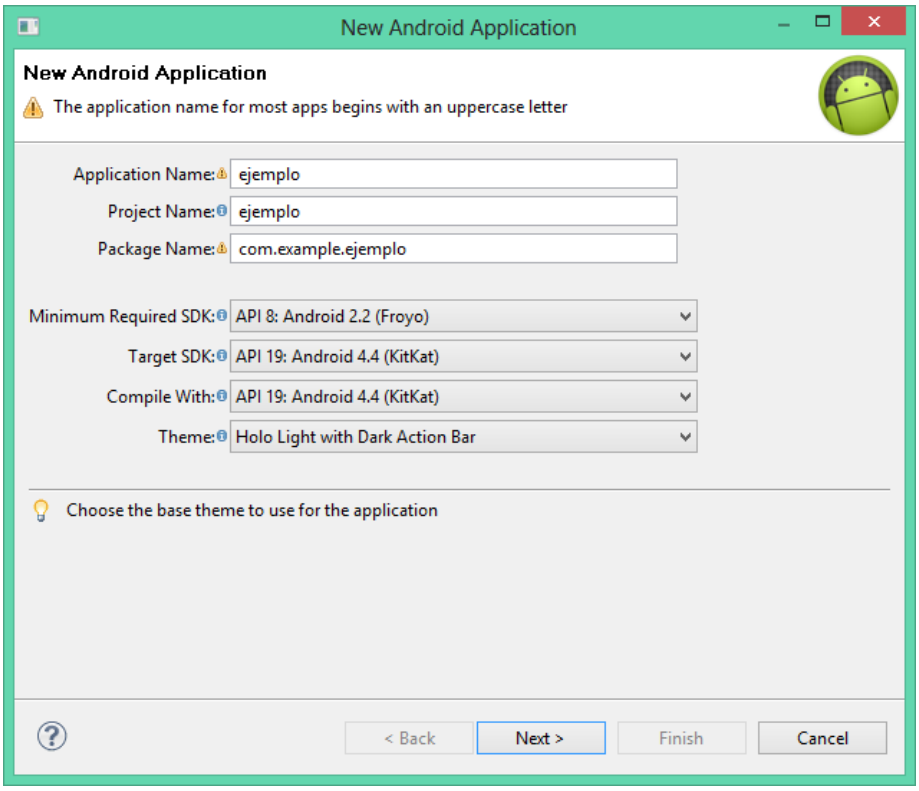

Ilustración 64: Creación proyecto Android – Datos iniciales

a. *Application name*: es el nombre de la aplicación que aparecerá en el dispositivo Android. Tanto en la barra superior cuando esté en ejecución como en el icono que se instalará en el menú de programas.
b. *Project name*: es el nombre del proyecto. Es el nombre de la carpeta que cuelga de *workspace* y da nombre al conjunto del proyecto.
c. *Package name*: indicamos el paquete con el espacio de nombre utilizado por nuestra aplicación. Hay que usar las reglas de los espacios de nombre en el lenguaje de programación Java. Las clases que creemos estará dentro de él. Esto también establece el nombre del paquete donde se almacenará la aplicación generada.
d. *Minimum required SDK*: este valor especifica el mínimo nivel del API que requiere tu aplicación. Por lo tanto, la aplicación no podrá ser instalada en dispositivos con una versión inferior. Procuraremos escoger valores pequeños para que la aplicación pueda instalarse en la mayoría de dispositivos. Un valor adecuado puede ser nivel de API 7, dado que cubriría más del 99,8% de los dispositivos. Escoger valores pequeños para este parámetro tiene un inconveniente: no podremos utilizar ninguna de las mejoras que aparezcan en los siguientes niveles de API.
e. *Target SDK*: indica la versión más alta de Android con la que se ha puesto a prueba la aplicación. Cuando salgan nuevas versiones del SDK, habrá que probar la aplicación y actualizar lo que sea necesario para que corra en esta nueva versión también.

f. *Compile with*: es la versión de la plataforma con la que se compila la aplicación. Se recomienda indicar la versión más reciente que haya aparecido. Las nuevas versiones no solo añaden funcionalidades al API, también añaden mejoras en el desarrollo. Por ejemplo, en algunas versiones se asigna un tema a la aplicación, o a partir de la versión 3.0 se verifica que un acceso a servidor *Web* se haga en un hilo auxiliar. Utilizar un *Target SDK* alto no está reñido con usar un *Minimum required* pequeño.

g. *Theme*: estilo que se utiliza en la aplicación. Un tema define los colores, fuentes y otros aspectos visuales de una aplicación.

4. Tras completar estos campos pulsa *next* para continuar. En la siguiente pantalla podrás configurar el icono de la aplicación. Se adaptará el tamaño de icono a cada una de las resoluciones de pantalla posibles en las que puede correr la aplicación. Pulsamos *next* para continuar.

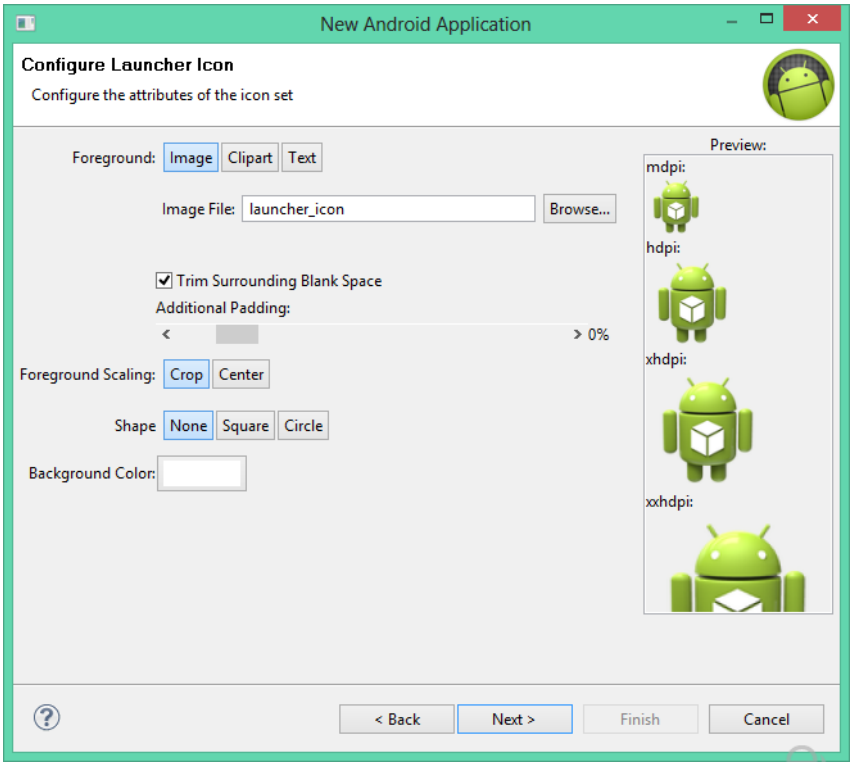

Ilustración 65: Creación proyecto Android - Icono

5. Si se marcó que se quería una actividad inicial, aquí podremos configurar su tipo. Como veremos más adelante una actividad es cada una de las pantallas en las que dividimos una aplicación. Pulsamos *next* para continuar.

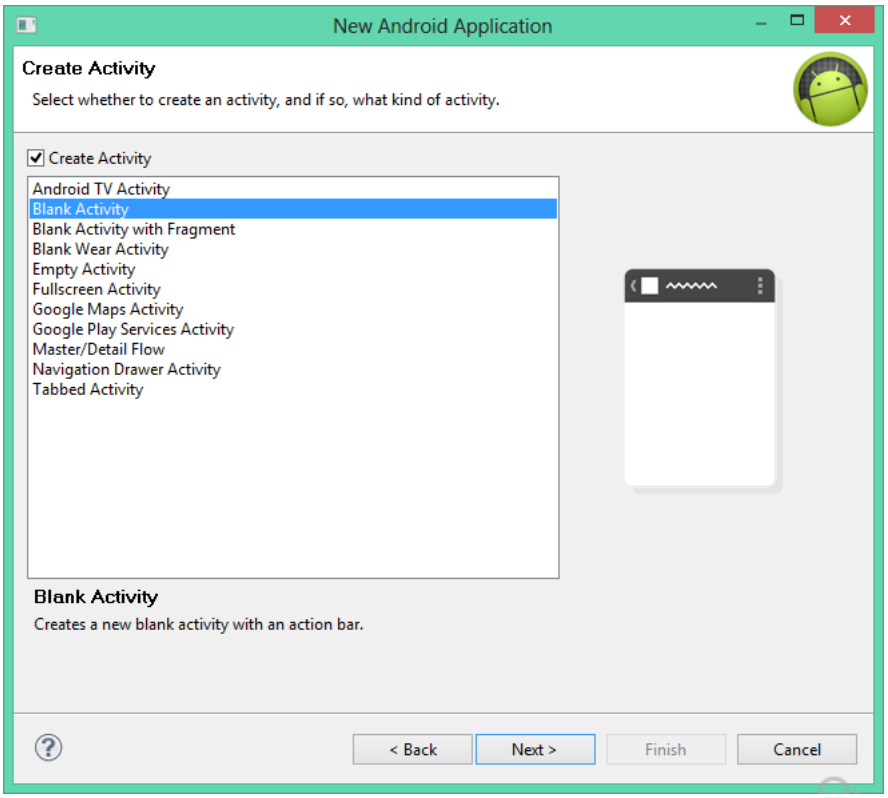

Ilustración 66: Creación proyecto Android – Tipo de Actividad inicial

6. Ahora podemos indicar el nombre de la actividad y el *layout* asociado a esta actividad. También podremos indicar el tipo de navegación. Pulsamos *finish* para finalizar la creación del proyecto.

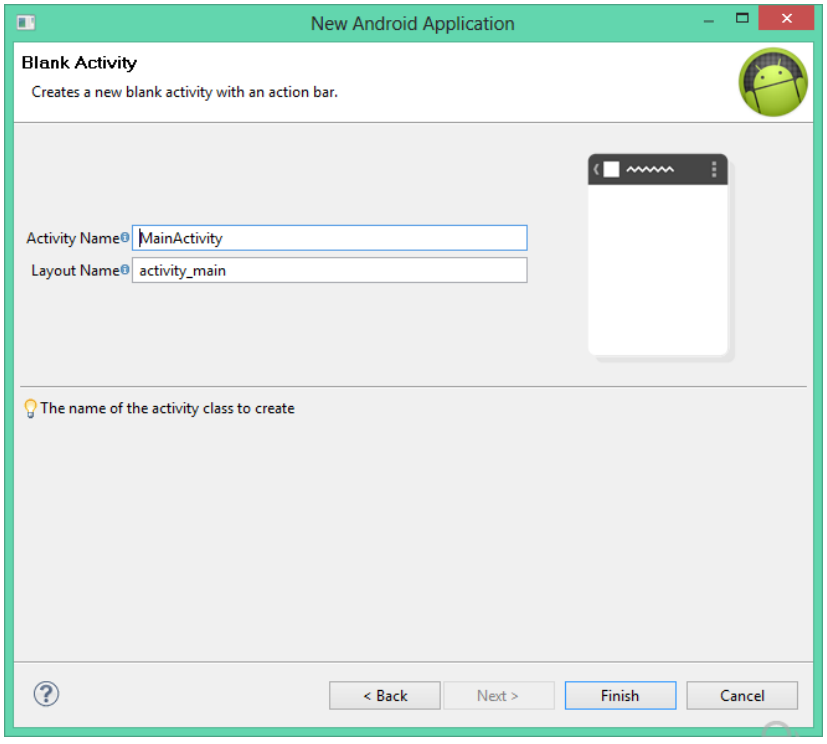

Ilustración 67: Creación proyecto Android – Actividad nombre

7. Ejecución en un terminal real

Ejecutar y depurar los programas en un terminal real es realizable desde el SDK. Es necesario conectar el dispositivo mediante USB al equipo. Es imprescindible haber instalado un *driver* específico en el equipo. Existe un *driver* genérico en la carpeta *android-sdk-windows\usb_driver* del SDK de Android. Aunque es probable que sea necesario usar el *driver* del fabricante del dispositivo. Los pasos a realizar para ejecutar la aplicación en un terminal son:

1. Abrir *Android SDK and AVD Manager* y asegurarse de que está instalado el paquete *USB Driver*. Puede que este *driver* no sea el adecuado para el terminal y haya que instalar uno específico del fabricante.

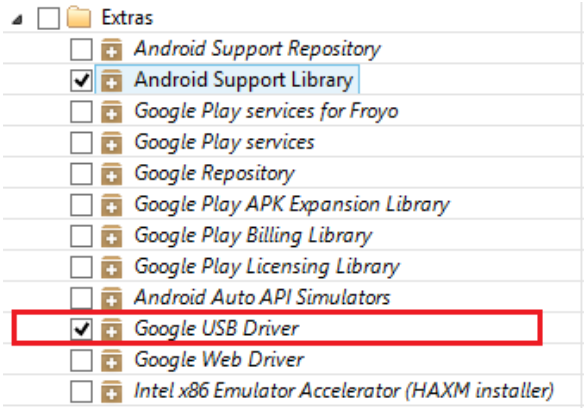

Ilustración 68: Ejecución en terminal – Instalación de USB Driver

2. En el terminal acceder al menú *Ajustes -> Opciones del desarrollador* y activar la opción de Depuración de USB activado.

Ilustración 69: Ejecución en terminal – Activar depuración terminal

3. Conectar el cable USB del terminal al equipo.
4. Indicar desde el equipo cual es el controlador a utilizar.

8. Distribuir una aplicación Android

Para correr una aplicación Android en un dispositivo esta ha de tener una firma digital instalada. Hay dos formas de conseguir esto:

- Mediante una clave de depuración: válido para correr la aplicación en un emulador o un dispositivo de depuración.

  En este caso no se necesita ninguna configuración específica por parte del usuario, debido a que el *plug-in* ADT firma de forma digital el '*apk*' generado para la instalación de la aplicación en el emulador mediante una clave auto generada.

- Mediante una clave privada: nos permite distribuir la aplicación. Para ello hemos de seguir una serie de pasos:
  a. Obtener una clave privada que cumpla con los siguientes requisitos: Solo estará en posesión de la persona o empresa que va a publicar la aplicación, la cual no se corresponde con la clave de depuración generada por ADT. Existe una herramienta llamada *KeyTool* que se encuentra en */usr/bin* que nos permite obtener una clave privada mediante un comando sencillo.
  b. Una vez obtenida la clave, es posible exportar la aplicación a un instalador de extensión apk listo para su distribución realizando los siguientes pasos:
     i. Desde Eclipse accedemos a *File -> Export –> Export Android Application*.
     ii. En la nueva ventana seleccionar el proyecto que se quiere exportar.

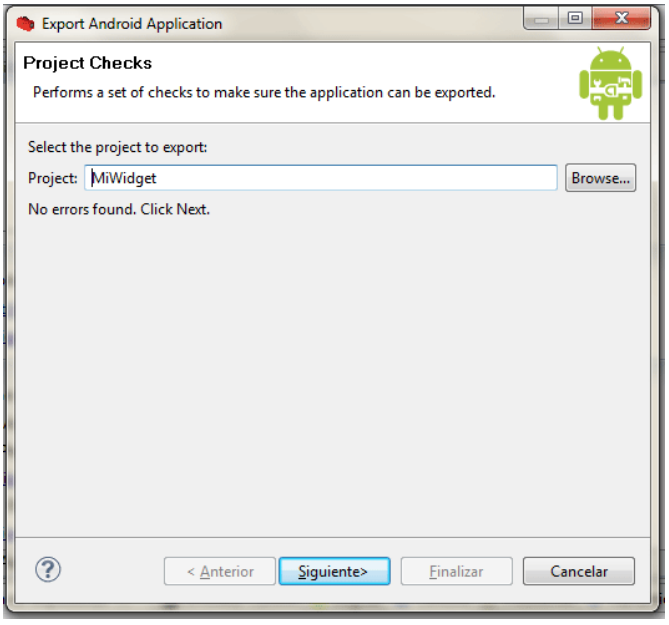

Ilustración 70: Distribución de aplicación Android – Selección de Proyecto

     iii. Seleccionar el almacén de claves creado en el paso anterior e introducir la clave de acceso a él.

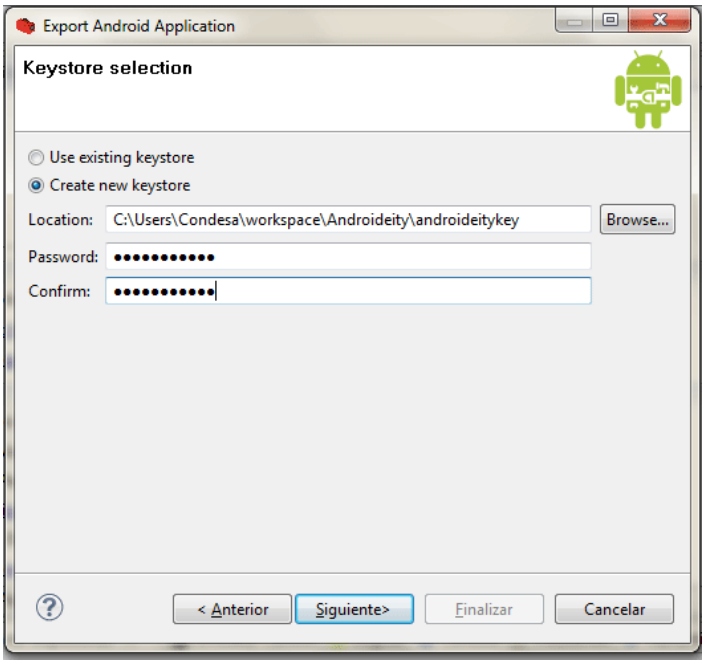

Ilustración 71: Distribución de aplicación Android – Clave de acceso

iv. Seleccionar la clave privada creada e indicar su clave de acceso.

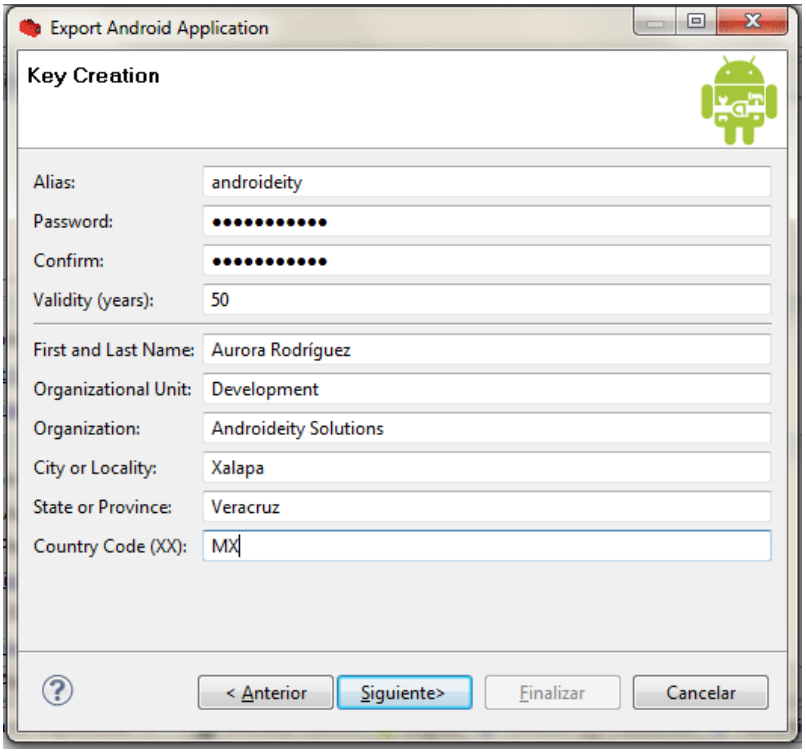

Ilustración 72: Distribución de aplicación Android – Selección de clave

v. Indicar la ubicación del archivo apk que será generado y pulsar *finish*.

Anexo II. Publicación de aplicaciones Android

Es evidente que la forma de medir el éxito de una aplicación es el número de descargas que tiene. Para que los usuarios de Android puedan descargar nuestra aplicación desde *Google Play* es necesario publicarla, y para ello hemos de seguir unos pasos previos: (Google, Developer Android, 2014)

1. El primer paso a dar es convertir nuestra cuenta de Google, en una cuenta de desarrollador. Para ello es necesario acceder al *Google Play Developer Console* mediante el *link https://play.google.com/apps/publish/signup/* .

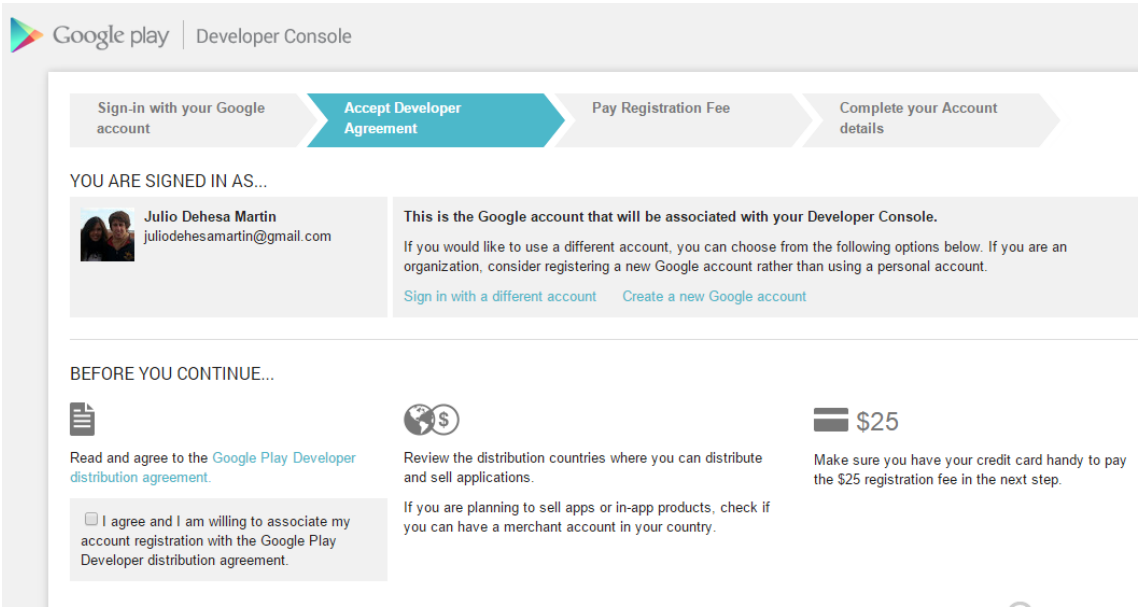

Ilustración 73: Publicación de aplicaciones – Cuenta de desarrollador

En esta página deberemos realizar un pago de 25 dólares una única vez, que nos permite ser distribuidor de aplicaciones en *Google Play* indefinidamente.

2. Una vez se haga efectivo el pago podremos entrar en *Google Play Developer Console*, que no es más que nuestro centro de gestión e información como desarrolladores. En esta página *Web* podremos ver:
    - El listado de nuestras aplicaciones
    - Servicios para *Google Play Games*
    - Informes sobre nuestros beneficios
    - Configuración
    - Anuncios
    - Alertas

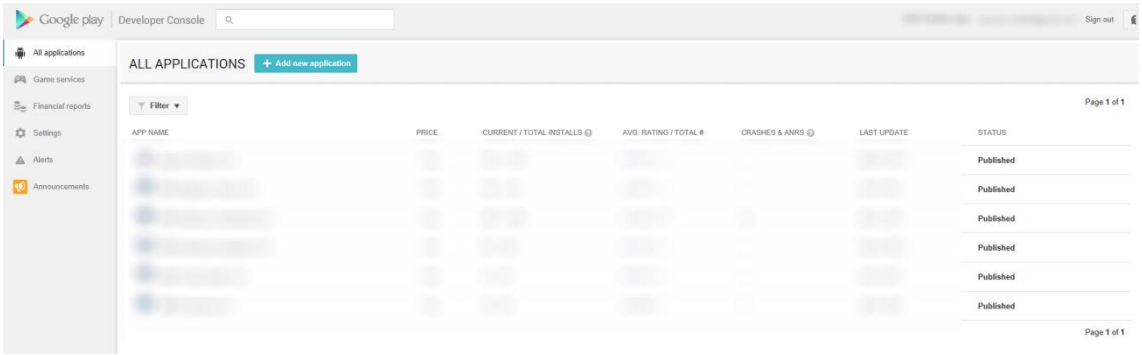

Ilustración 74: Publicación de aplicaciones – Google Play Developer Console

3. Nuestra intención es añadir una aplicación a *Google Play*, luego debemos pulsar el botón *Add new application*, seleccionaremos el lenguaje y el nombre de la aplicación en una ventana emergente.

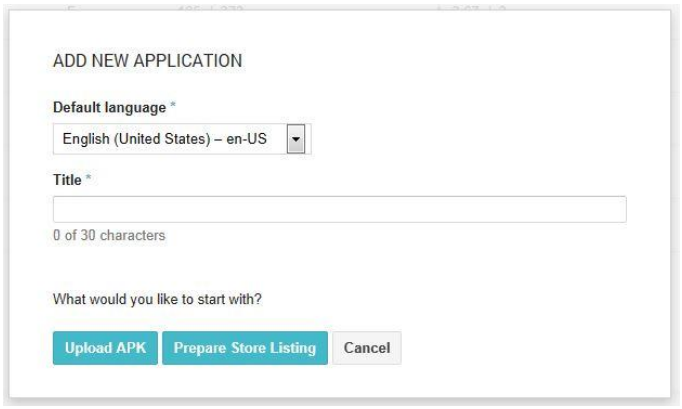

Ilustración 75: Publicación de aplicaciones – Selección de lenguaje y nombre

4. Una vez superado el paso anterior, se nos abrirá un formulario que deberemos rellenar con toda la información relativa a la aplicación. Podremos subir nuestro apk y desarrollar la descripción de la aplicación.

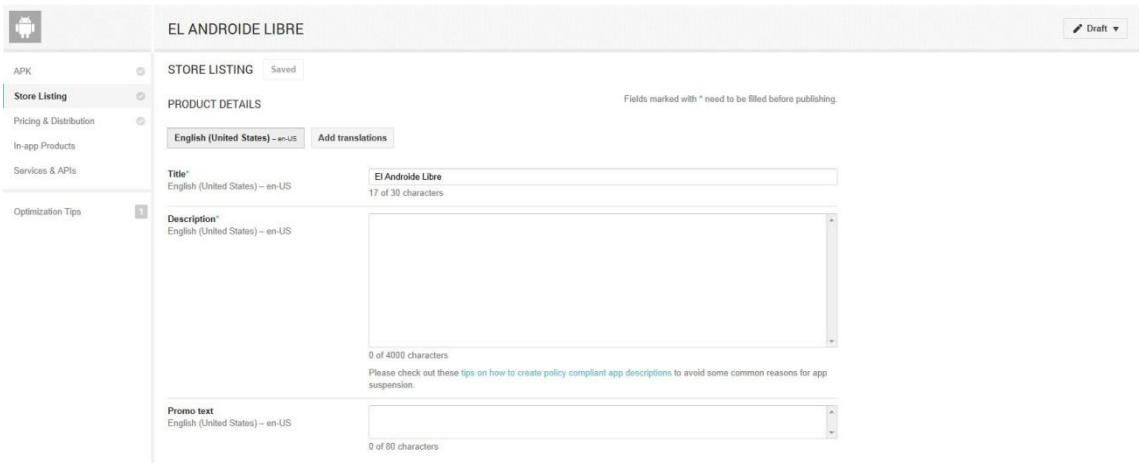

Ilustración 76: Distribución de aplicaciones – Subida de fichero

5. Una vez rellenado el formulario anterior y subido el fichero *'apk'* podremos incluir capturas de pantalla de la aplicación que se mostrarán en la pantalla de descargar de nuestra aplicación en *Google Play*.

Anexo III – Diseño de interfaz de usuario

El diseño de una interfaz de usuario adecuada se hace imprescindible a la hora de desarrollar una aplicación. Es la parte más en contacto con el usuario, la que puede ver, y la que crea la primera impresión de un usuario sobre una aplicación. No sólo es importante el aspecto de diseño de la aplicación, es decir, que sea atractiva. Es casi tanto o más importante como se han dispuesto los elementos sobre la pantalla para que el uso de la aplicación sea cómodo y adaptable a la pantalla y a las necesidades (Gironés, 2012).

En este anexo se mostrará en un primer ejemplo cómo diseñar una interfaz sencilla y posteriormente cómo podemos ir añadiendo complejidad al aspecto de la aplicación. Los métodos de interfaz usados por Android son sencillos y potentes, esto quiere decir, que con pocos conocimientos podemos lograr una interfaz con un buen aspecto.

Lo primero que hemos de indicar es que hay 2 formas de declarar elementos visuales (como cuadros de texto o botones), por código o mediante una declaración en un fichero XML. Lo declarado en un fichero XML serán elementos estáticos que deberán aparecer permanentemente en la actividad a la que estén ligados. También es conveniente indicar que algunas de sus propiedades pueden ser modificadas desde el código, por lo que son completamente estáticos. Lo declarado en el código es totalmente dinámico, puede aparecer o no aparecer según lo programemos.

Una aplicación bien sencilla sería un HolaMundo, quedaría de esta manera el código de la actividad y clase onCreate (que inicializa la vista) y la declaración de la vista (Google, Android developer, 2014):

```
package app.tutorial.holaMundo;

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

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

Código 1: Clase MainActivity de la aplicación HolaMundo

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

    <LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
```

```
        <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/helloworld"/>

</LinearLayout>
```

Código 2: Fichero XML de la aplicación HolaMundo

Para un HolaMundo únicamente necesitamos un elemento, un *TextView*. En este caso está contenido en un *layout* linear. En el *layout* podemos definir la orientación de la pantalla, que será vertical por defecto, podemos asignar una imagen o un color de fondo o si queremos dejar un espacio entre los bordes de la pantalla y los elementos que incluyamos en ella. En el caso del *LinearLayout* colocará los elementos hijos en una única fila horizontal o vertical según definamos. Si nuestra vista es linear, podremos asignarle pesos, de modo que cada elemento ocupará proporcionalmente la pantalla a razón del valor de su peso entre el valor de todos los elementos que componen la pantalla.

Según todo lo descrito en el anterior párrafo incluimos el ejemplo de un código que tiene implementados todos estos aspectos visuales de la aplicación. Hemos incluido un botón debajo del texto de "hola mundo" que ocupa un tercio de la pantalla. Esto no quiere decir que ocupe todo ese tercio. Según definamos sus atributos *height* (altura) y *width* (anchura): si les damos el valor *match_parent* o *fill_parent* ocuparán todo el espacio si damos el valor *wrap_content* se ajustarán a el contenido del botón.

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

<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:background="@drawable/webquest_background"
android:paddingTop="10sp"
android:paddingLeft="10sp">

      <TextView
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:layout_weight="2"
      android:text="@string/hello"/>

      <Button
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="@string/buttontext"
      android:layout_weight="1"/>

</LinearLayout>
```

Código 3: Fichero XML de la aplicación HolaMundo con botón

Si por ejemplo ahora quisiéramos declarar unos elementos al lado de otros, lo podríamos hacer con un *LinearLayout* sin embargo, la forma más práctica sería usar un *RelativeLayout*. La base de este tipo de *layout* es que la posición de cada uno de los elementos va definida de forma relativa respecto a la posición de otros elementos de la misma vista (Nudelman, 2013).

De este modo vamos a ejemplificar la siguiente vista:

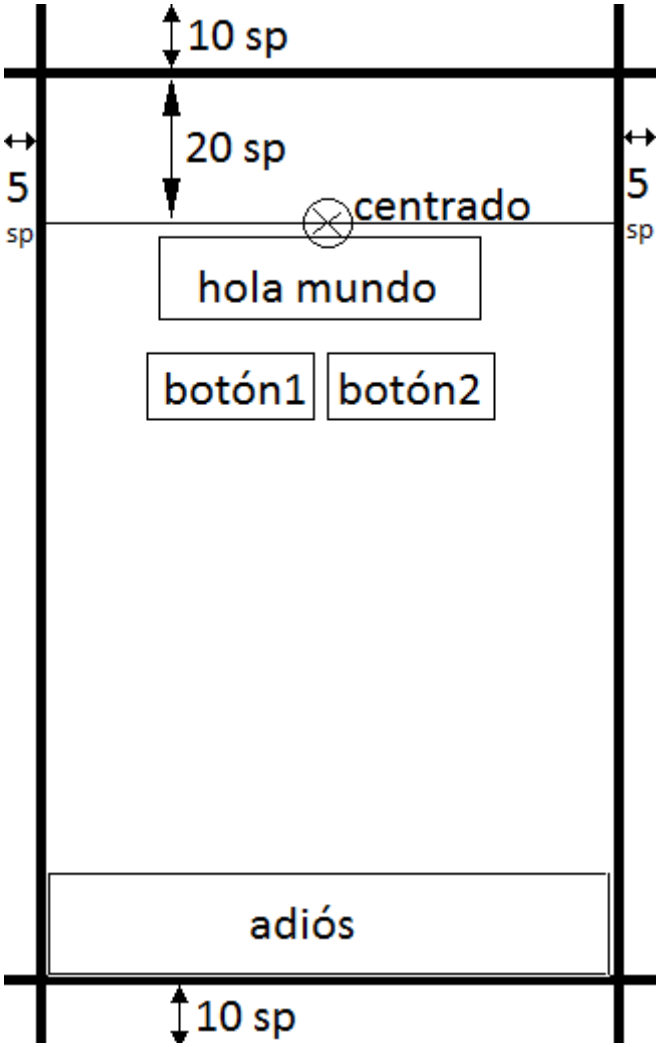

Ilustración 77: Representación simplificada de vista XML

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

<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:background="@drawable/background"
android:paddingTop="10sp"
android:paddingBottom="10sp"
android:paddingLeft="5sp"
android:paddingRight="5sp">
```

```
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:paddingTop="20sp"
        android:id="@+id/holamundo"
        android:text="@string/hello"/>

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/button1"
        android:layout_below="@id/holamundo"
        android:text="@string/buttontext1"/>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_alignTop="@id/button1"
        android:layout_toRightOf="@id/button1"
        android:id="@+id/space"/>

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/button2"
        android:layout_alignTop="@id/button1"
        android:layout_toRightOf="@id/space"
        android:text="@string/buttontext2"/>

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_toRightOf="@id/button1"
        android:text="@string/buttontext2"
        android:id="@+id/adios"/>

</RelativeLayout>
```

Código 4: Representación XML de HolaMundo complejo

En este caso, y en la mayoría, se hace necesaria la asignación de identificadores para cada elemento. Esto nos permitir estas relaciones de correspondencia entre los elementos. Pero también nos sirven para obtener referencias desde el código y poder hacer modificación de estos elementos estáticos. Por ejemplo podemos obtener una referencia a un *TextView* y cambiar el texto:

```
TextView holamundotext  = (TextView) findViewById(R.id.holamundo);
Holamundotext.setText("hola mundo, ¿qué  tal?");
```

Podemos cambiar su visibilidad:

```
holamundotext.setVisibility(0x00000004);
```

Con los botones podemos hacer estas operaciones también. Una de las usadas en la aplicación es la definición de la forma de un botón. Esto se realiza de forma estática en un fichero XML

que se localiza en la carpeta *drawable*. Aquí vemos un ejemplo que más adelante explicaremos:

```
<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android" >
    <stroke
        android:width="1dp"
        android:color="#505050"/>
    <corners
        android:radius="3dp" />

    <padding
        android:left="1dp"
        android:right="1dp"
        android:top="1dp"
        android:bottom="1dp"/>

    <solid android:color="#28D3D3"/>

</shape>
```

Código 5: Definición XML de un botón personalizado

El apartado *stroke* define el borde del botón. En este ejemplo configuramos su altura y color. El apartado *corners* define las esquinas del botón, mediante el atributo *radius* definimos cuánto de redondeadas queremos representarlas. El apartado *padding* define el margen del botón con otros elementos y *solid* el color de relleno del botón mediante el atributo *color*.

Tras esto, obtenemos una referencia al botón al que queramos asignar este estilo mediante la siguiente instrucción:

```
but.setBackground(getResources().getDrawable(R.drawable.button_style))
;
```

Anexo IV – Plan de validación

A continuación se presenta una tabla por cada actividad en las que se describen las pruebas llevadas a cabo por la aplicación y si han sido superadas:

MainActivity		
Prueba	**OK si…**	**Resultado**
Crear usuario nuevo	Entra y registra en las bases de datos	Esperado
Entrar con usuario existente en moodle	Entra	Esperado
Cerrar Sesión y crear usuario nuevo	Entra y registra nuevo usuario	Esperado
Cerrar Sesión **Entrar con usuario existente**	Entra a selección de asignaturas sin tener que introducir los datos	Esperado
Entrar con usuario erróneo	No entra, y muestra aviso	Esperado
Entrar con contraseña errónea	No entra, y muestra aviso	Esperado
Introducir caracteres no válidos al entrar	No entra, y muestra aviso	Esperado
Pulsar atrás	Muestra: ¿Salir o permanecer?	Esperado
Crear usuario nuevo en página moodle	Entra y registra en bases de datos	Esperado
Entrar con usuario existente en moodle	Entra	Esperado

Tabla 4: Plan de validación de MainActivity

RegisterActivity		
Prueba	**OK si…**	**Resultado**
Dejar cada box en blanco por separado	El proceso no continua, e informa	Esperado
Contraseña de menos de 6 caracteres	El proceso no continua, e informa	Esperado
Contraseña sin mayúsculas	El proceso no continua, e informa	Esperado
Contraseña sin caracteres especiales	El proceso no continua, e informa	Esperado
Contraseña sin números	El proceso no continua, e informa	Esperado
Correo con formato erróneo	El proceso no continua, e informa	Esperado
Pulsar atrás	Pasa a actividad principal	Esperado
Pulsar atrás y volver a entrar en la actividad de registro	Conservar datos introducidos en los boxes	Esperado
Probar a introducir tildes en el campo de usuario y contraseña	El proceso no continua, e informa	Esperado
Quitar internet, siguiente…	El proceso no continua, e informa	Esperado

Tabla 5: Plan de validación de RegisterActivity

RegisterCourseActivity		
Prueba	**OK si…**	**Resultado**
Intentar seleccionar varios elementos	No es posible	Esperado
Pulsar el elemento actualmente seleccionado	Se desmarca	Esperado
Volver atrás	Los datos se han guardado en sus boxes	Esperado
Pulsar siguiente sin seleccionar ningún curso	Informa y no deja continuar	Esperado
Selecciona un curso y pulsa siguiente	Pasa a la actividad de registro de asignatura	Esperado
Quitar internet, siguiente…	El proceso no continua, informa	Esperado
Intentar seleccionar varios elementos	No es posible	Esperado

Tabla 6: Plan de validación de RegisterCourseActivity

RegisterSubjectsActivity		
Prueba	**OK si…**	**Resultado**
Seleccionar, deseleccionar la misma asignatura	No ha de estar en el menu de la actividad *home*	Esperado
Seleccionar, deseleccionar, seleccionar la misma asignatura	Ha de estar en el menú de la actividad *home*	Esperado
Seleccionar y deseleccionar rápidamente	El menú contiene la selección mostrada antes de pulsar siguiente	Esperado
Quitar internet, siguiente…	El proceso no continua, e informa	Esperado

Tabla 7: Plan de validación de RegisterSubjectActivity

HomeActivity		
Prueba	**OK si…**	**Resultado**
Pulsar jugar sin seleccionar asignatura	Niega el acceso	Esperado
Seleccionar asignatura y pulsar jugar	Accede a juego	Esperado
Cambiar varias veces de asignatura, elegir y pulsar jugar	Niega el acceso	Esperado
Cambiar varias veces de asignatura y pulsar jugar	Accede a juego	Esperado
Mostrar y ocultar varias veces el tutorial	Muestra y oculta sin problema	Esperado
Quita internet, seleccionar asignatura, jugar (con asignatura ya jugada)	Avisa de que no actualizará la jugada, y jugará	Esperado
Quita internet, seleccionar asignatura, jugar (asignatura no jugada)	Avisa de que no puede jugar, no jugará	Esperado
Comprobar transición al resto de pestañas con internet	Transición normal, sin informar	Esperado
Comprobar transición al resto de pestañas SIN internet	Transición normal, informando	Esperado

Tabla 8: Plan de validación de HomeActivity

StatisticsActivity		
Prueba	**OK si…**	**Resultado**
Jugar, con aciertos y fallos en cada puntuación finalizando partida desde QuestIntroActivity	Las estadísticas están correctas	Esperado
Jugar, con aciertos y fallos en cada puntuación finalizando partida desde QuestActivity	Las estadísticas están correctas	Esperado
Jugar, con aciertos y fallos en cada puntuación finalizando partida por tiempo	Las estadísticas están correctas	Esperado
No llegar a jugar, habiendo entrado en QuestIntroActivity	Estadísticas no son afectadas	Esperado
Fallar todo y comprobar el estado del % de aciertos totales	Ha de ser 0	Esperado
Probar todo sin internet	Ha de funcionar correctamente	Esperado

Tabla 9: Plan de validación de StatisticActivity

ClassificationActivity		
Prueba	**OK si…**	**Resultado**
En home quitar internet, ir a ClassificationActivity, consultar clasificación, SIN datos existentes	No muestra clasificación e informa	Esperado
En home quitar internet, ir a ClassificationActivity, consultar clasificación, con datos existentes	Muestra clasificación e informa que puede no estar actualizado	Esperado
Realizar máxima puntuación, desde QuestActivity, QuestIntroActivity y por tiempo.	Está actualizado al entrar en ClassificationActivity	Esperado

Tabla 10: Plan de validación de ClassificationActivity

Anexo V – Estructura de la base de datos local

Tabla: achievement ***Contiene los datos necesarios para realizar el recuento de los logros y para almacenar los logros ya conseguidos.***		
Nombre del campo	**Tipo de variable**	**Uso**
_id	INTEGER PRIMARY KEY AUTOINCREMENT	Identificador propio que diferencia los logros de las asignaturas de cada uno de los jugadores
nameUser	TEXT	Nombre del usuario (*nick)*
userid	INT	Identificador único de cada usuario
nameSubject	TEXT	Nombre de la asignatura
nRightConsecutive	INT	Identifica qué logro ha alcanzado el usuario. 4: nada, 3: bronce, 2: plata y 1: oro.
nRightConsecutiveCount	INT	Número de preguntas consecutivamente acertadas. Se fija a 0 cada vez que el usuario falla una pregunta.
nMaxRightConsecutive		Número máximo de preguntas consecutivamente acertadas por usuario y asignatura.
scoreReached	INT	Máximo marcador logrado en la asignatura.
cleanGame	INT	Identifica el mínimo número de errores logrado por el usuario.
riskGameCount	INT	Identifica el número de preguntas pasadas por el usuario en una partida concreta. Se fija a 0 al acabar una partida.
riskGame	INT	Identifica el mínimo número de preguntas pasadas por el usuario en todas las partidas de la asignatura.
ovecomingScore	INT	Identifica la máxima puntuación conseguida por el usuario en dicha asignatura.
overcomingScoreCount	INT	Cuenta el número de veces que el usuario ha superado su propia máxima puntuación.

classificationPodium	INT	Identifica la mejor posición que el usuario ha logrado en la clasificación general de la asignatura.

Tabla 11: Estructura de la tabla achievement de la base de datos local

Tabla: answer ***Almacena toda la información (obtenida de la base de datos moodle) relativa a las respuestas del juego***		
Nombre del campo	**Tipo de variable**	**Uso**
id	INTEGER	Identificador de la base de datos moodle correspondiente a la pregunta que pertenece
text	TEXT	Texto del contenido de la respuesta
true	INTEGER	1: si es la respuesta correcta 0: en caso contrario

Tabla 12: Estructura de la tabla answer de la base de datos local

Tabla: category ***Tabla que relaciona las categorías de juego con cada una de las asignaturas y sus cursos correspondientes***		
Nombre del campo	**Tipo de variable**	**Uso**
course	TEXT	Identificador del curso relacionado con la categoría
subject	TEXT	Identificador de la asignatura correspondiente al curso
category	TEXT	Nombre de la categoría
categoryid	INT	Identificador de la categoría

Tabla 13: Estructura de la tabla category de la base de datos local

Tabla: classification *Tabla que contiene el volcado de datos de la clasificación alojada en la base de datos moodle*		
Nombre del campo	**Tipo de variable**	**Uso**
_id	INTEGER PRIMARY KEY AUTOINCREMENT	Identificador de posición usuario y asignatura en la clasificación
position	INT	Posición sobre la clasificación de la asignatura concreta de un usuario
nameUser	TEXT	Nombre del usuario
nameSubject	TEXT	Nombre de la asignatura
nameSubjectId	INT	Identificador de la asignatura
score	INT	Puntuación máxima del usuario en dicha asignatura

Tabla 14: Estructura de la tabla classification de la base de datos local

Tabla: question *Información relativa a las preguntas del juego volcada desde la base de datos moodle*		
Nombre del campo	**Tipo de variable**	**Uso**
id	INT	Identificador de la pregunta de valor, el almacenado en la base de datos moodle y mediante el que se le relaciona a las respuestas.
point	INT	Puntuación a la que pertenece la pregunta
name	TEXT	Descripción general de la pregunta
type	INT	Tipo de pregunta
categorynum	INT	Categoría a la que está asignada esta pregunta

Tabla 15: Estructura de la tabla question de la base de datos local

Tabla: statistic *Datos de juego locales del usuario en cada una de las partidas jugadas*		
Nombre del campo	**Tipo de variable**	**Uso**
_id	INTEGER PRIMARY KEY AUTOINCREMENT	Identifica las estadísticas de cada usuario en cada una de sus asignaturas
nameSubject	TEXT	Nombre de la asignatura
nameUser	TEXT	Nombre del usuario
nGames	INT	Número de partidas jugadas
nAnswerRight	INT	Número de respuestas acertadas
nAnswerRight100	INT	Número de respuestas acertadas de puntuación 100
nAnswerRight200	INT	Número de respuestas acertadas de puntuación 200
nAnswerRight300	INT	Número de respuestas acertadas de puntuación 300
nAnswerRight400	INT	Número de respuestas acertadas de puntuación 400
nAnswerRight500	INT	Número de respuestas acertadas de puntuación 500
nAnswer100	INT	Número de respuestas acertadas de puntuación 100
nAnswer200	INT	Número de respuestas acertadas de puntuación 200
nAnswer300	INT	Número de respuestas acertadas de puntuación 300
nAnswer400	INT	Número de respuestas acertadas de puntuación 400
nAnswer500	INT	Número de respuestas acertadas de puntuación 500
maxScore	INT	Máxima puntuación lograda por el usuario

bestPositionClassification	INT	Mejor posición del usuario en la clasificación general de la asignatura
userid	INT	Identificador del usuario

Tabla 16: Estructura de la tabla statistic de la base de datos local

Tabla: subject *Datos de las asignaturas cursos y usuarios registrados en el dispositivo de juego*		
Nombre del campo	**Tipo de variable**	**Uso**
userid	INT	Identificador del usuario
username	TEXT	Nombre del usuario
subject	TEXT	Nombre de la asignatura suscrita
subjectid	INT	Identificador de la asignatura

Tabla 17: Estructura de la tabla subject de la base de datos local

Tabla: user *Datos que identifican el registro del usuario en el dispositivo*		
Nombre del campo	**Tipo de variable**	**Uso**
_id	INTEGER PRIMARY KEY AUTOINCREMENT	Identificador único del usuario en el dispositivo
userid	INT	Identificador del usuario
username	TEXT	Nombre del usuario
pass	TEXT	Contraseña cifrada del usuario
course	TEXT	Curso del usuario
lastUser	BOOLEAN	Identifica si es el usuario actualmente registrado en el dispositivo o no

Tabla 18: Estructura de la tabla user de la base de datos local

Anexo VI – Tablas de la base de datos *moodle* de las que hace uso eLiza

Tabla: mdl_course			
Columna	**Tipo**	**Nulo**	**Predeterminado**
id	bigint(10)	No	
category	bigint(10)	No	0
sortorder	bigint(10)	No	0
password	varchar(50)	No	
fullname	varchar(254)	No	
shortname	varchar(100)	No	
idnumber	varchar(100)	No	
summary	text	Sí	NULL
format	varchar(10)	No	topics
showgrades	tinyint(2)	No	1
modinfo	longtext	Sí	NULL
newsitems	mediumint(5)	No	1
teacher	varchar(100)	No	Teacher
teachers	varchar(100)	No	Teachers
student	varchar(100)	No	Student
students	varchar(100)	No	Students
guest	tinyint(2)	No	0
startdate	bigint(10)	No	0
enrolperiod	bigint(10)	No	0
numsections	mediumint(5)	No	1

marker	bigint(10)	No	0
maxbytes	bigint(10)	No	0
showreports	smallint(4)	No	0
visible	tinyint(1)	No	1
hiddensections	tinyint(2)	No	0
groupmode	smallint(4)	No	0
groupmodeforce	smallint(4)	No	0
defaultgroupingid	bigint(10)	No	0
lang	varchar(30)	No	
theme	varchar(50)	No	
cost	varchar(10)	No	
currency	varchar(3)	No	USD
timecreated	bigint(10)	No	0
timemodified	bigint(10)	No	0
metacourse	tinyint(1)	No	0
requested	tinyint(1)	No	0
restrictmodules	tinyint(1)	No	0
expirynotify	tinyint(1)	No	0
expirythreshold	bigint(10)	No	0
notifystudents	tinyint(1)	No	0
enrollable	tinyint(1)	No	1
enrolstartdate	bigint(10)	No	0
enrolenddate	bigint(10)	No	0
enrol	varchar(20)	No	
defaultrole	bigint(10)	No	0

Tabla 19: Estructura de la tabla mdl_ course de la base de datos de moodle

Tabla: mdl_course			
Columna	**Tipo**	**Nulo**	**Predeterminado**
id	bigint(10)	No	
category	bigint(10)	No	0
sortorder	bigint(10)	No	0
password	varchar(50)	No	
fullname	varchar(254)	No	
shortname	varchar(100)	No	
idnumber	varchar(100)	No	
summary	text	Sí	NULL
format	varchar(10)	No	topics
showgrades	tinyint(2)	No	1
modinfo	longtext	Sí	NULL
newsitems	mediumint(5)	No	1
teacher	varchar(100)	No	Teacher
teachers	varchar(100)	No	Teachers
student	varchar(100)	No	Student
students	varchar(100)	No	Students
guest	tinyint(2)	No	0
startdate	bigint(10)	No	0
enrolperiod	bigint(10)	No	0
numsections	mediumint(5)	No	1
marker	bigint(10)	No	0
maxbytes	bigint(10)	No	0
showreports	smallint(4)	No	0
visible	tinyint(1)	No	1

hiddensections	tinyint(2)	No	0
groupmode	smallint(4)	No	0
groupmodeforce	smallint(4)	No	0
defaultgroupingid	bigint(10)	No	0
lang	varchar(30)	No	
theme	varchar(50)	No	
cost	varchar(10)	No	
currency	varchar(3)	No	USD
timecreated	bigint(10)	No	0
timemodified	bigint(10)	No	0
metacourse	tinyint(1)	No	0
requested	tinyint(1)	No	0
restrictmodules	tinyint(1)	No	0
expirynotify	tinyint(1)	No	0
expirythreshold	bigint(10)	No	0
notifystudents	tinyint(1)	No	0
enrollable	tinyint(1)	No	1
enrolstartdate	bigint(10)	No	0
enrolenddate	bigint(10)	No	0
enrol	varchar(20)	No	
defaultrole	bigint(10)	No	0

Tabla 20: Estructura de la tabla mdl_course de la base de datos de moodle

Tabla: mdl_course_categories			
Columna	**Tipo**	**Nulo**	**Predeterminado**
id	bigint(10)	No	
name	varchar(255)	No	
description	text	Sí	NULL
parent	bigint(10)	No	0
sortorder	bigint(10)	No	0
coursecount	bigint(10)	No	0
visible	tinyint(1)	No	1
timemodified	bigint(10)	No	0
depth	bigint(10)	No	0
path	varchar(255)	No	
theme	varchar(50)	Sí	NULL

Tabla 21: Estructura de la tabla mdl_course_categories de la base de datos de moodle

Tabla: mdl_eliza_categoria			
Columna	**Tipo**	**Nulo**	**Predeterminado**
id	bigint(10)	No	
courseid	bigint(10)	No	0
categoriaid	bigint(10)	No	0
namecategoria	varchar(255)	No	
personalizado	bigint(10)	No	0

Tabla 22: Estructura de la tabla mdl_eliza_categoria de la base de datos de moodle

Tabla: mdl_eliza_classification			
Columna	**Tipo**	**Nulo**	**Predeterminado**
id	int(11)	No	
user_id	int(11)	No	
user_name	text	No	
subject_name	text	No	
max_score	int(4)	No	
subject_id	int(11)	No	

Tabla 23: Estructura de la tabla mdl_eliza_classification de la base de datos de moodle

Tabla: mdl_eliza_preguntas			
Columna	**Tipo**	**Nulo**	**Predeterminado**
id	bigint(10)	No	
categorianum	bigint(10)	No	0
preguntaid	bigint(10)	No	0
nombrepregunta	varchar(255)	No	
textopregunta	mediumtext	No	
tipopregunta	bigint(10)	No	0
personalizado	bigint(10)	No	0
creador	varchar(255)	No	
textorespuesta	varchar(255)	No	
propuesta	bigint(10)	No	0
decrecimiento	bigint(10)	No	0
tipodecre	varchar(255)	No	Ninguno

Tabla 24: Estructura de la tabla mdl_eliza_preguntas de la base de datos de moodle

Tabla: mdl_eliza_respuestas			
Columna	**Tipo**	**Nulo**	**Predeterminado**
id	bigint(10)	No	
preguntanum	bigint(10)	No	0
textorespuesta	mediumtext	No	
correcta	tinyint(2)	No	0

Tabla 25: Estructura de la tabla mdl_eliza_respuestas de la base de datos de moodle

Tabla: mdl_user			
Columna	**Tipo**	**Nulo**	**Predeterminado**
id	bigint(10)	No	
auth	varchar(20)	No	manual
confirmed	tinyint(1)	No	0
policyagreed	tinyint(1)	No	0
deleted	tinyint(1)	No	0
mnethostid	bigint(10)	No	0
username	varchar(100)	No	
password	varchar(32)	No	
idnumber	varchar(255)	No	
firstname	varchar(100)	No	
lastname	varchar(100)	No	
email	varchar(100)	No	
emailstop	tinyint(1)	No	0
icq	varchar(15)	No	
skype	varchar(50)	No	
yahoo	varchar(50)	No	

aim	varchar(50)	No	
msn	varchar(50)	No	
phone1	varchar(20)	No	
phone2	varchar(20)	No	
institution	varchar(40)	No	
department	varchar(30)	No	
address	varchar(70)	No	
city	varchar(20)	No	
country	varchar(2)	No	
lang	varchar(30)	No	en_utf8
theme	varchar(50)	No	
timezone	varchar(100)	No	99
firstaccess	bigint(10)	No	0
lastaccess	bigint(10)	No	0
lastlogin	bigint(10)	No	0
currentlogin	bigint(10)	No	0
lastip	varchar(15)	No	
secret	varchar(15)	No	
picture	tinyint(1)	No	0
url	varchar(255)	No	
description	text	Sí	NULL
mailformat	tinyint(1)	No	1
maildigest	tinyint(1)	No	0
maildisplay	tinyint(2)	No	2
htmleditor	tinyint(1)	No	1
ajax	tinyint(1)	No	1

autosubscribe	tinyint(1)	No	1
trackforums	tinyint(1)	No	0
timemodified	bigint(10)	No	0
trustbitmask	bigint(10)	No	0
imagealt	varchar(255)	Sí	NULL
screenreader	tinyint(1)	No	0

Tabla 26: Estructura de la tabla mdl_user de la base de datos de moodle