Escola Universitària
Politècnica de Mataró

Centre adscrit a:

UNIVERSITAT POLITÈCNICA
DE CATALUNYA

Grado en Ingeniería Informática

AndroidUp

Memoria

Mireia Collado Ruiz
PONENTE: Catalina Juan Nadal

PRIMAVERA 2016

TecnoCampus
Mataró-Maresme

Dedicatoria

A Rodrigo, por estar siempre ahí.

A mis padres, Enrique y Esperanza, por tenerme paciencia.

A mi gran familia, por apoyarme en todo y hacerme ver que todo llega.

A todas esas personas que han formado parte de ésta experiencia, mil gracias.

Agradecimientos

Especial agradecimiento a los profesores del Tecnocampus

que me han formado en la profesional que soy hoy.

A la comunidad de stackoverflow.com

y a mis compañeros de trabajo del departamento móvil, Carles y Oriol.

Resumen

Este proyecto consiste en el desarrollo de una aplicación en Android nativo, usando Java como lenguaje de programación. Contará además con una API que usando Spring devolverá el contenido de la aplicación de forma dinámica y en tiempo real, haciendo de puente entre la app y la Base de Datos de MongoDB. Con esta aplicación, los usuarios podrán introducirse en el mundo de la programación Android mediante temas y subtemas ordenados en forma de tutoriales.

Resum

Aquest projecte consisteix en el desenvolupament d'una aplicació en Android natiu, usant Java com a llenguatge de programació. Comptarà a més amb una API que usant Spring tornarà el contingut de l'aplicació de forma dinàmica i en temps real, fent de pont entre l'app i la Base de Dades de MongoDB. Amb aquesta aplicació, els usuaris podran introduir-se al món de la programació Android mitjançant temes i subtemes ordenats en forma de tutorials.

Abstract

This project involves developing a native Android application using Java programming language. It will also have an API that using Spring will return the application content dynamically and in real time, making a bridge between the app and the MongoDB database. With this application, users can enter the world of Android programming, by topics and subtopics arranged in the form of tutorials.

Índice

Índice de figuras

Índice de tablas

Glosario de términos

AoT	Ahead of Time
API	Application Programming Interface
APK	Android Application Package
APP	Aplicación
ART	Android Runtime
Bytecode	Forma intermedia entre código y lenguaje máquina
DSL	Domain Specific Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
JAR	
JDT	Java Development Toolkit
JRE	Java Runtime Environment
POM	Project Object Model
POO	Programación Orientada a Objetos
RAM	Random Acces Memory
SDK	Software Development Kit
Smartphone	Teléfono inteligente
UI	User Interface
XML	eXtensible Markup Language
ZIP	

1. Objetivos

1.1. Propósito

Crear una aplicación móvil en Android que permita leer tutoriales de cómo programar en Android, organizada en temas y subtemas, extraídos en tiempo real de una Base de Datos alojada en un servidor y llamados mediante una API, que tendrá como objetivo la comunicación entre servidor y aplicación.

1.2. Finalidad

Conseguir una aplicación móvil en Android que ampliará los conocimientos de programación en Java adquiridos durante el Grado, además de ampliar el repertorio de herramientas de control de versiones, herramientas de integración e IDEs, entre otros.

1.3. Objeto

Una vez acabado el proyecto, se tendrá una aplicación compatible con los dispositivos Android más recientes, que se podrá publicar en la tienda de aplicaciones Google Play y que se podrá descargar para su utilización.

1.4. Alcance

La aplicación será desarrollada con Android Studio en el lenguaje de programación Java y con librerías Android integradas por la herramienta Gradle. El control de versiones se hará mediante GitHub. El temario de los temas y subtemas de la aplicación estará en un servidor de Digital Ocean y se consultará mediante una API creada con ése propósito. La API será desarrollada en IntellIJ con Java y Spring, con la ayuda de Maven. Se configurará con una interfície sencilla, intuitiva y respetando en la medida de lo posible el Material Design de Google. Los menús de la aplicación estarán disponibles en castellano e inglés una vez se cambie el idioma del dispositivo. Se destaca el hecho de que esta aplicación es sólo un prototipo y, por lo tanto, sólo tendrá una pequeña parte del temario y que, por falta de tiempo, algunas funcionalidades quedarán pendientes de implementar.

2. Estudio Previo

En cuanto a aplicaciones móvil se refiere, la mejor forma de hacer un estudio previo es optar por descargar diversas aplicaciones del market dónde se quiera subir la aplicación una vez esté acabada, para así estudiar a la competencia y saber qué cosas hacen bien y qué deberían mejorar.

2.1. Aplicaciones parecidas

Existen un gran número de aplicaciones móviles que tratan el mismo tema en el mercado. Si buscamos “android tutorial” en el mercado de aplicaciones Google Play nos encontramos con un sinfín de apps disponibles para descargar:

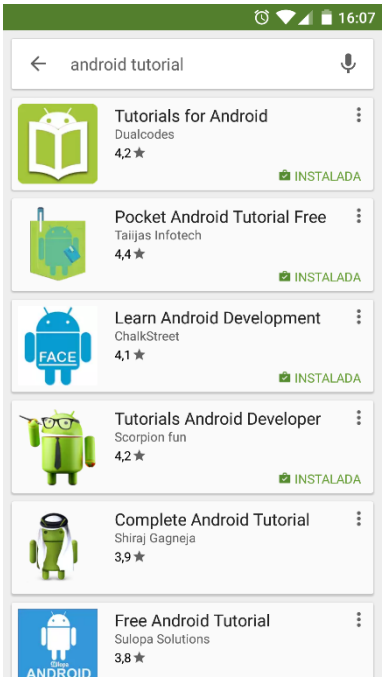

Fig. 2.1. Imagen de búsqueda “android tutorial” en Google Play

Con la finalidad de estudiar las posibles ventajas y desventajas de cada una, se ha realizado un estudio previo al desarrollo sobre éstas aplicaciones, que forman parte de la competencia. Al haber demasiadas para analizarlas todas, se han descargado algunas de las que están por encima de las 4/5 estrellas que dispone Google Play, con el objetivo de estudiar las mejores del mercado, descartando las que tienen poca valoración por parte del usuario.

2.1.1. Learn Android Development [1]

Al entrar en esta aplicación, se muestra un menú donde se puede elegir un tema. Cuando se selecciona un tema, aparece una pantalla dónde el usuario debe realizar acciones como introducir texto, apretar botones, etc. para visualizar los resultados. No lleva nada de teoría, sólo son casos prácticos y se puede consultar el código fuente apretando el botón de "VIEW SOURCE CODE", dónde se muestra la clase en Java, el XML y extras (manifest.xml).

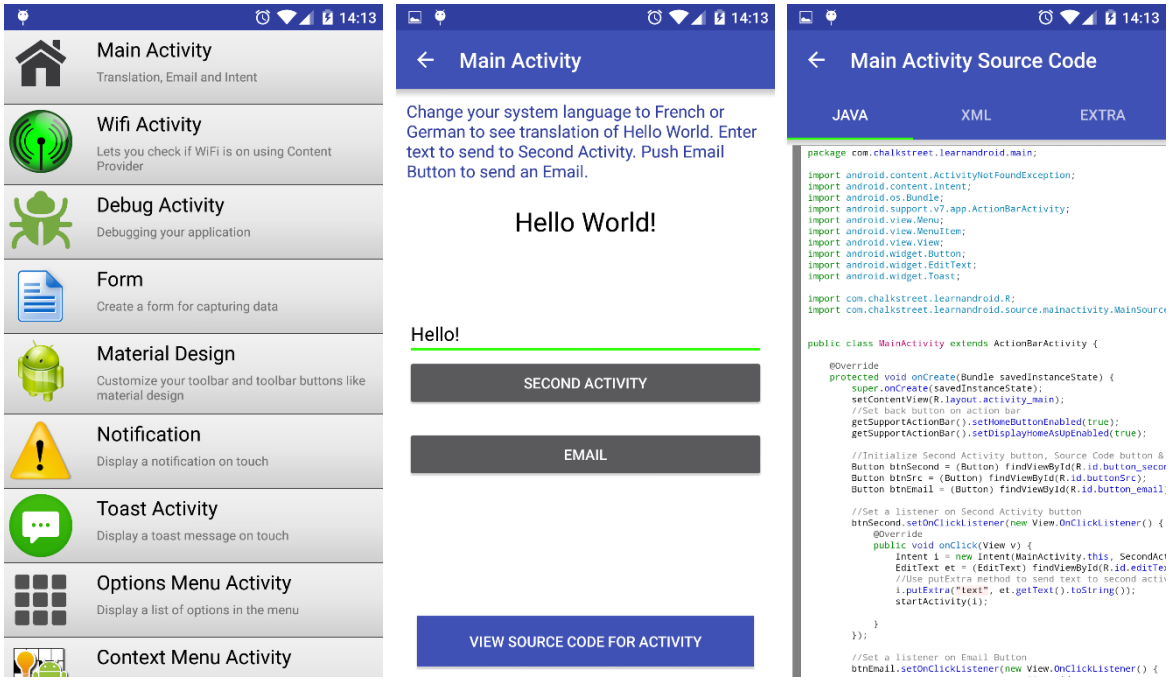

Fig. 2.3. Imágenes de la aplicación "Learn Android Development"

Ventajas	Desventajas
• Interfaz fácil y usable • Ejercicios prácticos • Visualización del código fuente • No lleva publicidad • Diseño adaptable al dispositivo	• No dispone de teoría

Tabla 2.1. Tabla ventajas y desventajas de la aplicación "Learn Android Development"

[1] Fig. 2.2. Icono de la aplicación "Learn Android Development"

2.1.2. Learn Android Development (15 days) [2]

Esta aplicación promete que en 15 días aprenderás Android. Divide entonces la teoría por días, mostrando sólo 3 cada vez. Es muy difícil de seguir, los temas son muy largos y para pasar de página hay que hacer scroll hasta abajo del todo, lo mismo que al acabar el último día que se muestra en pantalla, con lo que el usuario se puede perder fácilmente. En sí, la aplicación no concuerda con el tema, ya que ni el icono, ni ciertas imágenes, ni los colores son apropiados para un tutorial de Android.

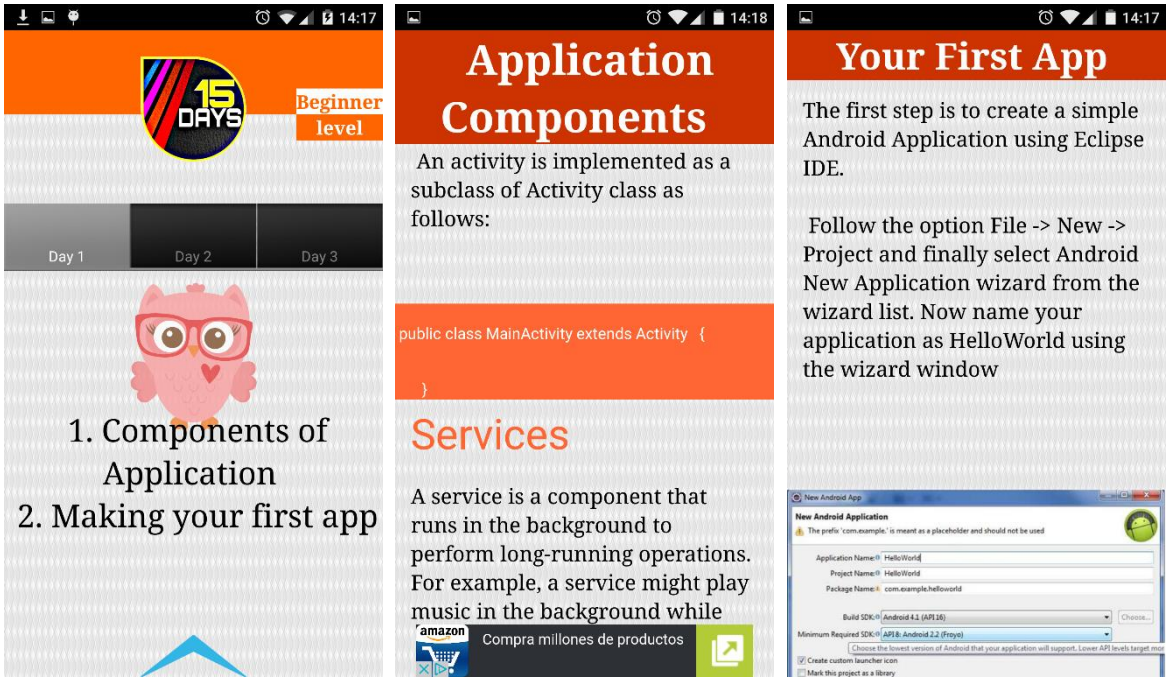

Fig. 2.5. Imágenes de la aplicación “Learn Android Development (15d)”

Ventajas	Desventajas
• Dispone de teoría	• Interfaz complicada y no usable • Publicidad a pie de página que distrae al usuario • Colores no acordes al tema • Imágenes de aves no relacionadas con Android • Icono de la aplicación no relacionado con el tema • Difícil de seguir, temas muy largos (demasiado scroll)

Tabla 2.2. Tabla ventajas y desventajas de la aplicación “Learn Android Development (15d)”

[2] Fig. 2.4. Icono de la aplicación “Learn Android Development (15d)”

2.1.3. learnandroid|android tutorial [3]

La primera pantalla que se muestra son 9 botones multicolor: Home, Tutorial, Examples, Interview, App Specific, Website, Blog, Videos y Contact. Demasiado botón para el usuario, que lo que busca es un tutorial. Si se selecciona Tutorial, la aplicación lleva a una pantalla dónde para empezar hay que apretar el botón izquierdo inferior para abrir el listado de temas. Se tendrá que seleccionar subtema, y cuando empieza el tutorial en la parte superior se encuentra el título y unos botones de compartir estáticos que dificultan la lectura y en la parte inferior, 5 botones que son muy poco intuitivos que provocan que el usuario se pierda.

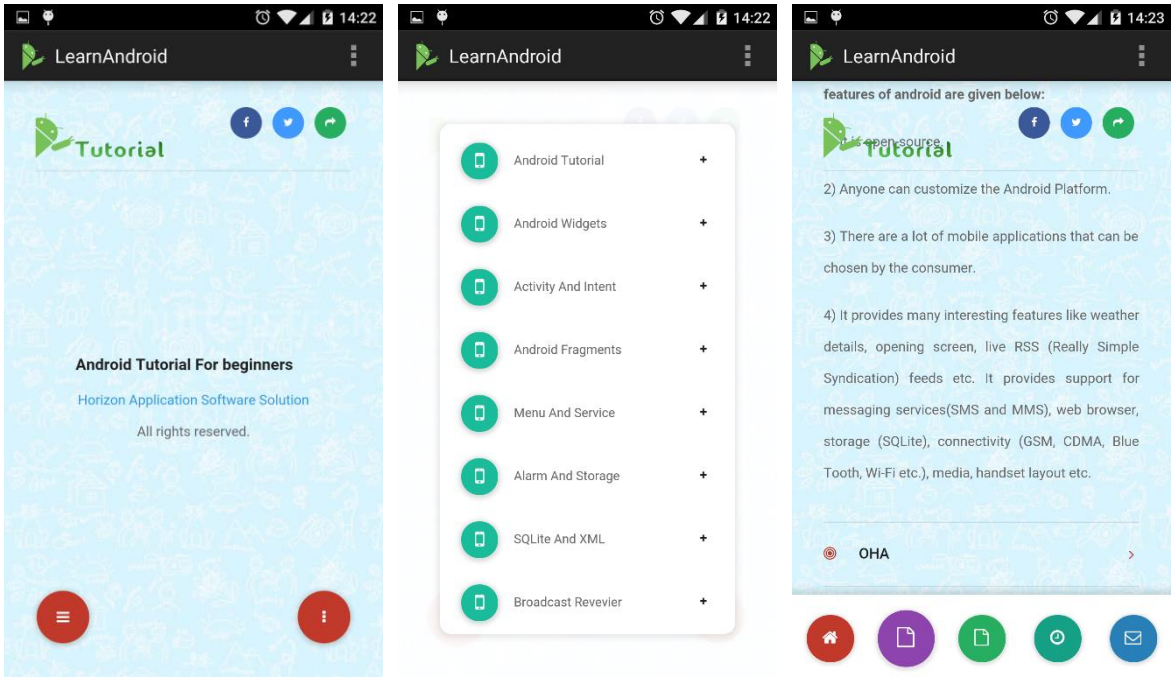

Fig. 2.7. Imágenes de la aplicación “learnandroid|android tutorial”

Ventajas	Desventajas
• Dispone de teoría • No lleva publicidad	• Interfaz complicada y difícil de seguir • Colores poco acordes al tema • Título y botones de compartir estáticos en la parte superior que dificultan la lectura • Botones poco intuitivos en la parte inferior

Tabla 2.3. Tabla ventajas y desventajas de la aplicación “learnandroid|android tutorial”

[3] Fig. 2.6. Icono de la aplicación “Tutorials for learnandroid|android tutorial”

2.1.4. Pocket Android Tutorial Free [4]

Al entrar, la aplicación consta de varias pestañas en la parte superior con los 4 temas principales del tutorial. Si se selecciona una pestaña, muestra la lista de subtemas a escoger. Al entrar en un subtema se muestran unos títulos que podemos ir clicando para abrir textos cortos y claros o una página con textos más largos y botones o frases que al hacerles clic, nos abren una imagen que podemos cerrar con un botón rojo central. La interfaz es clara e intuitiva, con botones de atrás en los subtemas, es fácil de seguir para el usuario.

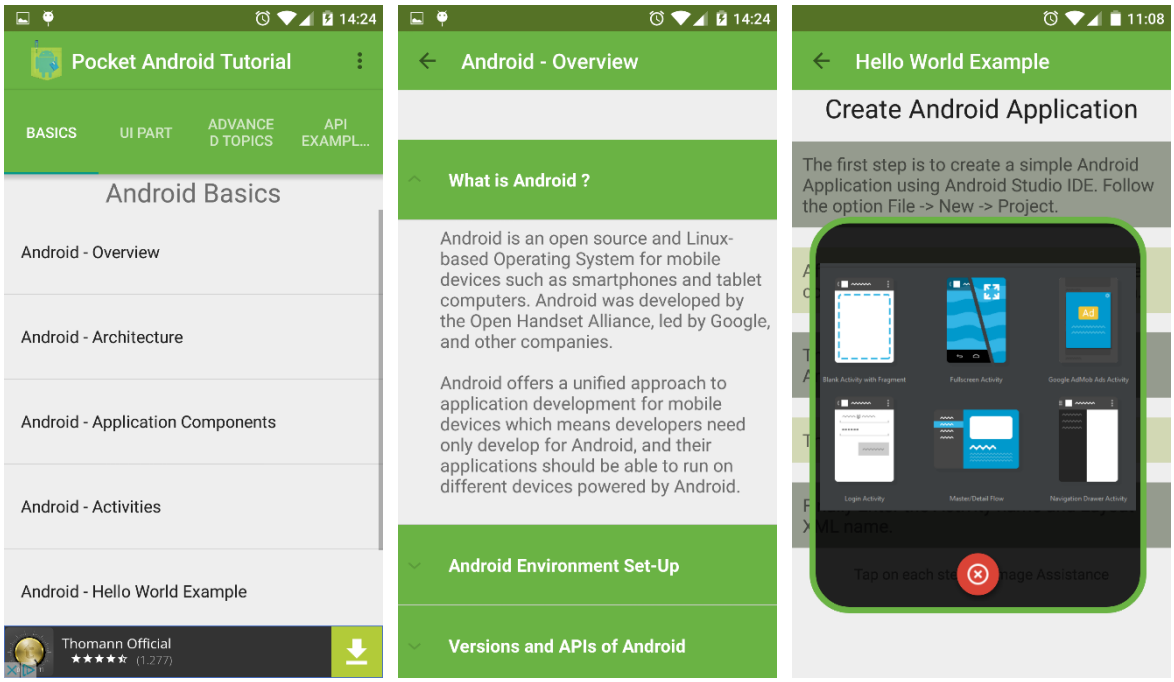

Fig. 2.9. Imágenes de la aplicación “Pocket Android Tutorial Free”

Ventajas	Desventajas
• Interfaz usable e intuitiva • Colores acordes al tema • Temas cortos y fáciles de seguir • Imágenes mostradas a parte	• Publicidad a pie de página que distrae al usuario

Tabla 2.4. Tabla ventajas y desventajas de la aplicación “Pocket Android Tutorial Free”

[4] Fig. 2.8. Icono de la aplicación “Tutorials for Android”

2.1.5. Tutorial for learning Android [5]

Cuando se abre la aplicación, encontramos los temas listados que al hacerles clic nos llevan a subtemas que, como en la aplicación anterior, al abrirlos muestra la teoría en forma de texto corto, con alguna imagen. Además, tiene ejercicios por ejemplo el punto 7 dónde te explica paso a paso cómo empezar con Android Studio. Tiene un menú lateral izquierdo que tiene poco sentido, ya que al abrirlo sólo está la Home, algo que por defecto ya se mostraba. Puede que esté implementado para futuras ampliaciones dónde añadirán más opciones. Tarda un poco en cargar depende qué texto en los subtemas.

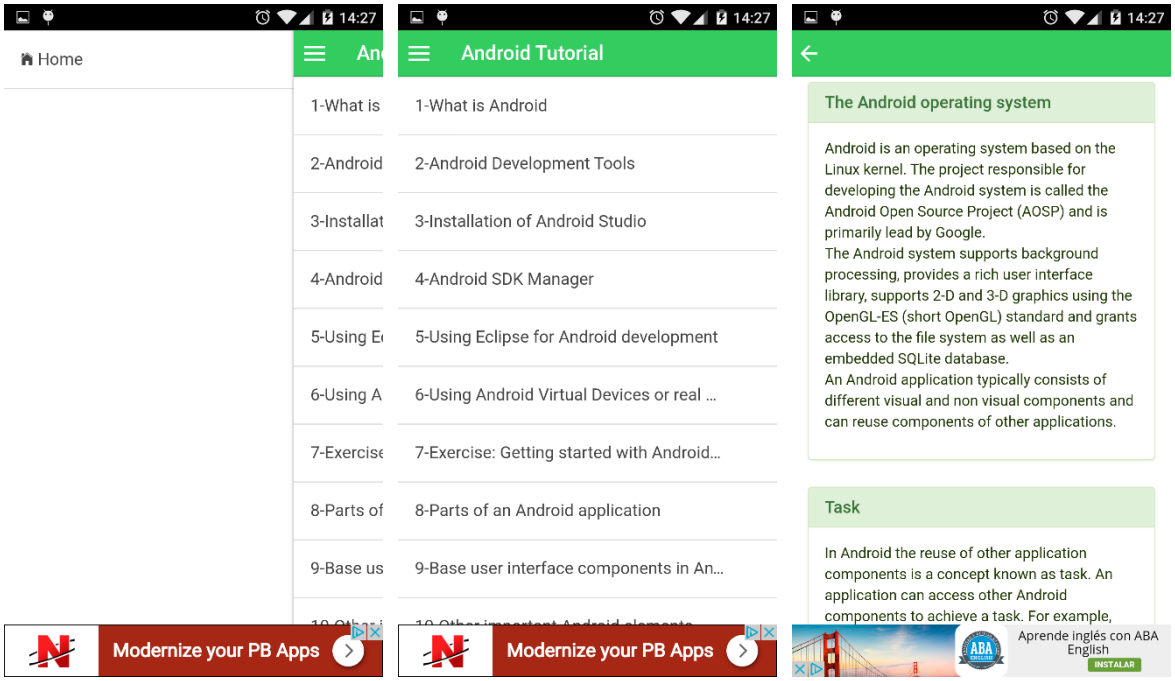

Fig. 2.11. Imágenes de la aplicación “Tutorial for learning Android”

Ventajas	Desventajas
• Interfaz usable, fácil de seguir • Colores acordes al tema • Dispone de teoría • Ejercicios prácticos	• Menú lateral izquierdo con poco sentido • Publicidad a pie de página que distrae al usuario • Lentitud en mostrar el texto

Tabla 2.5. Tabla ventajas y desventajas de la aplicación “Tutorials for learning Android”

[5] Fig. 2.10. Icono de la aplicación “Tutorials for Android”

2.1.6. Tutorials Android Developer [6]

La pantalla principal de esta aplicación está compuesta por los temas en verde y los subtemas en gris. Al elegir un subtema, lleva a unas pantallas con textos un poco largos y con los botones de volver al menú y siguiente abajo del todo, lo que dificulta al usuario seguir la línea de las páginas. Cuando se muestran imágenes, éstas no están adaptadas al dispositivo, por lo que hay que hacer scroll horizontal para poder verlas.

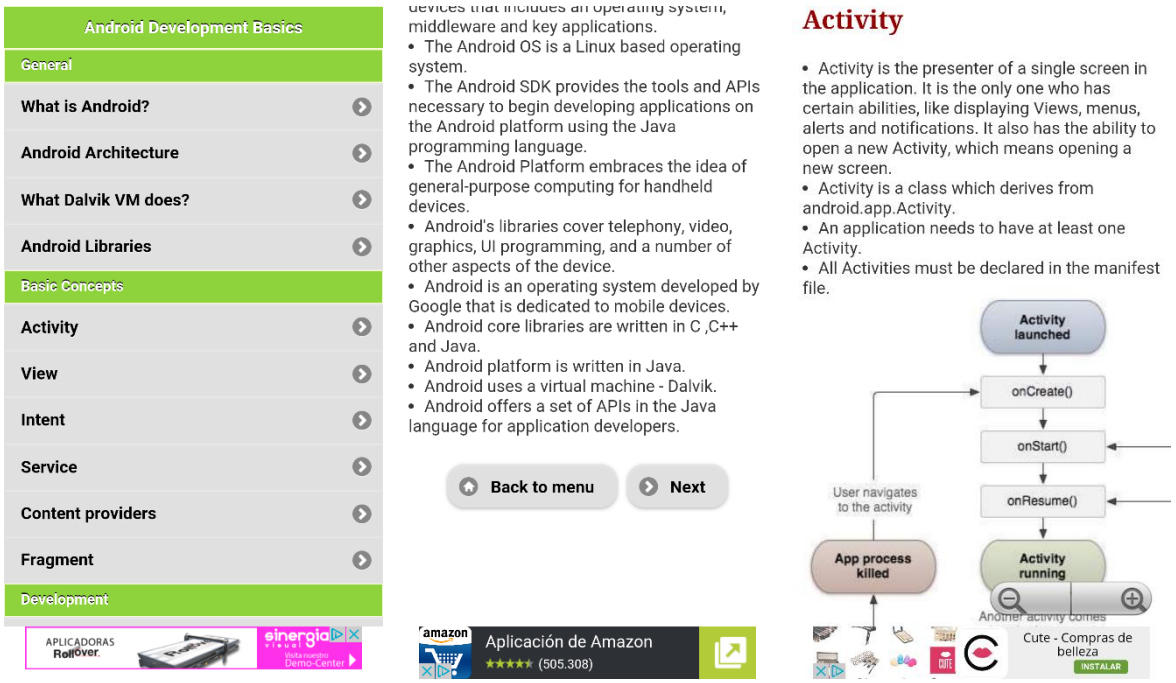

Fig. 2.13. Imágenes de la aplicación "Tutorials Android Developer"

Ventajas	Desventajas
• Dispone de teoría	• Interfaz poco intuitiva • Colores disonantes • Publicidad a pie de página que distrae al usuario • Difícil de seguir, temas largos (mucho scroll) • Elementos no adaptados al dispositivo

Tabla 2.6. Tabla ventajas y desventajas de la aplicación "Tutorials Android Developer"

[6] Fig. 2.12. Icono de la aplicación "Tutorials Android Developer"

2.1.7. Tutorials for Android [7]

Al entrar en la aplicación se muestran 3 botones: Start Tutorials, Questions/Answers y About. Si se aprieta Start Tutorials, no lleva a una lista de temas, que a su vez al clicarlos nos lleva a sus subtemas y éstos a su página dónde se muestra el texto de la teoría. Además, cuenta con unos botones en la parte superior derecha para pasar directamente al siguiente subtema o al anterior. En conjunto, la aplicación es fácil de seguir. En el apartado de Questions/Answers hay 50 preguntas frecuentes sobre Android con sus respectivas respuestas.

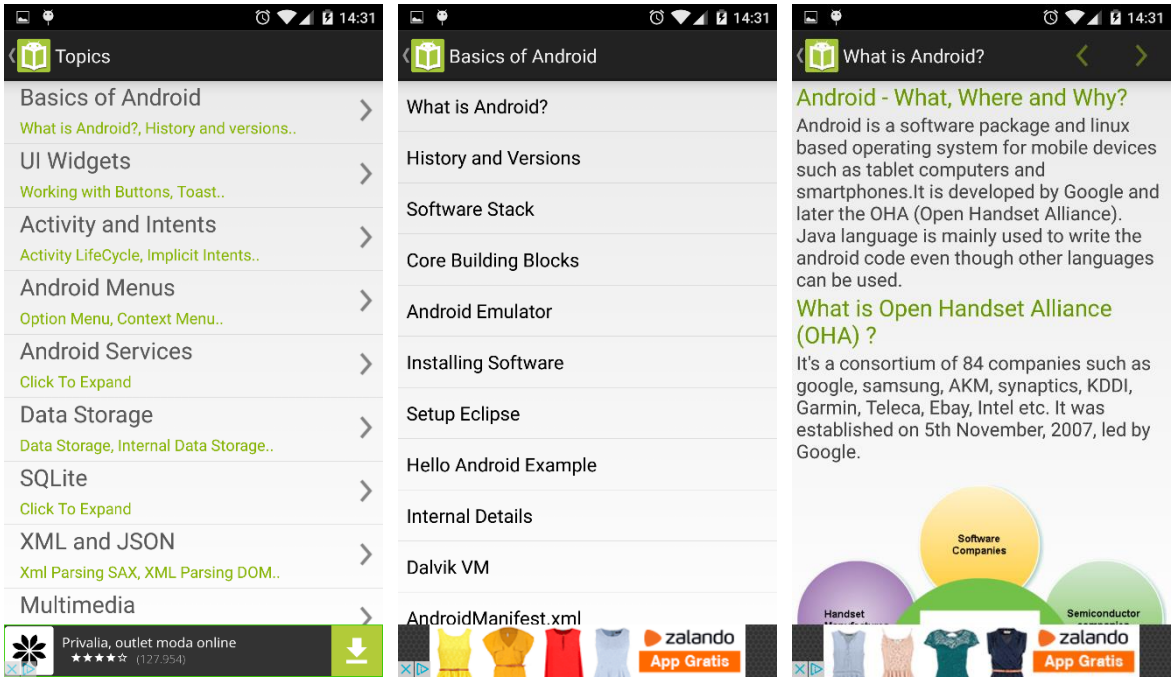

Fig. 2.15. Imágenes de la aplicación “Tutorials for Android”

Ventajas	Desventajas
• Interfaz fácil e intuitiva • Colores acordes al tema • Dispone de teoría • Temas cortos y claros • Diseño adaptable al dispositivo	• Publicidad a pie de página que distrae al usuario

Tabla 2.7. Tabla ventajas y desventajas de la aplicación “Tutorials for Android”

[7] Fig. 2.14. Icono de la aplicación “Tutorials for Android”

2.1.8. Tabla comparativa

A continuación, se muestra una tabla comparativa entre las aplicaciones de las que se ha realizado el estudio. Algunos de los apartados se han valorado de forma subjetiva después de haber estudiado el Material Design de Google y como experiencia en usuaria consumidora de aplicaciones de diversa temática.

							
Interfaz usable	X			X	X		X
Diseño adecuado al tema	X			X	X	X	X
Teoría		X	X	X	X	X	X
Código Java	X						
Seguimiento de temas							
Sin publicidad	X		X				

Tabla 2.8. Tabla comparativa de características entre las aplicaciones parecidas

2.1.9. Conclusiones del Estudio Previo

Después de estudiar éstas 7 aplicaciones, se puede asegurar que lo importante es disponer de una interfaz usable y fácil de seguir para el usuario final, con un diseño y colores adecuados al tema, ya que por ejemplo al verde se le relaciona directamente con Android o al azul con la enseñanza. La teoría debe estar correctamente concentrada, sin ser muy larga y tediosa, dividida en temas y subtemas. Es bueno poner ejercicios para hacer fuera de la aplicación o enseñar código Java para orientar al programador. La publicidad en esta clase de aplicaciones está mal vista, ya que distrae mucho al usuario de la lectura y lo desconcentra. Ninguna aplicación dispone de seguimiento de temas o sistema de Login.

3. Introducción a Android

A continuación, se hará una pequeña introducción a Android, explicando qué es, las versiones de Android que existen en el mercado, el nivel de la API, los componentes y la estructura de un proyecto Android y qué son el APK y la firma digital.

3.1. Sistema operativo Android

Android es un sistema operativo que se emplea en dispositivos móviles, presente también en tabletas, televisores y automóviles. Ha sido creado por Google y está basado en Linux, por lo que es de código abierto.

Es el sistema operativo móvil más usado, por delante de iOS y Windows Phone, más del 80% de los smartphones lo usan. Tiene una tienda llamada Google Play dónde se pueden descargar las aplicaciones para Android.

3.2. Versiones Android

Existen 13 versiones de Android distribuidas hasta la fecha desde que apareció Apple Pie en 2007, serán 14 con la llegada de Nougat (Android 7.0) este año. Todas reciben el nombre de un postre en inglés y en orden alfabético.

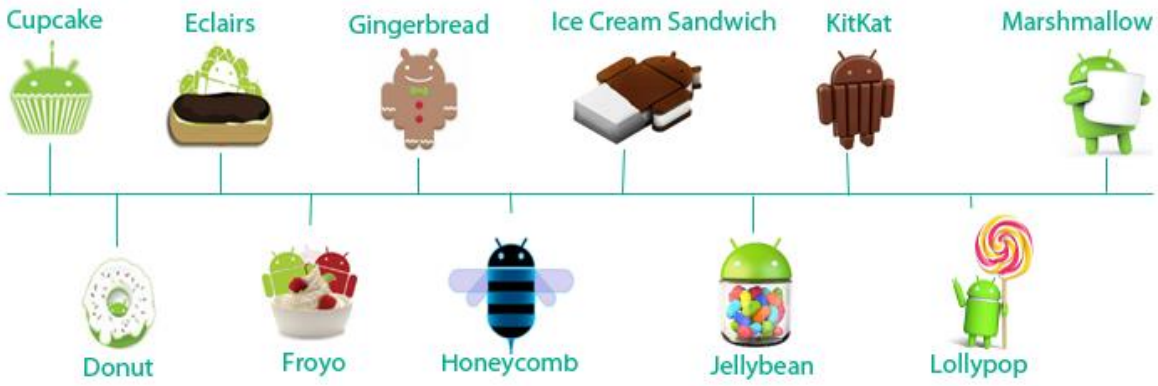

Fig. 3.1. Imagen de logotipos de algunas versiones de Android

Apple Pie (Android 1.0) Anunciado en 2007 y lanzada de forma comercial con el HTC Dream en septiembre de 2008.

Banana Bread (Android 1.1) Lanzado en febrero de 2009.

Cupcake (Android 1.5) Lanzado en abril de 2009.

Donut (Android 1.6) Lanzado en septiembre de 2009, significó la llegada de equipos como el Sony Ericsson Xperia X10.

Eclair (Android 2.0) Lanzado en octubre de 2009, fue el sistema operativo de teléfonos como el Motorola Droid.

Froyo (Android 2.2) Lanzado en mayo de 2010, vino acompañado de equipos como el HTC Inspire, el LG Optimus M y el primer Samsung Galaxy S.

Gingerbread (Android 2.3) Lanzado en diciembre de 2010, trajo smartphones como el LG Optimus One y el Samsung Google Nexus S.

Honeycomb (Android 3.0) Lanzado en febrero de 2011, fue un sistema exclusivo para tabletas y se estrenó en la Motorola Xoom.

Ice Cream Sandwich (Android 4.0) Lanzado en octubre de 2011, fue uno de los cambios más importantes de Android, y fue compatible con tabletas y smartphones.

Jelly Bean (Android 4.1) Anunciado en junio de 2012, y su primer dispositivo fue el Nexus 7, si bien después llegaron el Nexus 4 de LG y el Nexus 10 de Samsung.

KitKat (Android 4.4) Lanzado en octubre de 2013, trajo consigo a smartphones como el Galaxy S5, el Motorola Moto X y el Zony Xperia Z2.

Lollipop (Android 5.0) Lanzado en noviembre de 2014, introdujo Material Design, un nuevo diseño en las aplicaciones de Android.

Marshmallow (Android 6.0) Lanzado en septiembre de 2015, continúa la propuesta de Android Lollipop.

Nougat (Android 7.0) En proceso de lanzamiento.

3.2.1. Nivel de API

El nivel de la API es un valor numérico que identifica la versión de la plataforma Android. Las actualizaciones de API se diseñan de manera que la siguiente API siga siendo compatible con las versiones anteriores, añadiendo elementos o funcionalidades. Como se verá más adelante, al desarrollar una aplicación en Android se debe indicar en el Manifest la versión mínima y máxima de API.

3.2.2. Versiones de las plataformas

En la tabla 3.1 y en el gráfico 3.2, se podrá apreciar el número aproximado de dispositivos que ejecutan una versión determinada de Android:

Version	Codename	API	Distribution
2.2	Froyo	8	0.1%
2.3.3 - 2.3.7	Gingerbread	10	1.7%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	1.6%
4.1.x	Jelly Bean	16	6.0%
4.2.x		17	8.3%
4.3		18	2.4%
4.4	KitKat	19	29.2%
5.0	Lollipop	21	14.1%
5.1		22	21.4%
6.0	Marshmallow	23	15.2%

Tabla 3.1. Tabla de versiones de API en dispositivos Android [1]

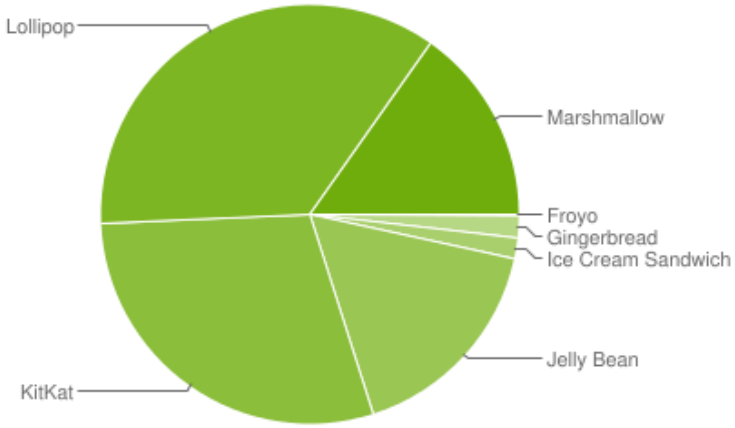

Fig. 3.2. Gráfico de versiones de Android en dispositivos móviles [1]

La información se recopila mediante Google Play y fue recogida durante 7 días concluyendo el 1 de agosto de 2016. Cualquier versión con menos del 0,1% de distribución no se muestra en la tabla.

Como se puede observar en la gráfica, la mayoría de los dispositivos utilizan las versiones Lollipop (35.5%) y KitKat (29.2%), seguidas de Jelly Beans (16.7%) y Marshmallow (15.2%). Se recomienda desarrollar a partir del nivel de API 16 (Jelly Beans), ya que para APIs más pequeñas se pueden llegar a tener problemas de compatibilidad con los Fragments o ActionBars. Se deben usar las librerías de soporte de Google para que la compatibilidad entre versiones sea lo más alta posible.

3.2.3. Medida y densidad de las pantallas

En la siguiente tabla, se podrá apreciar el número aproximado de dispositivos Android que contienen una configuración de pantalla concreta, definida por una combinación de dimensión y densidad de pantalla.

La información se recopila mediante Google Play y fue recogida durante 7 días concluyendo el 1 de agosto de 2016. Cualquier configuración de pantalla con menos del 0,1% de distribución no se muestra en la tabla.

	ldpi	mdpi	tvdpi	hdpi	xhdpi	xxhdpi	Total
Small	1.8%						1.8%
Normal		3.8%	0.1%	40.0%	27.3%	15.5%	86.7%
Large	0.2%	4.3%	2.1%	0.5%	0.5%		7.6%
Xlarge		2.9%		0.3%	0.7%		3.9%
Total	2.0%	11.0%	2.2%	40.8%	28.5%	15.5%	

Tabla 3.2. Tabla de medida y densidad de pantalla en dispositivos Android [1]

A continuación, se muestra de forma más gráfica con 2 gráficos, uno para la medida de pantalla y otro para la densidad de pantalla:

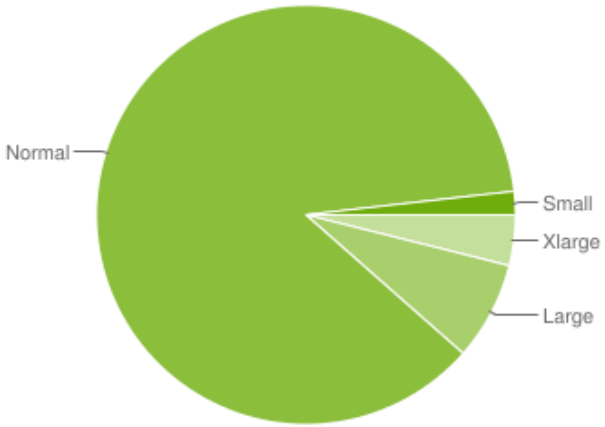

Fig. 3.3. Gráfico de medidas de pantallas en dispositivos Android [1]

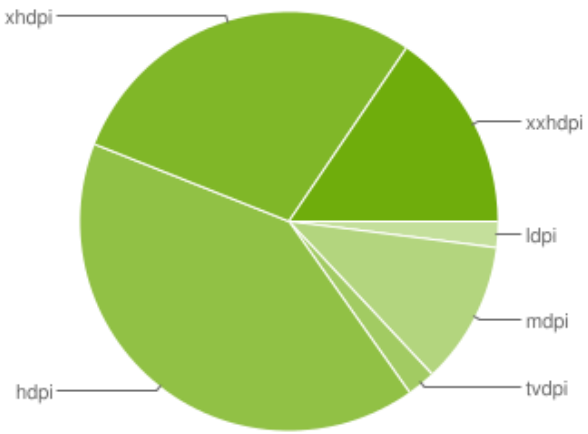

Fig. 3.4. Gráfico de densidades de pantallas en dispositivos Android [1]

Como se observa en la tabla y en los gráficos, la mayoría de dispositivos Android usan una pantalla de medida normal (86.7%) donde la resolución habitual es la hdpi (40.0%). Se destaca el crecimiento de los dispositivos con pantallas más grandes con xhdpi (27.3%) y xxhdpi (15.5%), dejando cada vez más atrás la medida pequeña mdpi (3.8%).

3.3. Componentes de una aplicación

A continuación, se detallarán los aspectos más importantes de los componentes básicos de una aplicación Android.

3.3.1. Manifest

El archivo AndroidManifest.xml es un fichero indispensable en un proyecto Android, que cada aplicación deberá contener en su directorio raíz y con el mismo nombre. Muestra información esencial acerca de la App, el sistema Android accede a él antes de compilar cualquier línea de código del proyecto. Cumple las siguientes funciones:

- Establece el nombre del package de la aplicación, para identificarla de manera única.
- Se declara el nombre de la aplicación, el icono y el tema.
- Permite definir componentes de la aplicación como Activities, Services, etc.
- Determina la Activity que será lanzada inicialmente.

- Se declaran los permisos que necesita la aplicación para realizar determinadas tareas, como el acceso a contactos del teléfono o realizar llamadas telefónicas.

La estructura general es la siguiente:

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

<manifest>       //Elemento raiz del fichero. Debe contener un elemento <application>

    <uses-permission />      //Permisos que debe aceptar el usuario al instalar la App
    <permission />           //Permiso de seguridad para limitar acceso a un componente o funcionalidad especifica
    <permission-tree />      //Nombre base para un árbol de permisos (ej: com.academiaandroid.proyecto.audio)
    <permission-group />     //Nombre para un grupo lógico de permisos afines
    <instrumentation />      //Permite monitorizar interacciones de la App con el sistema
    <uses-sdk />             //Compatibilidad de la App con una o mas versiones Android (nivel de APIs soportadas)
    <uses-configuration />   //HW y SW especifico que necesita la App (por ejemplo, un teclado fisico)
    <uses-feature />         //HW o SW que utiliza la App (ej: bluetooth, camara...)
    <supports-screens />     //Tamaños de pantallas soportadas por la App
    <compatible-screens />   //Configuraciones de pantallas compatibles con la App
    <supports-gl-texture />  //Formatos de compresión de textura GL soportados

    <application>            //Declaración de la Aplicación

        <activity>           //Declaracion de Activity
            <intent-filter>  //Especifica tipo de Intents a los que puede responder
                <action />
                <category />
                <data />
            </intent-filter>
            <meta-data />
        </activity>

        <activity-alias>      //Alias para la Activity
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </activity-alias>

        <service>            //Declaración de Servicio
            <intent-filter> . . . </intent-filter>
            <meta-data/>
        </service>

        <receiver>           //Declaración de Broadcast Receiver
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </receiver>

        <provider>           //Declaración de Content Provider
            <grant-uri-permission />
            <meta-data />
            <path-permission />
        </provider>

        <uses-library />     //Especifica libreria compartida que debe ser enlazada

    </application>

</manifest>
```

Fig. 3.5. Estructura general de AndroidManifest.xml

Permisos

Cada aplicación que se ejecuta en Android, lo hace desde un entorno limitado. Si se desea acceder a recursos del sistema o de otra aplicación, se necesitará solicitar permisos de manera explícita, y dependiendo del tipo de permiso, el sistema lo concederá de manera automática o solicitará al usuario su aprobación. A continuación, se muestran algunos ejemplos:

```xml
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.GET_ACCOUNTS"/>
<uses-permission android:name="android.permission.USE_CREDENTIALS"/>
<uses-permission android:name="android.permission.READ_OWNER_DATA"/>
<uses-permission android:name="android.permission.SET_WALLPAPER"/>
<uses-permission android:name="android.permission.DEVICE_POWER"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.CALL_EMERGENCY_NUMBER"/>
```

3.3.2. View

Son los elementos que componen la interfaz gráfica de una aplicación, como botones, listas desplegables, cuadros de texto, etc. Se pueden definir en los archivos xml o dinámicamente en código Java.

3.3.3. Activity

Es el componente principal de la interfaz gráfica de una aplicación Android. A cada Activity se le asigna una ventana en la cual se dibuja la interfaz de usuario, que a su vez se construye con componentes View. De forma resumida, podría decirse que es cada una de las pantallas que se crean en una aplicación Android.

Por lo general, una aplicación está formada por diferentes Activities, que están más o menos ligadas entre sí. Cuando se suceden varias, éstas se van almacenando en una pila mediante el mecanismo de LIFO (Last In First Out: la última que entra en la pila es la primera que sale) y cuando el usuario pulsa el botón atrás, se extrae la Activity actual de la pila y se reanuda la Activity anterior.

Una Activity tiene esencialmente cuatro estados:

- Activa

  Si la Activity está en primer plano, se dice que está activa o ejecutándose.

- En pausa

  Si la Activity ha perdido el foco, pero sigue siendo visible (por ejemplo, cuando se inicia otra Activity que no ocupa la pantalla entera), se dice que está en pausa o detenida. Una Activity detenida sigue permaneciendo en memoria, pero puede ser 'matada' o interrumpida por el sistema en condiciones de baja memoria.

- Parada

  Si la Activity deja de ser visible (por ejemplo, cuando se inicia otra Activity que ocupe la pantalla entera), se dice que está parada. Aun así, sigue permaneciendo en memoria, aunque puede ser 'matada' por el sistema en condiciones de baja memoria.

- Eliminada

  Si una Activity está pausada o parada, el sistema puede eliminarla de memoria matando el proceso o mandándole la señal de que finalice. Cuando vuelva a ser visible por el usuario, ésta debe ser completamente reiniciada.

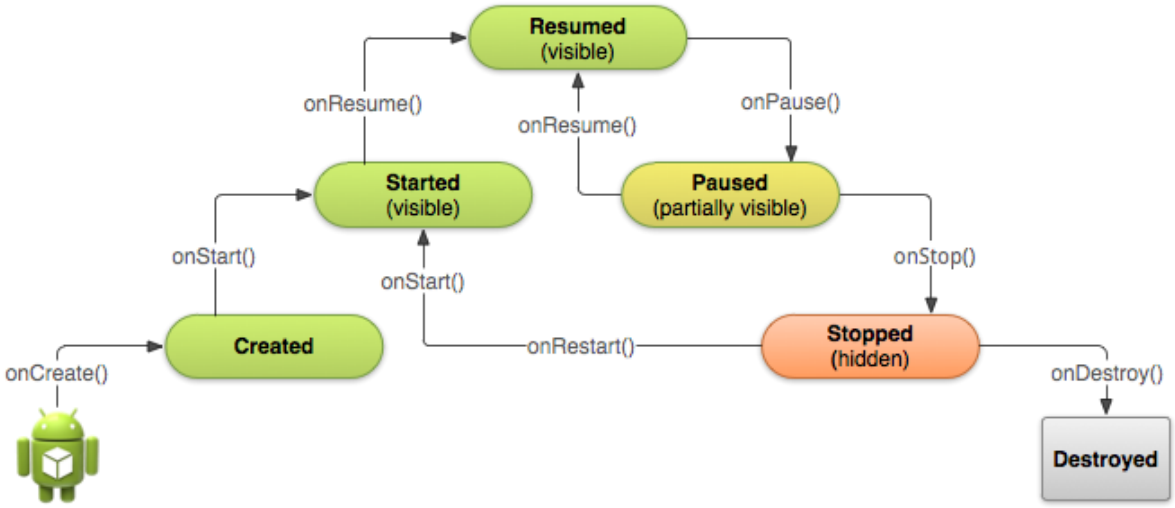

Fig. 3.6. Gráfico simplificado del ciclo de vida de una Activity [2]

A nivel de código seria:

```
1  public class Activity extends ApplicationContext
2  {
3       protected void onCreate(Bundle savedInstanceState);
4
5       protected void onStart();
6
7       protected void onRestart();
8
9       protected void onResume();
10
11      protected void onPause();
12
13      protected void onStop();
14
15      protected void onDestroy();
16 }
```

Fig. 3.7. Código de una Activity con sus métodos de control de estado

Cada Activity que se crea, se debe definir en el AndroidManifest.xml con la etiqueta <activity>. Para iniciar una Activity, utilizaremos `Context.startActivity()` o `Context.startActivityForResult()` (cuando queramos que se devuelva algún resultado).

En el siguiente diagrama se describe de forma gráfica los distintos estados por los que puede pasar una Activity:

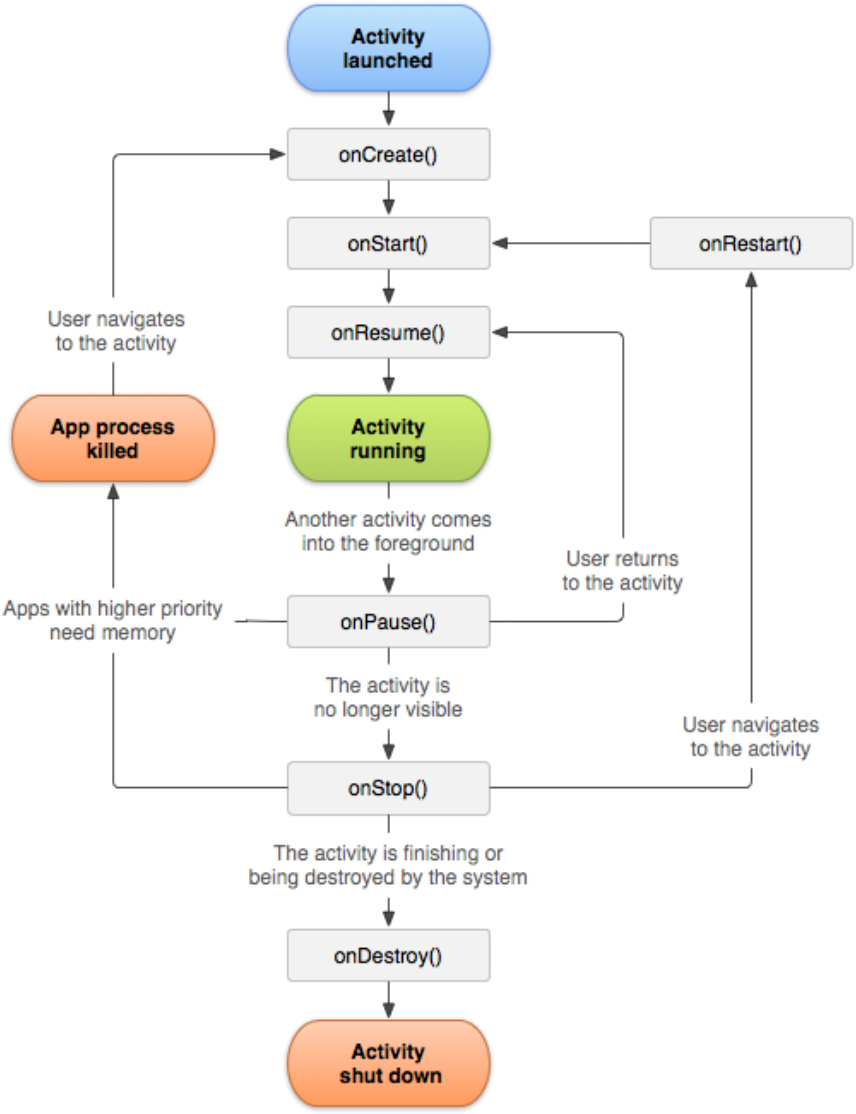

Fig. 3.8. Ciclo de vida de una Activity [3]

Consta de dos partes bien diferenciadas:

- La parte lógica es el código, la clase Java encargada de implementar las funcionalidades de la aplicación.

- La parte gráfica es un archivo xml situado en la carpeta *nombreproyecto/res/layout*, dónde se definen los componentes gráficos que formarán la interfaz gráfica de usuario.

Existen otros componentes que funcionan dentro del ámbito de una Activity llamados Fragments, que amplían y enriquecen las posibilidades de interacción con el usuario.

3.3.4. Frament

Es un componente que funciona dentro del ámbito de una Activity. Su finalidad es la de ampliar la lógica utilizada para navegar entre pantallas o Activities, pudiendo definir varios Fragments dentro de una misma Activity, interactuando entre ellos y haciendo una UI más dinámica. Se introdujeron en la versión Android 3.0 (API 11).

Todo Fragment debe estar embebido dentro de una Activity, por lo que el ciclo de vida de un fragment estará ligado al de la Activity dónde se ha definido. Si la Activity se encuentra en ejecución, se podrá manipular de forma independiente cada Fragment, y si la Activity se pausa o se destruye, los Fragments asociados también se pausarán o destruirán respectivamente.

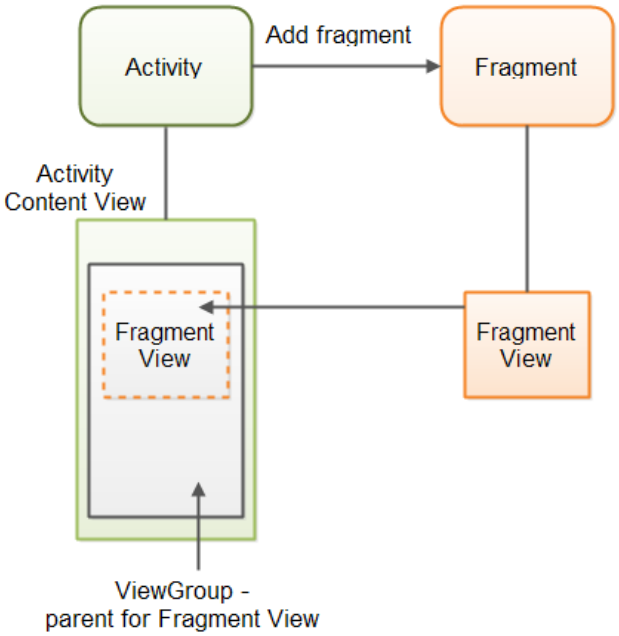

Fig. 3.9. Gráfico de añadir un Fragment a una Activity

Al igual que una Activity, el Fragment tiene un ciclo de vida:

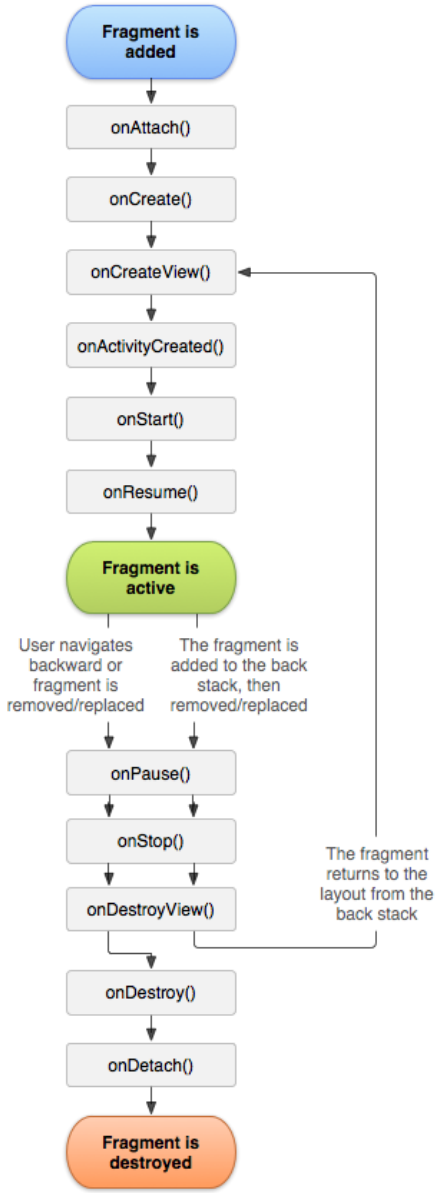

Fig. 3.10. Ciclo de vida de un Fragment [4]

En todo ciclo de vida de un Fragment, se recomienda utilizar las siguientes funciones:

- onCreate(): El sistema llama a esta función cuando se crea el Fragment.

- onCreateView(): El sistema llama a esta función cuando se dibuja por primera vez el Fragment en la interfaz de usuario.

- onPause(): El sistema llama a esta función cuando el usuario deja de utilizar el Fragment (no implica que éste sea destruido).

Los Fragments permiten la división de una Activity, posibilitando la modificación de la apariencia de una Activity en tiempo de ejecución:

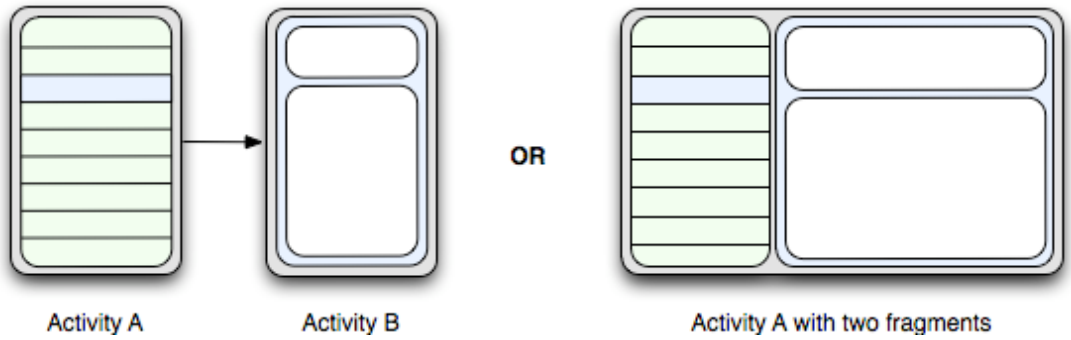

Fig. 3.11. Uso de Fragments en una Activity

Ventajas del uso de Fragments

- Proporciona diseños más dinámicos y flexibles para pantallas más grandes.
- No son necesarios cambios muy profundos en la jerarquía de vistas.
- Cada Fragment definido en una Activity es independiente del resto de Fragments, y por lo tanto reutilizable.
- Facilita la tarea de desarrollo de Apps que funcionen correctamente tanto en tablets como en dispositivos móviles (multidispositivo).
- Un Fragment tiene su propio Layout y su propio ciclo de vida.

3.3.5. Intent

El Intent es el elemento básico de comunicación entre los componentes que describimos en la App. Mediante un intent se podrá llamar a una Activity, iniciar un servicio, enviar un mensaje broadcast, iniciar otra aplicación, abrir un navegador, etc. Su uso más importante es para iniciar Activities, por lo que se puede considerar como la unión entre Activities.

Los objetos Intent están formados por un paquete de información, que contiene datos de interés para el componente que la recibe, como la acción que será ejecutada y los datos necesarios para llevarla a cabo, e incluso permite enviar parámetros a una segunda Activity con el método *putExtra(tag, value):*

```
Intent intent = new Intent(this, AnotherActivity.class);
intent.putExtra("tag", "value");
startActivity(intent);
```

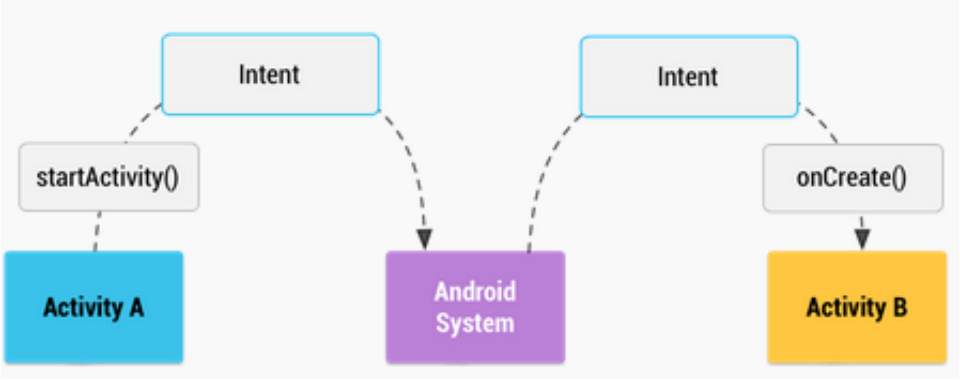

Fig. 3.12. Gráfico de Intent de una Activity

3.4. Estructura de un proyecto Android

Un proyecto Android puede contener varios módulos. Cada módulo corresponde a una aplicación o ejecutable diferente. Disponer de varios módulos en un mismo proyecto es útil cuando se quieran crear varias versiones de nuestra aplicación, por ejemplo, destinada a diferentes dispositivos (móvil, Wear, TV, Glass, etc.) o si se quiere tener ejecutables con niveles diferentes de SDK.

Se muestra en la siguiente página, un módulo base llamado app:

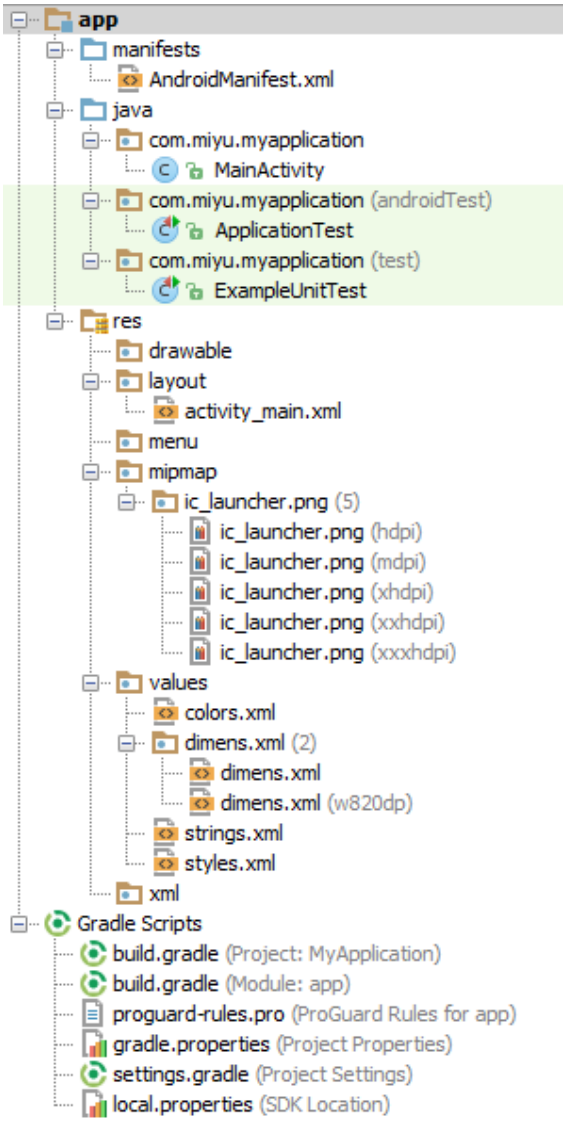

Fig. 3.13. Módulo base de un proyecto de Android

Cada módulo en Android está formado por un descriptor de la aplicación (manifest), el código fuente en Java (java), una serie de ficheros con recursos (res) y ficheros para construir el módulo (Gradle Scripts).

Elementos de un módulo Android:

- AndroidManifest.xml: Como ya hemos visto, este fichero describe la aplicación Android. Se define su nombre, paquete, icono, estilos, etc. Se indican las actividades, las intenciones, los servicios y los proveedores de contenido de la aplicación. También se declaran los permisos que requerirá la aplicación.

- java: Carpeta que contiene el código fuente de la aplicación. Se puede observar que los ficheros Java se almacenan en carpetas según el nombre de su paquete.

    - MainActivity: Clase Java con el código de la actividad inicial.

    - ApplicationTest: Clase Java pensada para insertar código de testeo de la aplicación utilizando el API JUnit.

- res: Carpeta que contiene los recursos usados por la aplicación.

    - drawable: En esta carpeta se almacenan los ficheros de imágenes (JPG o PNG) y descriptores de imágenes en XML.

    - layout: Contiene ficheros XML con vistas de la aplicación. Las vistas permiten configurar las diferentes pantallas que compondrán la interfaz de usuario de la aplicación. Se utiliza un formato similar al HTML usado para diseñar páginas web.

    - mipmap: Es una carpeta con la misma finalidad que res/drawable. La única diferencia es que, si se ponen los gráficos en mipmap, estos no son rescaldados para adaptarlos a la densidad gráfica del dispositivo donde se ejecuta la aplicación, sino que se buscará en las subcarpetas el gráfico con la densidad más parecida y se utilizará directamente. Es recomendable guardar los iconos de aplicación en esta carpeta. En el proyecto se ha incluido el fichero ic_launcher.png que será utilizado como icono de la aplicación.

    - menu: (añadido al proyecto) Carpeta con ficheros XML de los menús de cada actividad.

    - values: También se utilizan ficheros XML para indicar valores usados en la aplicación, de esta manera se pueden cambiar desde estos ficheros sin necesidad de ir al código fuente. En colors.xml se definen los tres colores primarios de la aplicación. En dimens.xml, se define el margen horizontal y

vertical por defecto. En el fichero strings.xml, se definen todas las cadenas de caracteres de la aplicación, lo que hará que sea muy sencillo traducir una aplicación a otro idioma. Finalmente, en styles.xml, se pueden definir los estilos y temas de la aplicación.

- xml: (añadido al proyecto) Carpeta con otros ficheros XML requeridos por la aplicación.

- Gradle Scripts: En esta carpeta se almacenan una serie de ficheros Gradle que permiten compilar y construir la aplicación. Algunos hacen referencia al módulo app y el resto son para configurar todo el proyecto. El fichero más importante es build.gradle (Module:app), dónde se configuran las opciones de compilación del módulo:

3.5. APK y firma digital

Para poder distribuir una aplicación Android y que ésta pueda ser ejecutada en un dispositivo, se debe generar un archivo APK, que no es más que un archivo compilado ejecutable con extensión *.apk*.

Es un formato parecido al JAR para Java (Java Archive). Es básicamente un fichero comprimido ZIP que empaqueta los componentes del proyecto Android, como el AndroidManifest, las clases, los recursos… Para que éstos puedan ser desplegados como aplicación y pueda ser ejecutada en un emulador Android. El fichero en sí, se puede abrir para ver sus componentes con un programa de compresión de archivos como 7-Zip, WinRAR o similar.

Si además se quiere publicar la aplicación en la Play Store de Google, se debe firmar digitalmente el APK generado.

4. Entorno de desarrollo (Herramientas)

En este capítulo se hará un pequeño resumen del entorno de desarrollo y las herramientas usadas a lo largo del proyecto.

4.1. Java para Android

Fig. 4.1. Icono Java for Android

Existen muchos lenguajes de programación para desarrollar en Android, pero Java es el lenguaje que se recomienda por parte de Google. Aunque la programación nativa de Android es en Java, Android no usa Java "puro". La escritura y el desarrollo de una aplicación Android es muy parecida a un proyecto Java normal, pero a la hora de compilar y ejecutar el código, existe una diferencia principal. Cuando Java compila, traduce el programa a una forma intermedia entre código y lenguaje máquina llamada Bytecode, que se ejecuta en una máquina virtual, y así, el programa funcionará en la plataforma específica siempre que se disponga de un Java Runtime Environment (JRE) para traducir el Bytecode. La forma en la que lo hace Android es parecida, pero cuando se instala la aplicación en el dispositivo, el Bytecode se convierte en código máquina específico y optimizado para ése dispositivo concreto. A este proceso se le conoce como compilación Ahead of Time (AoT) y es posible gracias a la máquina virtual Android Runtime (ART), lo que es ahora el antiguo Dalvik. Esta conversión del Bytecode de Java es lo que hace que escribir en Android sea menos "puro", ya que éste cambio limita la portabilidad de la aplicación, negando una de las premisas más extendidas de Java: "escribir una vez, ejecutar en todas partes".

Android adopta el paradigma de la Programación Orienta a Objetos (POO) de Java, y está diseñado para adaptarse a los conceptos de encapsulación, herencia y polimorfismo. Además, todos los objetos en Android heredan de la clase Object. Android sólo ofrece un subconjunto de las bibliotecas de Java, y además son única y exclusivamente para Android.

4.2. Android Studio

Fig. 4.2. Icono Android Studio

Android estudio es un entorno de desarrollo integrado para Android. Entre 2013 y 2014 reemplazó a Eclipse como IDE oficial para el desarrollo de aplicaciones Android. Es Open Source y está basado en el software IntelliJ de JetBrains y se publica de forma gratuita a través de la Licencia Apache 2.0. Posee plantillas de diseño comunes de Android, refactorización y arreglos rápidos, renderización en tiempo real, soporte para construcción basada en Gradle y herramientas Lint para detectar programas de rendimiento y compatibilidad de versiones. Está disponible para las plataformas Microsoft Windows, Mac OS X y GNU/Linux.

4.3. Android SDK

Para el desarrollo de aplicaciones Android es imprescindible instalar el Android SDK. SDK son las siglas de Sofware Development Kit, un kit de desarrollo de software con el cual podemos desarrollar aplicaciones y ejecutar un emulador del sistema Android de la versión correspondiente. Las aplicaciones Android nativas se desarrollan en el lenguaje Java usando este kit. Proporciona las bibliotecas API y las herramientas para la creación y depuración de aplicaciones Android. Permite poder utilizar nuestro dispositivo desde el ordenador en vez de usar el emulador.

Fig. 4.3. Icono Android SDK

4.4. Gradle

Una de las principales ventajas de usar Android Studio es la de poder hacer uso de Gradle. Gradle es una herramienta para automatizar la construcción de los proyectos, por ejemplo, manejar de forma fácil las dependencias, disponer de varios entornos para un mismo proyecto, para tareas de compilación, testing, empaquetado y despliegue o poder generar diferentes APKs de un mismo proyecto con diferentes configuraciones de UI entre otros.

Fig. 4.4. Icono Gradle

Gradle usa “Domain Specific Language” (DSL) basado en Groovy para declarar la forma de construir el proyecto. Para configurarlo simplemente se escribe código en Groovy, que es muy parecido a Java. Es muy flexible en para configurar, disponiendo de tareas ya armadas por defecto. Es un proyecto Open Source licenciado bajo Apache Software Licence (ASL).

Se hablará a continuación en los puntos 4.5, 4.6, 4.7, 4.8, 4.9, 4.10, 4.11 y 4.12 de algunas librearías Android importantes para el desarrollo del proyecto AndroidUp, añadidas al proyecto con la ayuda de Gradle.

4.5. Butter Knife

La librería Butter Knife, desarrollada y mantenida por Jake Wharton de Squre Inc. dispone de anotaciones que ayudan a los desarrolladores Android a instanciar las views de la Activity o del Fragment, relacionando los elementos de las vistas con el código, simplificándolo. También tiene anotaciones para manejar eventos como onClick(), onLongClick() entre otros.

Fig. 4.5. Icono Butter Knife

Se puede usar agregando la dependencia al archivo build.gradle del proyecto en Android Studio, añadiendo permiso de Internet en el AndroidManifest.xml, y es muy útil para situaciones donde el diseño de los layouts se vuelve complejo y el proceso de findViewByID() se torna repetitivo, complicando la lectura del código.

Gradle ref. `compile 'com.jakewharton:butterknife:8.0.1'`

4.6. Volley

Fig. 4.6. Icono Volley

Volley es una librería desarrollada por Google para optimizar el envío de peticiones HTTP desde las aplicaciones Android a servidores externos. Este componente actúa como interfaz de alto nivel, dejando al programador libre de la administración de hilos y procesos de parsing.

Es un cliente que facilita la comunicación de red en las apps, enfocado totalmente a peticiones, evitando así la creación de código repetitivo de

manejo de tareas asíncronas y facilitando el parseo de datos de flujo externo. Gestiona automáticamente los trabajos en segundo plano, procesa concurrentemente las peticiones y posee priorización y cancelación de peticiones.

Gradle ref. `compile 'com.mcxiaoke.volley:library:+'`

4.7. CardView

CardView hace referencia a una card. Las cards son un componente gráfico popularizado por Google Now y que ahora forman parte de Material Design, parecidas a un "post-it" y que contienen varios elementos como imágenes, textos, iconos, menús contextuales, acciones, etc.

Fig. 4.7. Ejemplo sencillo de CardView

Android SDK proporciona una implementación del componente card con el widget CardView.

Gradle ref. `compile 'com.android.support:cardview-v7:23.3.0'`

4.8. CircleImageView


Fig. 4.8. Ejemplo de CircleImageView

Librería Android que proporciona una ImageView circular perfecta para imágenes de perfil. Facilita la tarea de poner bordes redondos en las ImageView.

Creada por Henning Dodenhof.

Gradle ref. `compile 'de.hdodenhof:circleimageview:2.1.0'`

4.9. CircleProgressView

Librería Android que proporciona una vista animada de CircleView. Se utiliza como indicador de carga y para mostrar el progreso o los valores de manera circular.

Creada por Jakob Grabner.

Gradle ref. `compile 'com.github.jakob-grabner:Circle-Progress-View:v1.2.9'`

4.10. Picasso

Librería Android que permite la carga de imágenes externas en la aplicación, a menudo en una sola línea de código. Maneja automáticamente errores comunes en la carga de imágenes en Android, transformando imágenes complejas con el mínimo uso de memoria.

Creada por Square, Inc.

Gradle ref. `compile 'com.squareup.picasso:picasso:2.5.2'`

4.11. Java Mail

Fig. 4.9. Icono de JavaMail API

El API de JavaMail proporciona los elementos necesarios para construir aplicaciones de correo y mensajería de forma independiente al protocolo utilizado.

Creada por Oracle.

Gradle ref.

```
compile files('libs/mail.jar')
compile files('libs/activation.jar')
compile files('libs/additionnal.jar')
```

4.12. EventBus

Librería Android que proporciona un bus de eventos de publicación/subscripción optimizado para Android. Simplifica la comunicación entre componentes como Activities y Fragments y soluciona problemas con sus ciclos de vida.

Fig. 4.10. Icono de EventBus

Creada por Markus Junginger de greenrobot.

Gradle ref. `compile 'org.greenrobot:eventbus:3.0.0'`

4.13. GitHub

Fig. 4.11. Icono de GitHub

GitHub es una plataforma de desarrollo colaborativo de software para alojar proyectos utilizando el sistema de control de versiones Git. El código se almacena por defecto de forma pública, aunque también se puede guardar de forma privada, creando una cuenta de pago. GitHub sirve para alojar repositorios de código y brinda herramientas muy útiles para el trabajo en equipo a la hora de desarrollar proyectos. Utiliza el framework Ruby on Rails por GitHub Inc. Proporciona una wiki y una página web por cada proyecto, bifurcaciones o ramas del proyecto y funcionalidades de red social y trabajo colaborativo entre programadores.

Fig. 4.12. Icono de GitHub Desktop

Dispone de una herramienta de escritorio llamada GitHub Desktop con una interfaz gráfica en vez de trabajar con instrucciones de la terminal. Se puede ver el Historial de Push realizados sobre el repositorio y detecta cambios locales en el proyecto, dejando hacer push o pull al usuario mediante el botón Sync.

4.14. GitBook

GitBook es una herramienta para crear documentación de proyectos y libros técnicos usando Markdown y Git/GitHub. Permite incluir ejemplos y ejercicios interactivos en JavaScript para publicarlos online vía GitHub.

Fig. 4.13. Icono de GitBook

Es documentación fácilmente editable y abierta a contribuciones de forma transparente. GitBook está implementado usando node.js y crea documentos en distintos formatos como PDF, ebook o web.

4.15. Asana

Fig. 4.14. Icono de Asana

Asana es una aplicación web y móvil diseñada para mejorar la comunicación y colaboración en equipo. Tiene muchas funcionalidades, como espacios de trabajo, proyectos, tareas, etiquetas, notas, comentarios y un buzón que organiza y actualiza la información en tiempo real.

El producto está diseñado para facilitar a las personas y a los equipos la planificación y la gestión de sus proyectos y tareas. Cada equipo tiene un espacio de trabajo y a su vez los espacios de trabajo contienen proyectos y los proyectos contienen tareas.

4.16. JSON

Fig. 4.15. Icono de JSON

JSON (JavaScript Object Notation) es un formato de texto ligero para el intercambio de datos. JSON es un subconjunto de la notación literal de objetos de JavaScript, aunque debido a su amplia adopción como alternativa a XML, se considera un formato de lenguaje independiente.

Se usa en MongoDB para crear los documentos y las colecciones.

4.17. MongoDB

MongoDB es un sistema de base de datos NoSQL orientado a documentos, que permite a los esquemas cambiar rápidamente y de forma escalable cuando las aplicaciones evolucionan, proporcionado la funcionalidad que los desarrolladores esperan de las bases de datos tradicionales y aportando escalabilidad y reducción de costes.

Fig. 4.16. Icono de MongoDB

En lugar de guardar los datos en tablas como se hace en las bases de datos relacionales, MongoBD guarda estructuras de datos en documentos similares a JSON, con un esquema dinámico, usando una especificación llamado BSON. Esto hace que la integración de datos sea más fácil y rápida. Posee gran disponibilidad y alto rendimiento, tanto para lectura como para escritura, potenciando la computación en memoria.

Fue desarrollada por la compañía de software 10gen y es de código libre.

4.18. Robomongo

Robomongo es una herramienta multiplataforma con la que se puede administrar gráficamente las bases de datos NoSQL y se ha convertido en un importante aliado para los usuarios de MongoDB, ya que trabajar con la Shell por defecto puede ser en muchos casos complicado.

Robomongo toma la Shell y la integra en un entorno gráfico con toda su funcionalidad, añadiendo cosas nuevas como múltiples conexiones a bases de datos separadas por pestañas, resaltado de sintaxis y autocompletado de código y distintos modos de visualización de los resultados.

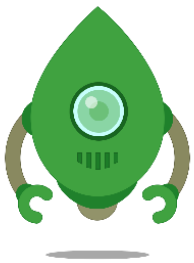

Fig. 4.17. Icono de Robomongo

4.19. PostMan

Postman es una extensión gratuita para el navegador Google Chrome, que permite probar servicios web de forma fácil. Basta con indicar la url, el método HTTP (POST, GET, etc.) y los parámetros de la petición.

Fig. 4.18. Icono de Postman

Conforme se usa Postman, automáticamente se van guardando las peticiones más recientes en el historial, para poder tenerlas a mano. También permite definir colecciones en donde se puede guardar una serie de métodos para reutilizarlos con más facilidad y poder compartirlos con otras personas en formato JSON. Además, se pueden crear ambientes y definir variables específicas para ellos. Si se usa autenticación en el servicio web, Postman permite utilizar identificación HTTP básica OAuth 2.0.

4.20. Digital Ocean

Digital Ocean es un proveedor estadounidense de servidores virtuales privados, ofrecen el alquiler de servidores cloud a precios muy asequibles. El término que utilizan para llamar a los servidores en la nube es “Droplets”. La compañía también provee discos duros SSD y virtualización KVM. Su API es RESTfull, utiliza OAuth y soporta IPv6.

Fig. 4.19. Icono de Digital Ocean

4.21. Eclipse

Eclipse es una plataforma de software compuesto por un conjunto de herramientas de programación de código abierto. Esta plataforma se usa para desarrollar Entornos de Desarrollo Integrados (IDE), como el IDE de Java llamado Java Development Toolkit (JDT) y el compilador (ECJ).

Fig. 4.20. Icono de Eclipse

Fue desarrollado originalmente por IBM, pero ahora pertenece a la Fundación Eclipse, una organización independiente que fomenta la comunidad de código abierto. Eclipse dispone de un editor de texto con analizador sintáctico y autocompletado. La compilación del código es en tiempo real y se le pueden añadir plugins.

4.22. Spring

Fig. 4.21. Icono de Spring

Spring es un framework para el desarrollo de aplicaciones, de código abierto para la plataforma Java, una alternativa para el modelo EJB (Enterprise JavaBean).

Spring comprende diversos módulos que proveen un gran rango de servicios, como el acceso a datos (herramienta de mapeo para objetos relacionales con bases de datos NoSQL), gestión de las transacciones (para APIs de gestión y transacciones de objetos Java), framework de acceso remoto, administración remota, modelo vista controlador, entre otros.

4.23. Maven

Maven es una herramienta de software para la gestión y construcción de proyectos Java, similar en funcionalidad a Apache Ant, pero tiene el modelo de configuración de construcción más simple, basado en el formato XML.

Fig. 4.23. Icono de Maven

Utiliza un Project Object Model (POM) para describir el proyecto de software a construir, sus dependencias con otros módulos y componentes externos y el orden de construcción de los elementos. Viene con objetivos predefinidos como la compilación del código y su empaquetado.

5. Planificación

La planificación se empezó a hacer en Asana:

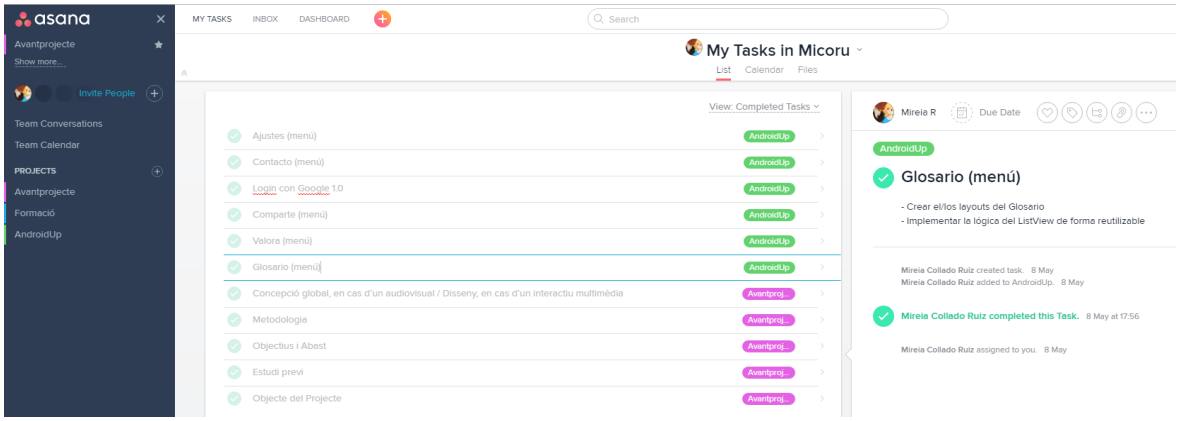

Fig. 5.1. Asana “Completed Tasks”

Sobre todo, en la parte del Anteproyecto fue de mucha ayuda, organizando ideas y tareas a hacer. Pero a partir de ahí hasta mayo, cada vez se hacía más difícil estar por poner tickets y aceptarlos cuando estuvieran hechos, ya que la dificultad en la implementación se fue incrementando y se pasaban días volviendo a abrir tickets porque no estaban del todo acabados.

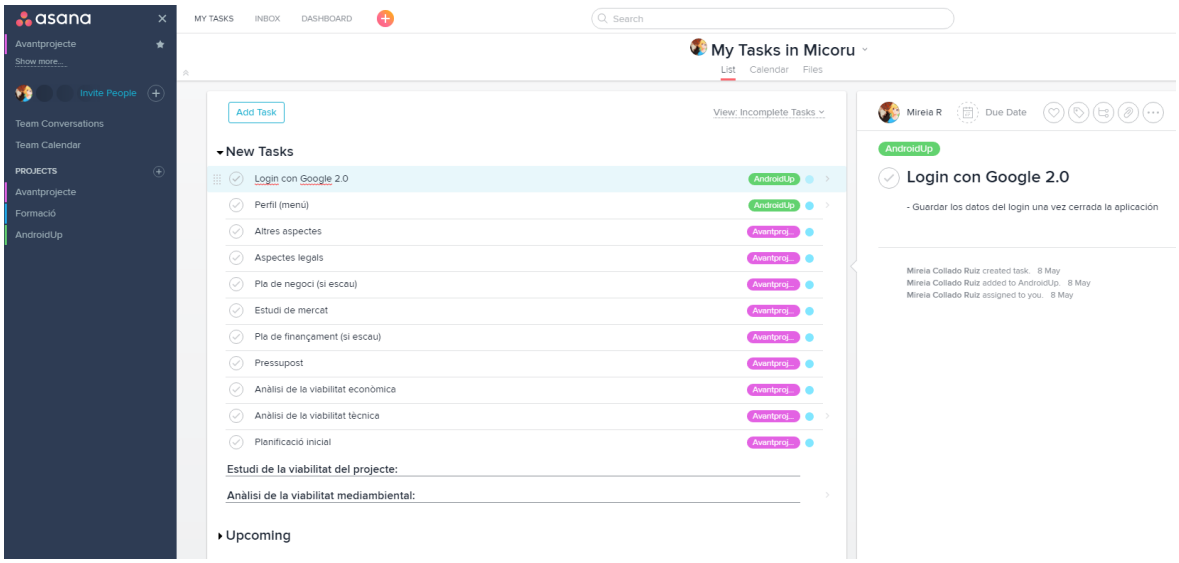

Fig. 5.2. Asana “Incomplete Tasks”

Se dejaron de subir tickets poco a poco, ya que la gran parte de las veces se abrían y cerraban el mismo día. Asana es una herramienta muy buena para el trabajo en equipo, pero no acaba de funcionar si es uno sólo, ya que constantemente se tienen que ir posteando y cogiendo los tickets que se acaban de poner. Lo ideal, sería un jefe de proyecto que postee los tickets y un equipo de programadores que cojan ésos tickets y los desarrollen.

Además, se empezó a hacer la documentación del proyecto en GitHub:

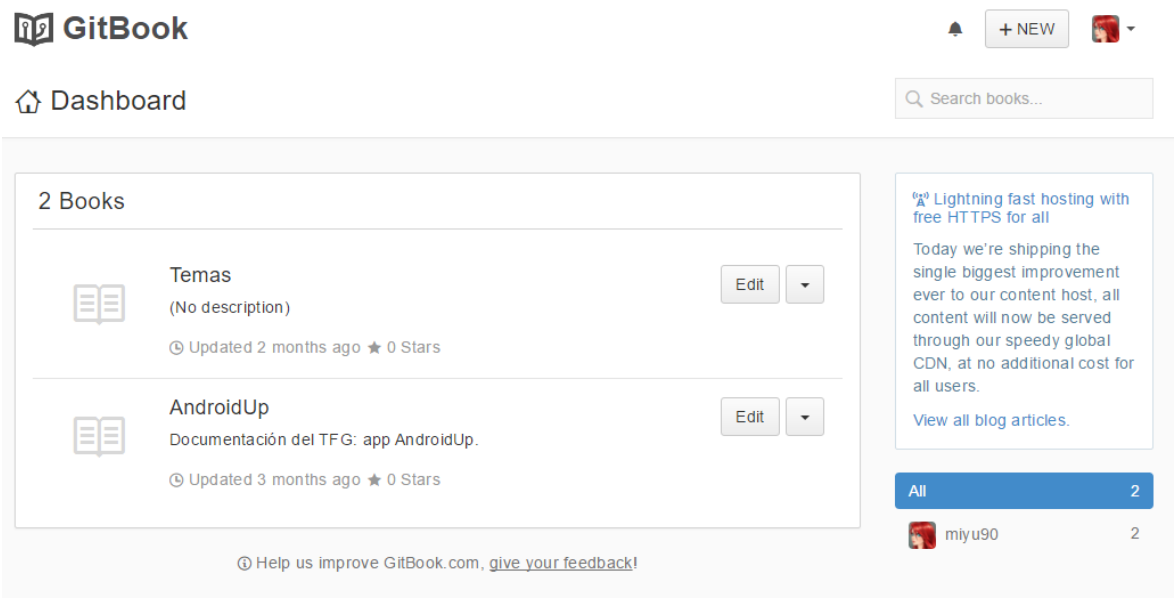

Fig. 5.3. Dashboard de GitBook

Hasta que se llegó a la conclusión de que la documentación final se debía de entregar en formato PDF y que estaría sujeta a formatos y estilos de Word. Entonces, GitBook se convirtió en un sitio para plasmar los problemas encontrados en la implementación, y alguna que otra tarea más:

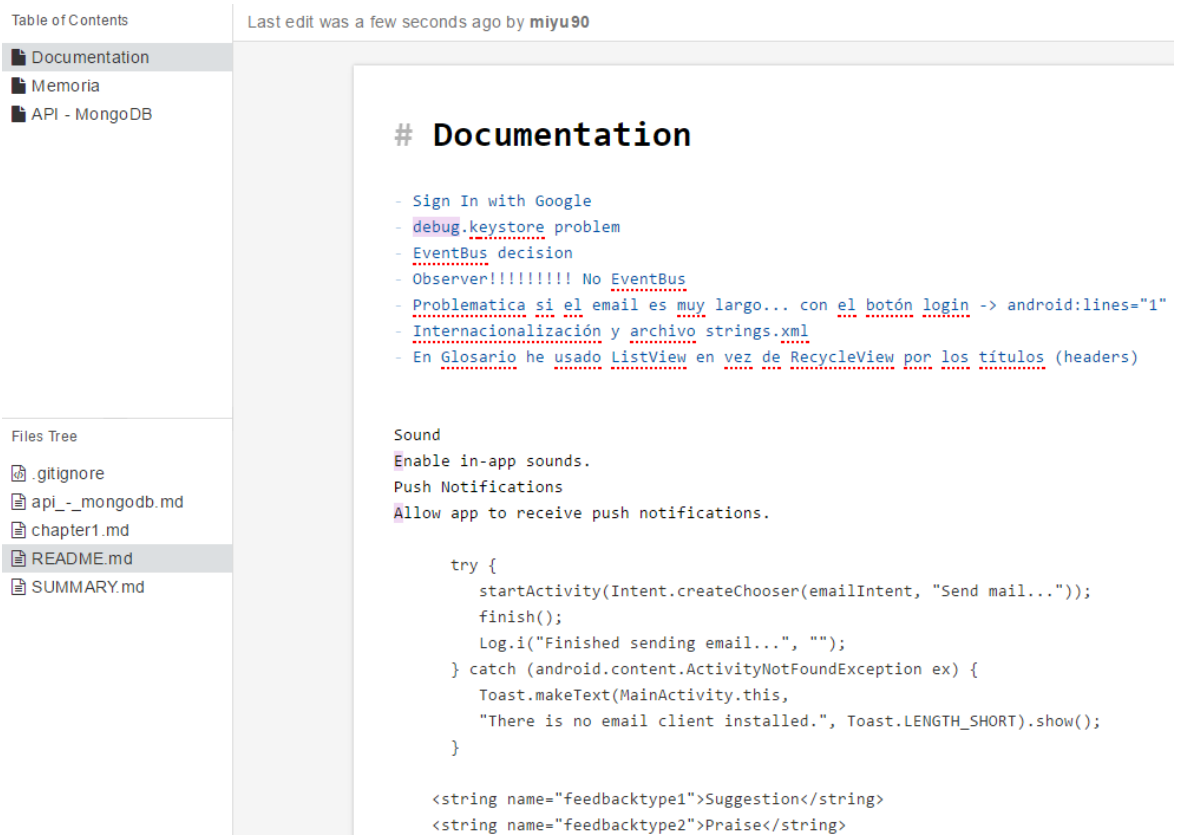

Fig. 5.4. Documentación en GitBook

Ha sido bueno trabajar y ver éstas dos herramientas tan potentes, pero Asana es demasiado completa para una sola persona y GitBook está más enfocado a la documentación técnica y a mostrar la información de forma digitalizada.

Al final, se acabó apuntando los días en los que se hacía formación, implementación o documentación, entendiendo que en algunos casos se puede haber hecho combinaciones de dos tareas, las más comunes formación/implementación e implementación/documentación.

A continuación, se muestra el calendario de planificación para éste proyecto de final de carrera, con su correspondiente leyenda.

CALENDARIO 2016:

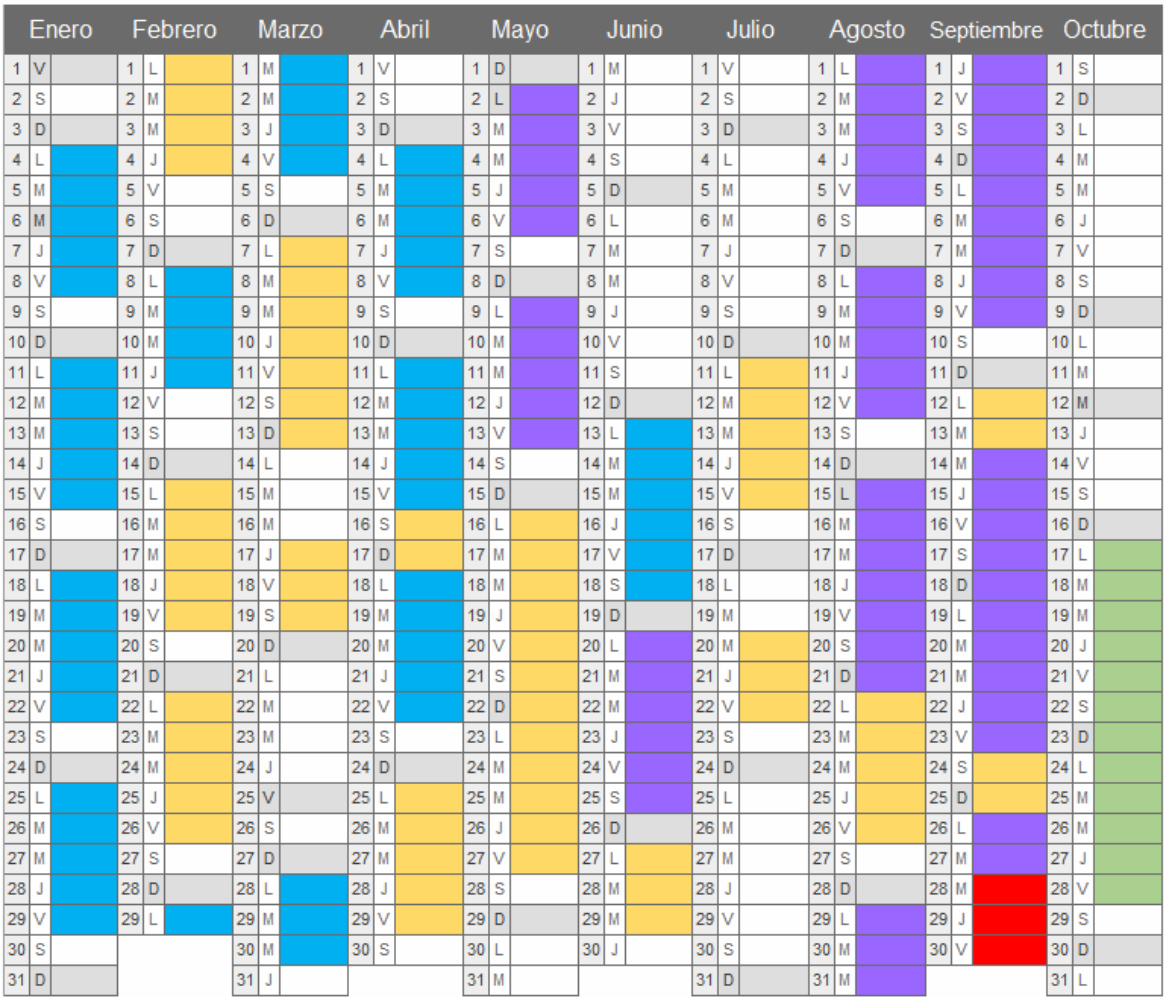

Fig. 5.5. Calendario de planificación AndroidUp 2016

Fig. 5.6. Leyenda del calendario de planificación AndroidUp 2016

Como se muestra en la Fig. 5.5., enero fue el mes donde se concentró más formación, tanto en lenguaje Android como en herramientas, llegando a hacer un curso en Udemy [5] sobre programación Android e implementando los primeros layouts y pantallas. El 15 de enero se entregó el Anteproyeto, que recogía parte de las investigaciones y formación de los meses de noviembre, diciembre y enero.

Las fases de implementación abundan entre febrero y mayo. Fue el 14 de mayo dónde se divisó que era imposible entregar la documentación el 1 de junio, ya que aún faltaba bastante de la implementación, concretamente la parte de integración con la API, así que se pidió el aplazamiento del proyecto.

Antes de hacer un pequeño descanso en junio justo al empezar a trabajar, se concentró un pico de implementación del Login de Google, una de las partes que más dificultades ha traído al proyecto. Después de eso, empezó una fase de formación sobre JSON y obtención de datos en Android, para así implementarlo de la forma más correcta.

Agosto y septiembre se dedicaron casi enteramente a la documentación. Sólo se recurría a la implementación al detectar un bug o proceso mal desarrollado, pero ha sido en septiembre dónde más se ha redactado y documentado todos los procesos del proyecto.

Del 28 al 30 de septiembre se entrega la memoria, un punto crítico en el proyecto. Entre el 17 y el 28 de octubre se hará la exposición y defensa del proyecto delante de un jurado de profesores. Antes de que llegue la fecha de la exposición, se prevé mejorar la implementación de la aplicación y en la medida de lo posible, añadir ampliaciones.

En general, la planificación se ha extendido mucho, teniendo en cuenta que era necesario. La parte buena es que se ha tenido más tiempo para aprender más aspectos de Android y para mejorar la calidad de la implementación final.

6. Definición de la Aplicación

Es importante hacer una fase de análisis y diseño antes de empezar el desarrollo de la aplicación, para poder seguir un esquema de lo que se quiere obtener al final del proceso.

En éste apartado se detallará la descripción del aplicativo, se mostrarán sus mockups, los requisitos no funcionales y los funcionales, los casos de uso, la estructura que tiene la base de datos y la estructura de la API.

6.1. Descripción de la aplicación

El proyecto AndroidUp es una aplicación móvil que proporciona al usuario poder aprender Android de forma sencilla. La app está basada en un sistema de elección de temas y dentro de ellos subtemas, donde encontraremos el contenido a leer por el usuario.

La aplicación está programada en Java y el IDE escogido para realizarla es Android Studio.

El color escogido para el contexto de la aplicación es el **verde**, siguiendo la temática del logotipo de Android, que es un muñeco verde.

6.2. Mockup

Se ha construido un mockup online. Los mockups son utilizados principalmente por los diseñadores de la aplicación, para adquirir comentarios de los clientes finales. Se ha realizado en la plataforma moqups [6], que permite de forma gratuita (con funciones limitadas) el diseño de pantallas.

Las pantallas siguen una estructura pensada para que el usuario sepa dónde está en todo momento y sea fácil de seguir.

Se detallarán las funcionalidades de cada pantalla y el por qué de la estructura.

A continuación, se expone el mockup completo de la aplicación, siguiendo el orden lógico natural de las pantallas:

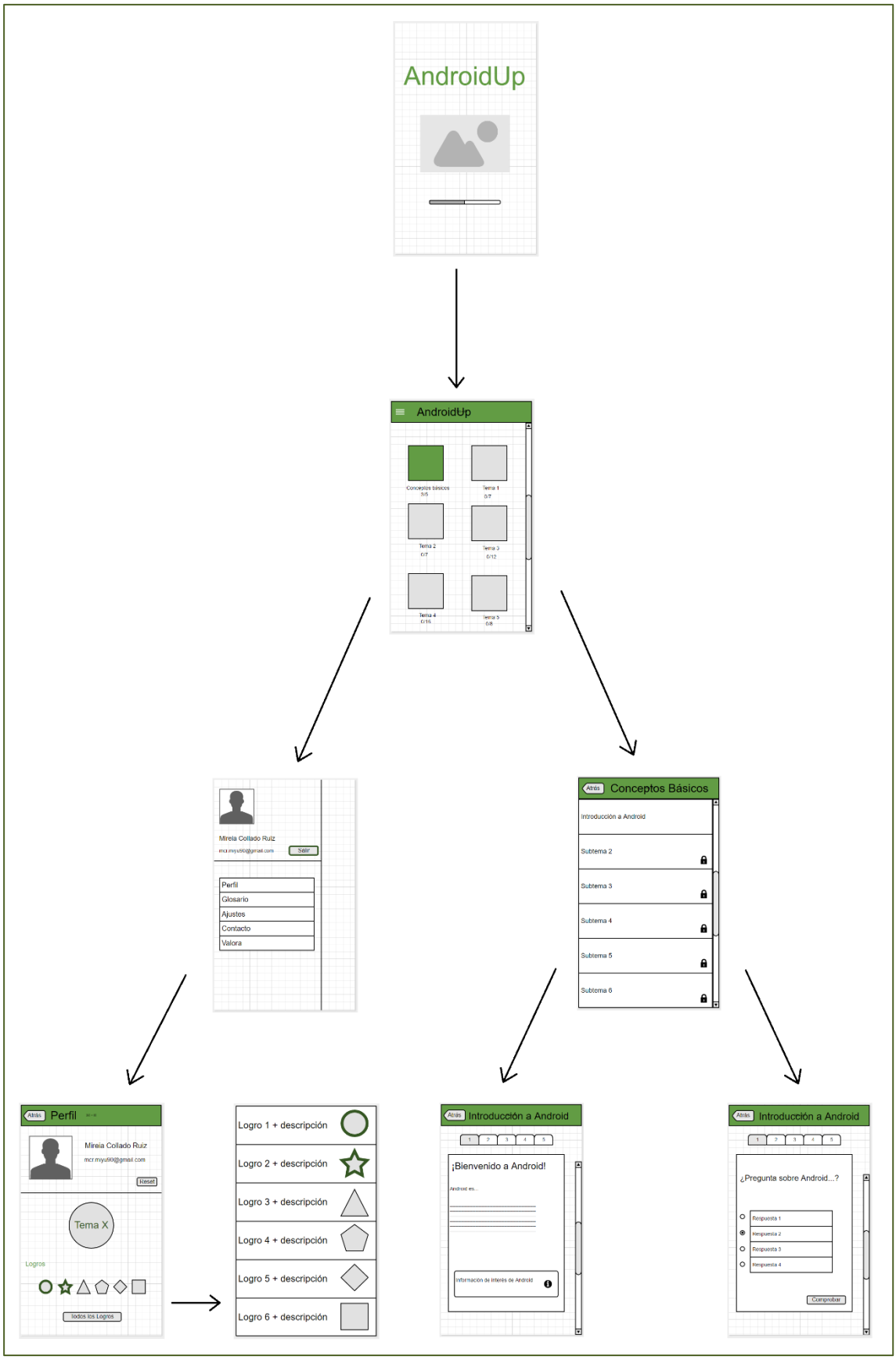

Fig. 6.1. Mockup completo de la aplicación AndroidUp con navegación de pantallas

6.2.1. Pantalla de Inicio

Se considera la posibilidad de añadir una pantalla “Splash Screen” que al abrir la app contenga el logotipo centrado y el nombre AndroidUp encima, y tras un segundo y medio salte al menú principal.

Fig. 6.2. Pantalla de Inicio de AndroidUp

La idea se ha descartado de momento por la rapidez de obtención de las imágenes y títulos de los temas que obtiene la API.

6.2.2. Menú Principal

El menú principal se compone de imágenes y títulos de los temas, mostrados en orden y extraídos de la Base de Datos dinámicamente en tiempo de ejecución.

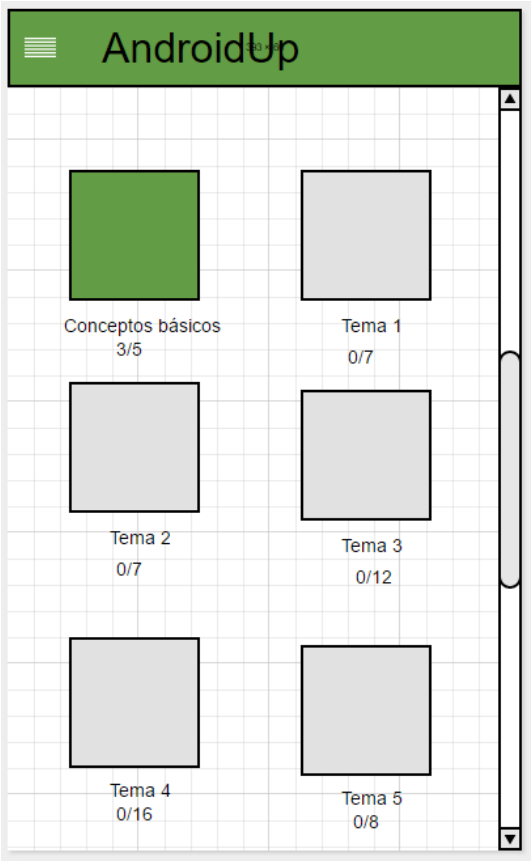

Fig. 6.3. Menú principal de AndroidUp

Las imágenes de los temas son botones de selección táctil que llevan a sus subtemas. Se define el comportamiento de seguimiento de temas, que actualmente no está implementado, indicado con una fracción que expone cuántos subtemas del tema en cuestión hay en total y cuántos de ellos se han completado. Los temas superados completamente aparecerán entonces resaltados en color. Se considera preparar un sistema de bloqueo de temas (no implementado actualmente), para superar el tema 2, antes debe estar completo el tema 1, y así sucesivamente. Para ver todos los temas del menú principal, se dotará a la pantalla de scroll. Tendrá además un botón de menú lateral, explicado más adelante.

6.2.3. Pantalla Tema

Si se escoge un tema, se abre una lista de subtemas con el título de cada uno de ellos, obtenidos en tiempo de ejecución. En la parte superior aparece el nombre del tema.

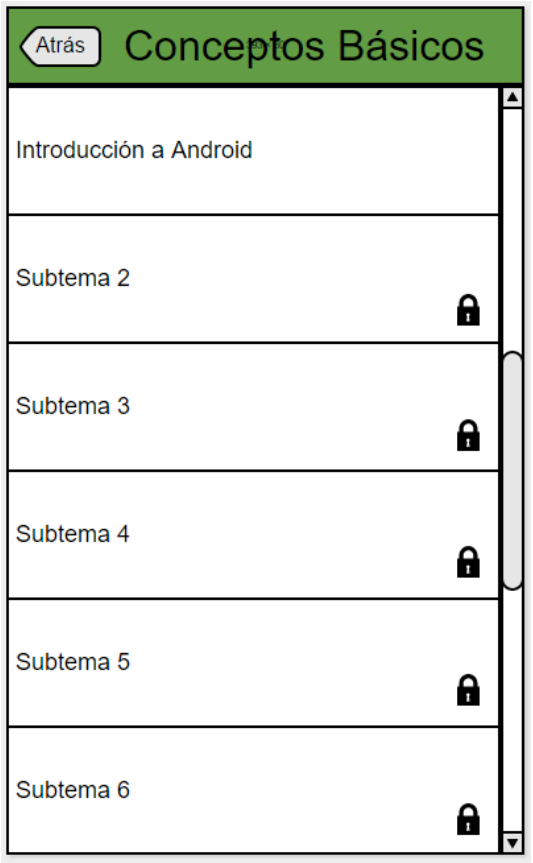

Fig. 6.4. Pantalla Tema de AndroidUp

Se considera preparar un sistema de bloqueo de subtemas (no implementado actualmente) que desbloquee los subtemas a medida que se van completando los anteriores.

6.2.4. Pantalla Subtema

Dentro de un subtema, en la parte superior se encuentra el nombre del subtema y un botón para volver atrás (menú principal). Justo debajo, se encuentra un menú de navegación de pestañas, para recorrer las páginas del subtema. La mayoría de las páginas contendrán teoría o teoría y preguntas conjuntamente, y en algunos casos serán cuestionarios con sólo preguntas. Las pestañas se crearán en tiempo de ejecución y poseen navegación lateral para recorrerlas.

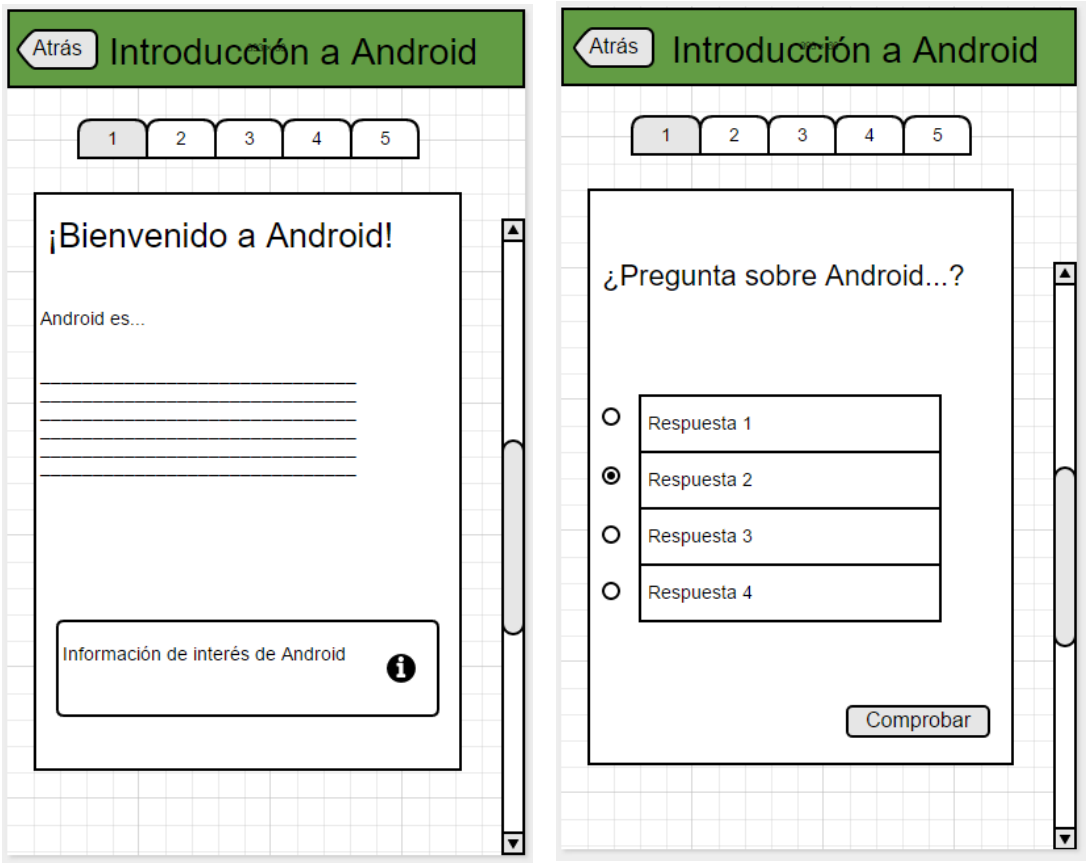

Fig. 6.5. Pantallas Subtema (teoría y pregunta) de AndroidUp

Si es una página de teoría, se encuentra el título de la página y una zona de texto con el contenido de la teoría. Además, en algunos casos podrá contener imágenes, código resaltado con colores y contenido importante a destacar.

Si la página es de pregunta, aparece el texto de la pregunta y entre 3 y 4 opciones de respuesta. Se dispone de un botón para comprobar si la respuesta es correcta o no. El orden las preguntas cambiará de posición cada vez que entremos a la página.

6.2.5. Menú lateral

Se dispone de un menú lateral con dos partes diferenciadas. La primera tiene un botón de Login donde el usuario podrá entrar a la aplicación usando su cuenta de Gmail de Google. Una vez logueado, aparecerá la foto de perfil de Google con el nombre y el email. La otra sección tiene una lista de opciones, entre ellas se considera importante destacar el Perfil del usuario, un Glosario de palabras, los Ajustes globales de la aplicación, un formulario de Contacto y Valorar la aplicación.

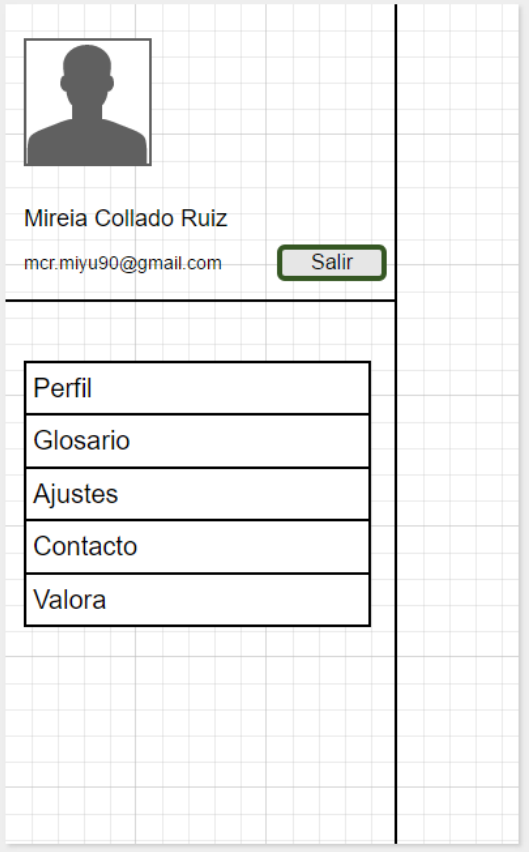

Fig. 6.6. Menú Lateral de AndroidUp

6.2.6. Perfil

En la pantalla de perfil, hay información del usuario: imagen de Google, foto y email. Además, se muestra el seguimiento de temas completados (no implementado actualmente) y los logros que ha conseguido (no implementado actualmente) resaltados en color y un botón para ver todos los logros.

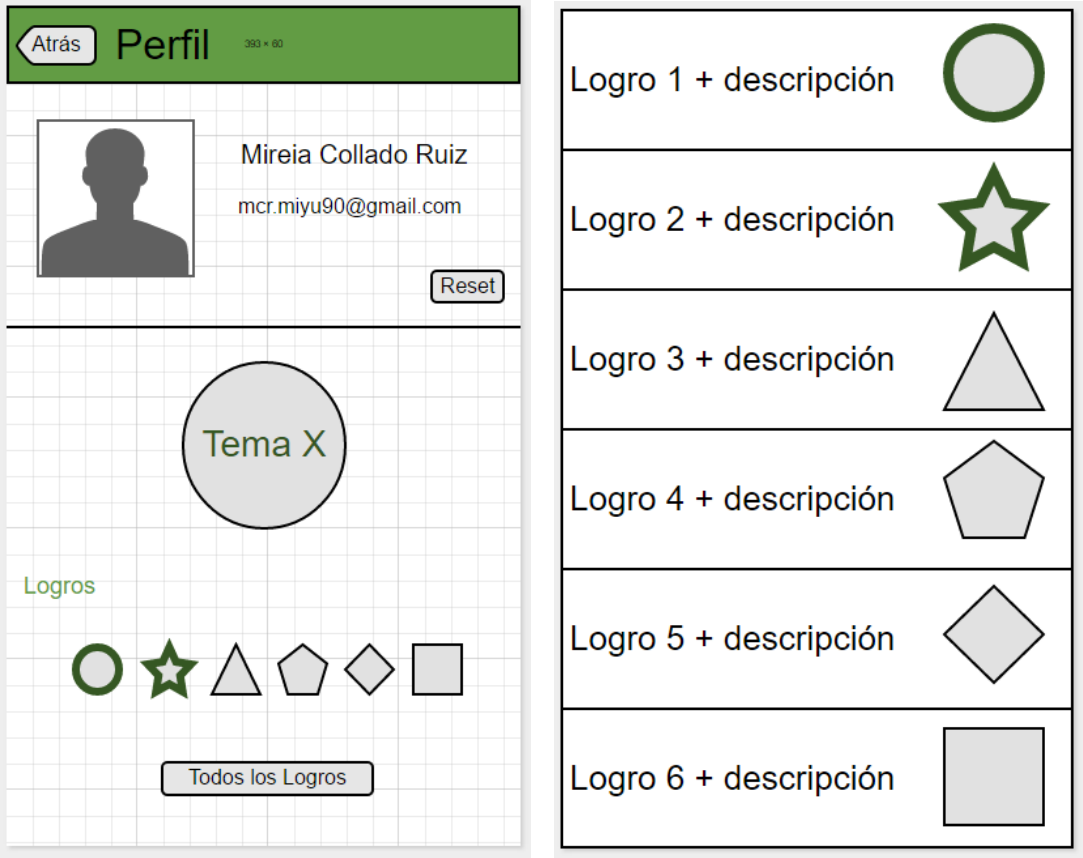

Fig. 6.7. Perfil de usuario y pantalla de logros de AndroidUp

En la pantalla de logros hay una lista con todos los logros disponibles para conseguir con su nombre y descripción.

Se podrá más adelante resetear el progreso del usuario apretando el botón de Reset del perfil, borrando todo el seguimiento de temas y los logros conseguidos.

6.3. Requerimientos no funcionales

Android tiene muchas versiones y se ha de tener en cuenta para programar la aplicación para llegar al mayor número de dispositivos. La versión de Android SDK mínima que se recomienda es la 16, y la versión target será la 23, así se refleja en el módulo de app build.gradle:

```
android {
    compileSdkVersion 23
    buildToolsVersion "23.0.3"

    defaultConfig {
        applicationId "com.miyu.androidup"
        minSdkVersion 16
        targetSdkVersion 23
        versionCode 1
        versionName "1.0"
    }
```

Fig. 6.8. Captura de un fragmento del módulo de app build.gradle

Será posible acceder a la aplicación desde cualquier sitio siempre que se disponga de conexión a Internet (wifi o 3G/4G indistintamente), ya que el temario está alojado en un servidor externo y la información se recogerá de allí. No será posible el acceso al temario de la aplicación cuando el servidor esté caído, al no poder obtener los temas y subtemas, se mostrará un error de servidor.

6.3.1. Usabilidad

Es importante la creación de una interfície amigable para el usuario final, para que le sea fácil moverse entre pantallas y no perderse por los temas y subtemas del temario de la aplicación. Para ello, se han aplicado las consignas del Material Design de Google, sintetizando los principios clásicos de un buen diseño con la innovación de las apps. Todo esto está enfocado al sistema operativo Android, para obtener un diseño más limpio y usable de la aplicación final, con temas y colores armónicos.

6.3.2. Rapidez

El acceso y la interacción con la aplicación ha de ser rápido, por lo que se usarán principalmente Fragments para mostrar el contenido en la aplicación. También se usará MongoDB pensando en la rapidez de obtención de datos mediante la API, intentando que ésta última haga los menos accesos posibles a la base de datos, por ejemplo, si sólo se han de recuperar los títulos e imágenes de los temas, no hacer una consulta que también nos devuelva todos sus subtemas y a su vez todas las páginas del subtema.

6.3.3. Escalabilidad

El contenido de los temas y subtemas será ampliable en tiempo real. La carga de temario estará alojada en el servidor y será consumida por la aplicación de forma dinámica, lo que hace que simplemente se deban añadir nuevos temas y subtemas a la base de datos para que el contenido de la aplicación crezca.

Será posible la implementación de otras funcionalidades a la app, ya que se ha intentado seguir una nomenclatura de variables y métodos sostenible y una arquitectura de paquetes basada en funcionalidades, además de utilizar complementos Android no obsoletos e intentando mantener la compatibilidad de versiones en la medida de lo posible.

Además, algunas funcionalidades estarán preparadas para ser programadas, aunque por falta de tiempo no será posible incluirlas en esta primera versión de la aplicación.

6.3.4. Mantenimiento

El mantenimiento de la aplicación residirá principalmente en la API y la base de datos (añadir, modificar o eliminar contenido) y sobretodo en que no caiga el servidor, ya que todo el contenido del temario se encontrará alojado allí.

6.4. Requisitos funcionales

Como se define en la ingeniería de requisitos, los requisitos funcionales establecen los comportamientos del sistema. A continuación, se muestran los requisitos funcionales de la aplicación AndroidUp. El comportamiento de estos requisitos se expondrá más adelante en los casos de uso.

6.4.1. Requisito 1: Mostrar tema, subtemas y páginas

La aplicación muestra los temas en tiempo de ejecución, y al elegir uno, sus subtemas. Al escoger un subtema, se nos mostrarán las páginas del mismo, todo en tiempo de ejecución, extrayendo el contenido de la Base de Datos mediante la API.

6.4.2. Requisito 2: Hacer Login con Google en la aplicación

La aplicación permite hacer Login con la cuenta Gmail de Google al usuario. Al cerrar la aplicación y volverla a abrir, la sesión quedará abierta hasta que se haga Logout.

6.4.3. Requisito 3: Mostrar listado de palabras del Glosario

La aplicación tiene una pantalla con un Glosario, donde se encuentra una lista de palabras y su definición para consulta del usuario. Las secciones se crean y las palabras se muestran en tiempo de ejecución.

6.4.4. Requisito 4: Consultar y resetear perfil

La aplicación permite consultar el perfil del usuario logueado, su seguimiento (no implementado) y resetear el seguimiento (no implementado).

6.4.5. Requisito 5: Aplicar ajustes a la aplicación

La aplicación permitirá aplicar ajustes (no implementado) como habilitar o deshabilitar los sonidos (no implementado) y habilitar o deshabilitar las notificaciones push (no implementado).

6.4.6. Requisito 6: Contactar con la creadora

La aplicación tiene un apartado donde se encuentra un formulario a rellenar que se envía al equipo AndroidUp, en éste caso a su creadora. El formulario podrá ser sugerencia, elogio, queja o aviso de error detectado por el usuario.

6.4.7. Requisito 7: Compartir la aplicación

La aplicación permite al usuario compartirla mediante otra aplicación a elección del usuario (que tenga instalada en el móvil), por ejemplo, Whatsapp, Facebook, Gmail, etc.

6.4.8. Requisito 8: Valorar la aplicación

La aplicación permite al usuario valorarla en el mercado de aplicaciones Google Play.

6.4.9. Requisito 9: Cambio automático de idioma

La finalidad de este requisito es que el idioma por defecto de la aplicación sea el del dispositivo. En caso de que el idioma del dispositivo no sea uno de los disponibles (actualmente sólo hay disponibles castellano e inglés), se usará por defecto el español.

6.5. Casos de uso

Se muestran a continuación los 9 casos de uso de la aplicación AndroidUp:

CU-001: Login Google

CU-002: Logout Google

CU-003: Ver Tema

CU-004: Listar Palabras Glosario

CU-005: Consultar Perfil

CU-006: Aplicar Ajustes Aplicación

CU-007: Contactar Creadora

CU-008: Compartir Aplicación

CU-009: Valorar Aplicación

6.5.1. CU-001: Login Google

CU-001	**Login Google**	
Versión	2.0	
Dependencias	-	
Actor	Usuario de la aplicación	
Pre-condición	El usuario debe tener instalada la aplicación en su dispositivo. El usuario debe tener conexión a internet.	
Descripción	El sistema deberá comportarse como se describe en el siguiente caso de uso cuando se quiera hacer un Login a la aplicación con Google.	
Flujo principal	**Paso**	**Acción**
	1	El usuario quiere loguearse en la aplicación.
	2	El usuario aprieta el botón del menú lateral y seguidamente el botón de Login.
	3	El Sistema muestra la pantalla de logueo.
	4	El usuario aprieta el botón de iniciar sesión con Google.
	5	El Sistema muestra los correos Gmail vinculados al dispositivo.
	6	El usuario elige un correo Gmail.
	7	El Sistema logue al usuario al sistema y vuelve a la pantalla principal de temas, cerrando el menú lateral. Las credenciales del usuario (imagen, nombre y email de Google) se muestran en el menú lateral.
Post-condición	El usuario se loguea a la aplicación con Google.	
Extensiones o flujos alternativos	**Paso**	**Acción**
	1	No funcional, si el usuario no quiere loguearse en la aplicación, el flujo principal no comienza.
	4	Si no hay conexión a internet, el Sistema mostrará un mensaje de error de red.
	5	Si sólo hay un correo vinculado al dispositivo, se conecta directamente a ése correo.
	6	Si el usuario no elige un correo, tendrá que volver al paso 4.
	7	Si cualquiera de las credenciales es null, el Sistema pondrá en su lugar una credencial vacía.
Comentarios	El Login se hará con Google.	

Tabla 6.1. Tabla del caso de uso “Login Google”

6.5.2. CU-002: Logout Google

CU-002	Logout Google	
Versión	2.0	
Dependencias	**CU-001**: Login Google (necesaria)	
Actor	Usuario de la aplicación	
Pre-condición	El usuario debe tener instalada la aplicación en su dispositivo. El usuario debe estar logueado en la aplicación. El usuario debe tener conexión a internet.	
Descripción	El sistema deberá comportarse como se describe en el siguiente caso de uso cuando se quiera hacer un Logout de la aplicación con Google.	
Flujo principal	**Paso**	**Acción**
	1	El usuario quiere desloguearse de la aplicación.
	2	El usuario aprieta el botón de Logout del menú lateral.
	3	El Sistema desloguea al usuario del sistema y vuelve a la pantalla principal de temas, cerrando el menú lateral. Las credenciales del usuario (imagen, nombre y email de Google) ya no se muestran en el menú lateral.
Post-condición	El usuario se desloguea de la aplicación con Google.	
Extensiones o flujos alternativos	**Paso**	**Acción**
	1	No funcional, si el usuario no quiere desloguearse de la aplicación, el flujo principal no comienza.
Comentarios	-	

Tabla 6.2. Tabla del caso de uso “Logout Google”

6.5.3. CU-003: Ver Tema

CU-003	**Ver Tema**	
Versión	3.0	
Dependencias	-	
Actor	Usuario de la aplicación	
Pre-condición	El usuario debe tener instalada la aplicación en su dispositivo. El usuario debe tener conexión a internet.	
Descripción	El sistema deberá comportarse como se describe en el siguiente caso de uso cuando se quiera ver un tema completo, con todos sus subtemas y las páginas de los subtemas.	
Flujo principal	**Paso**	**Acción**
	1	El usuario quiere ver un tema completo.
	2	El usuario aprieta sobre un tema en la pantalla principal de temas.
	3	El Sistema muestra todos los subtemas del tema en orden.
	4	El usuario elige un subtema.
	5	El Sistema muestra las páginas del subtema en orden.
	6	El usuario navega por las páginas para ver todo el subtema
	7	El usuario va hacia atrás y selecciona el siguiente subtema. Volver al punto 5 hasta terminar de ver todas las páginas de todos los subtemas del tema.
Post-condición	El usuario recorre el tema escogido, viendo sus subtemas y las páginas de los subtemas.	
Extensiones o flujos alternativos	**Paso**	**Acción**
	1	No funcional, si el usuario no quiere ver el tema, el flujo principal no comienza.
Comentarios	-	

Tabla 6.3. Tabla del caso de uso “Ver Tema”

6.5.4. CU-004: Listar Palabras Glosario

CU-004	Listar Palabras Glosario	
Versión	1.0	
Dependencias	-	
Actor	Usuario de la aplicación	
Pre-condición	El usuario debe tener instalada la aplicación en su dispositivo. El usuario debe tener conexión a internet.	
Descripción	El sistema deberá comportarse como se describe en el siguiente caso de uso cuando se quiera consultar el listado de palabras del glosario.	
Flujo principal	**Paso**	**Acción**
	1	El usuario quiere consultar el listado de palabras del glosario.
	2	El usuario aprieta el botón “Glosario” del menú lateral.
	3	El Sistema cierra el menú lateral y muestra la lista de palabras del glosario por secciones y ordenadas alfabéticamente.
Post-condición	El usuario ve la lista de palabras del glosario.	
Extensiones o flujos alternativos	**Paso**	**Acción**
	1	No funcional, si el usuario no quiere consultar la lista de palabras del glosario, el flujo principal no comienza.
	3	. Si no hay conexión a internet, el listado de palabras no se mostrará.
Comentarios	-	

Tabla 6.4. Tabla del caso de uso “Listar Palabras Glosario”

6.5.5. CU-005: Consultar Perfil

CU-005	**Consultar Perfil**	
Versión	1.0	
Dependencias	**CU-001**: Login Google (necesaria)	
Actor	Usuario de la aplicación	
Pre-condición	El usuario debe tener instalada la aplicación en su dispositivo. El usuario debe estar logueado en la aplicación. El usuario debe tener conexión a internet.	
Descripción	El sistema deberá comportarse como se describe en el siguiente caso de uso cuando se quiera consultar el perfil de usuario.	
Flujo principal	**Paso**	**Acción**
	1.1	El usuario quiere consultar su perfil.
	1.2	El usuario aprieta el botón “Perfil” del menú lateral.
	1.3	El Sistema cierra el menú lateral y muestra el perfil del usuario, con la imagen de Google, el nombre y el email, además de información sobre el seguimiento de temas y los logros del usuario.
Flujo secundario	2.1	El usuario quiere consultar su perfil.
	2.2	El usuario aprieta la parte superior del menú lateral, dónde se encuentran la imagen de Google, el nombre y el email.
	2.3	El Sistema cierra el menú lateral y muestra el perfil del usuario, con la imagen de Google, el nombre y el email, además de información sobre el seguimiento de temas y los logros del usuario.
Post-condición	El usuario consulta su perfil.	
Extensiones o flujos alternativos	**Paso**	**Acción**
	1.1 / 2.1	No funcional, si el usuario no quiere consultar su perfil, el flujo principal no comienza.
	1.2	. Si el usuario no está logueado en la aplicación, el sistema mostrará un mensaje comunicando que antes de consultar el perfil hay que loguear.
	2.2	. Si el usuario no está logueado al sistema, el flujo secundario no funcionará.
Comentarios	-	

Tabla 6.5. Tabla del caso de uso “Consultar Perfil”

6.5.6. CU-006: Aplicar Ajustes Aplicación

CU-006	Aplicar Ajustes Aplicación	
Versión	1.0	
Dependencias	-	
Actor	Usuario de la aplicación	
Pre-condición	El usuario debe tener instalada la aplicación en su dispositivo.	
Descripción	El sistema deberá comportarse como se describe en el siguiente caso de uso cuando se quiera aplicar ajustes a la aplicación.	
Flujo principal	**Paso**	**Acción**
	1	El usuario quiere aplicar ajustes a la aplicación.
	2	El usuario aprieta el botón “Ajustes” del menú lateral.
	3	El Sistema cierra el menú lateral y muestra la pantalla de ajustes globales de la aplicación como, por ejemplo, habilitar o deshabilitar sonidos, notificaciones push, etc.
	4	El usuario aplica los ajustes que desea.
Post-condición	El usuario aplica ajustes a la aplicación.	
Extensiones o flujos alternativos	**Paso**	**Acción**
	1	No funcional, si el usuario no quiere aplicar ajustes a la aplicación, el flujo principal no comienza.
Comentarios	En una primera iteración, los cambios se guardarán en el dispositivo (de forma local). Valorar si es necesario que los ajustes se guarden en la información del usuario (en la nube).	

Tabla 6.6. Tabla del caso de uso “Aplicar Ajustes Aplicación”

6.5.7. CU-007: Contactar Creadora

CU-007	**Contactar Creadora**	
Versión	1.0	
Dependencias	**CU-001**: Login Google (no necesaria)	
Actor	Usuario de la aplicación	
Pre-condición	El usuario debe tener instalada la aplicación en su dispositivo. El usuario debe tener conexión a internet.	
Descripción	El sistema deberá comportarse como se describe en el siguiente caso de uso cuando se quiera contactar con la creadora de la aplicación.	
Flujo principal	**Paso**	**Acción**
	1	El usuario quiere contactar con la creadora de la aplicación.
	2	El usuario aprieta el botón “Contacto” del menú lateral.
	3	El Sistema cierra el menú lateral y muestra un formulario con el nombre y email del usuario si éste está logueado (en blanco si no), además de varios campos a cumplimentar como el motivo del contacto, los detalles y si se requiere respuesta.
	4	El usuario introduce todos los datos del formulario que se le piden y aprieta el botón “Enviar”.
	5	El Sistema recoge los datos y los envía en forma de correo al email de la creadora de la aplicación.
Post-condición	Se envía un correo al email de la creadora de la aplicación.	
Extensiones o flujos alternativos	**Paso**	**Acción**
	1	No funcional, si el usuario no quiere loguearse en la aplicación, el flujo principal no comienza.
	3	Si el usuario así lo desea, podrá cambiar la información de nombre y email, aunque aparezca el nombre y email del Login.
	4	Si envía algún campo obligatorio vacío, el sistema le avisará con un error.
Comentarios	-	

Tabla 6.7. Tabla del caso de uso “Contactar Creadora”

6.5.8. CU-008: Compartir Aplicación

CU-008	Compartir Aplicación	
Versión	1.0	
Dependencias	-	
Actor	Usuario de la aplicación	
Pre-condición	El usuario debe tener instalada la aplicación en su dispositivo. El usuario debe tener conexión a internet.	
Descripción	El sistema deberá comportarse como se describe en el siguiente caso de uso cuando se quiera compartir la aplicación.	
Flujo principal	**Paso**	**Acción**
	1	El usuario quiere compartir de la aplicación.
	2	El usuario aprieta el botón “Comparte” del menú lateral.
	3	El Sistema cierra el menú lateral y muestra un recuadro con las aplicaciones disponibles del dispositivo con las que el usuario puede compartir la aplicación.
	4	El usuario elige una aplicación disponible y envía el enlace por el medio escogido.
Post-condición	El usuario comparte la aplicación.	
Extensiones o flujos alternativos	**Paso**	**Acción**
	1	No funcional, si el usuario no quiere compartir la aplicación, el flujo principal no comienza.
	4	Si el usuario no elige una aplicación, tendrá que volver al paso 1.
Comentarios	Sólo aparecen disponibles las aplicaciones que el usuario tiene instaladas en su dispositivo.	

Tabla 6.8. Tabla del caso de uso “Compartir Aplicación”

6.5.9. CU-009: Valorar Aplicación

CU-009	Valorar Aplicación	
Versión	1.0	
Dependencias	-	
Actor	Usuario de la aplicación	
Pre-condición	El usuario debe tener instalada la aplicación en su dispositivo. El usuario debe tener conexión a internet.	
Descripción	El sistema deberá comportarse como se describe en el siguiente caso de uso cuando se quiera valorar la aplicación.	
Flujo principal	**Paso**	**Acción**
	1	El usuario quiere valorar de la aplicación.
	2	El usuario aprieta el botón “Valora” del menú lateral.
	3	El Sistema cierra el menú lateral y abre el Google Play.
	4	El usuario valora la aplicación en Google Play.
Post-condición	El usuario valora la aplicación en Google Play.	
Extensiones o flujos alternativos	**Paso**	**Acción**
	1	No funcional, si el usuario no quiere valorar la aplicación, el flujo principal no comienza.
	4	Si el usuario aprieta el botón atrás, se volverá a la aplicación.
Comentarios	La aplicación se valorará en el market Google Play.	

Tabla 6.9. Tabla del caso de uso “Valorar Aplicación”

5.5.10. Actor

En todos los casos, el actor principal del proyecto AndroidUp es el usuario que utilizará la aplicación. Se espera que se trate de un usuario con un mínimo de conocimientos en programación orientada a objetos y familiarizado con Java.

6.6. Diagrama de casos de uso

El siguiente diagrama de casos de uso muestra una visión general de las acciones que puede realizar el usuario, descritas en el punto anterior:

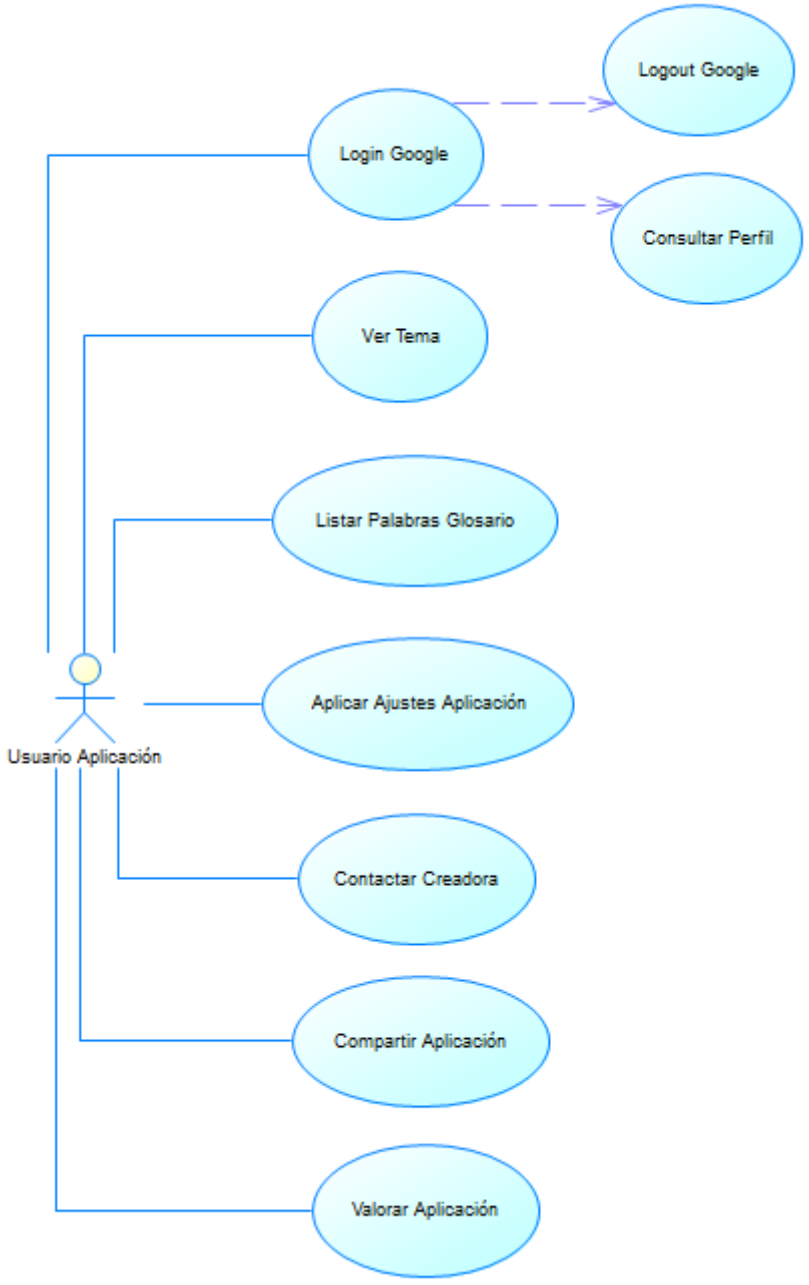

Fig. 6.9. Diagrama de casos de uso

6.7. Diagrama de clases del dominio

Contaremos con 4 clases del dominio básicas y 2 de ampliación.

Básicas:

- Tema
- Subtema
- Página
- Palabra

Ampliación:

- Usuario
- Logro

Se muestra a continuación un diagrama que las relaciona entre ellas:

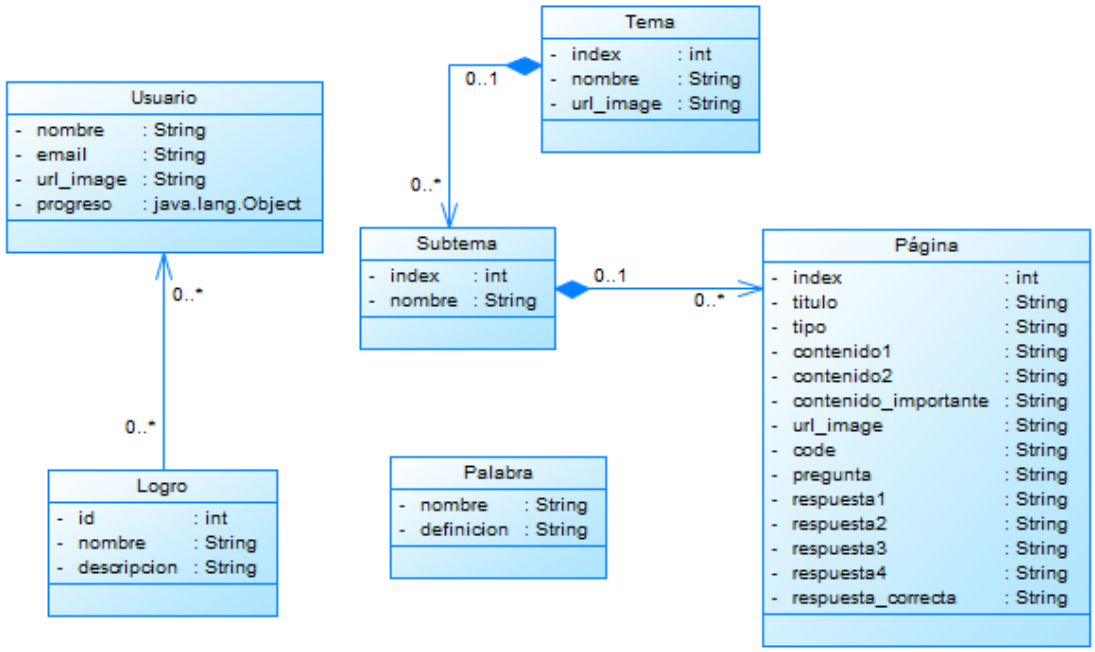

Fig. 6.10. Diagrama de clases del dominio

6.8. Estructura de la Base de Datos

La Base de Datos escogida es MongoDB. Por las siguientes razones:

- Por la estructura documental: Como la aplicación mayormente contiene datos estructurados en temas que contienen subtemas que a su vez contienen páginas, se pensó que era más adecuado la estructura de documentos que ofrece MongoDB, de ésta forma, la integración de datos es más fácil y rápida.
- Se quería probar una Base de Datos no relacional, y MongoDB proporciona mucha documentación y tiene mucha comunidad.
- El repositorio de Spring con MongoDB es muy completo.

Su estructura de documentos es la siguiente:

```
Tema {
        index
        nombre
        url_image

        subtemas [
                { index
                nombre }
        ]
        páginas [
                { index
                titulo
                tipo
                contenido1
                contenido2
                contenido_importante
                url_imagen
                code
                pregunta
                respuesta1
                respuesta2
                respuesta3
```

```
            respuesta4
            respuesta_correcta }
      ]
}
```

```
Palabra {
      nombre
      definición
}
```

```
Logro {
      id
      nombre
      descripcion
}
```

```
Usuario {
      nombre
      email
      url_image
      progreso [
            temas [
                  {id}
            ]
            logros [
                  {id}
            ]
      ]
}
```

6.9. Definición de la API REST

La API se ha construido con Spring utilizando SpringBoot. SpringBoot proporciona una base de librerías y configuraciones que facilitan la creación de una primera versión de la API. Algunas características son:

- La posibilidad de crear aplicaciones stand-alone (que no depende de otros programas para ser ejecutado)
- Incluye un Tomcat integrado (no hay necesidad de distribuir ficheros war)
- Incluye un fichero POM para simplificar la configuración de Maven

7. Desarrollo de la Aplicación

En el capítulo anterior se han mostrado las definiciones y el análisis previo al desarrollo. En éste capítulo se mostrará el desarrollo final de la aplicación, con el diseño de la interfaz, sus características y funcionalidades y cómo y con qué han sido implementadas, los problemas encontrados y las soluciones dadas o propuestas para resolverlos, la base de datos funcionando y el código y funcionamiento de la API y cómo se ha hecho el control de versiones.

7.1. Diseño de la Interfaz

Se muestran en éste apartado las pantallas finales de la aplicación, vistas desde un dispositivo One Plus One X con Android 5.1.1 con API 22:

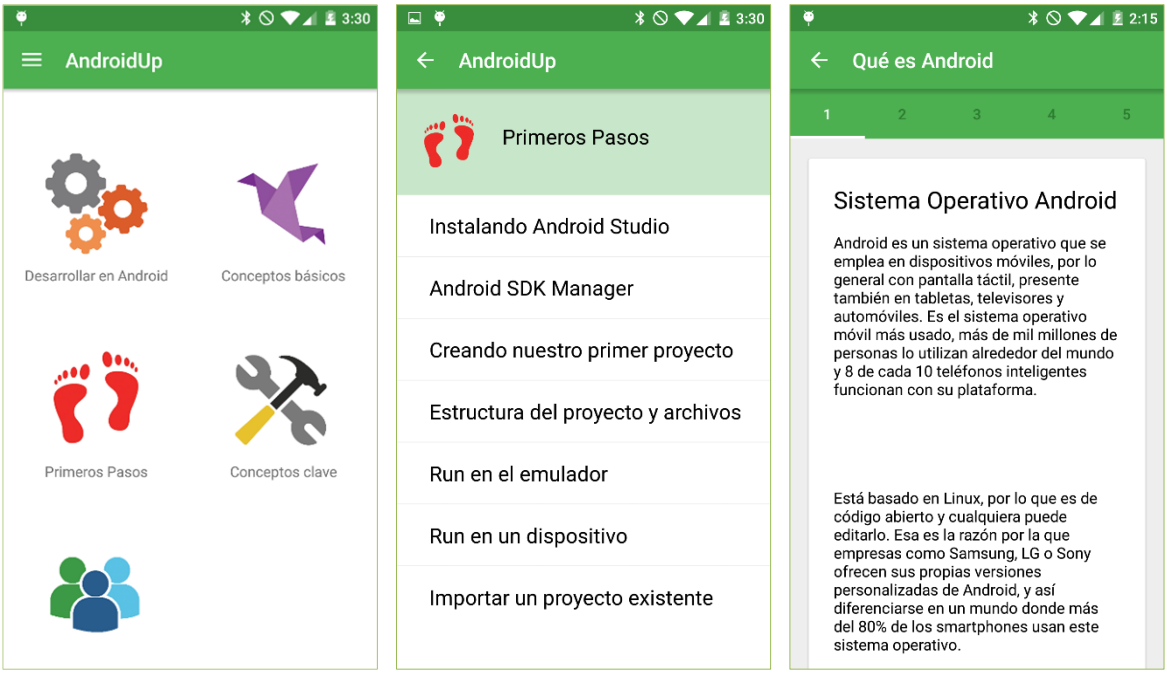

Fig. 7.1. Pantallas Tema, Subtema y Página de AndroidUp

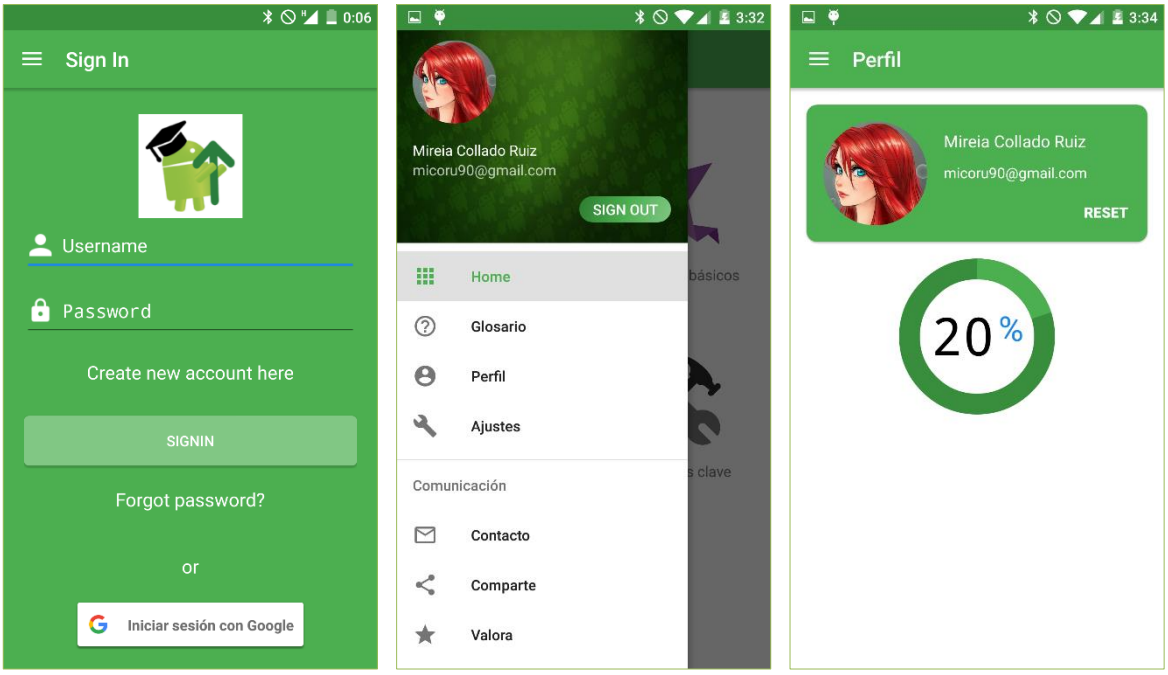

Fig. 7.2. Pantallas Login, Menú Lateral y Perfil de AndroidUp

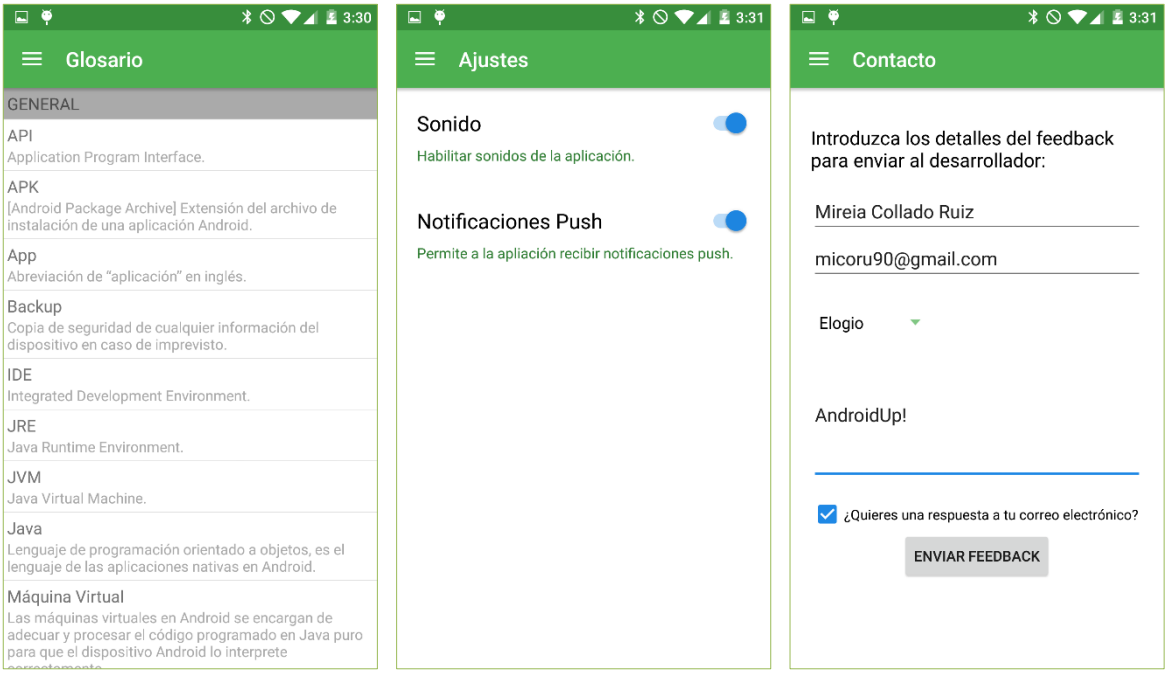

Fig. 7.3. Pantallas Glosario, Ajustes y Contacto de AndroidUp

7.2. Diagrama de paquetes

El siguiente diagrama de paquetes muestra 3 niveles. El primero es el paquete del que heredan todos los demás: [com.miyu.android], y a partir de él se encuentran los paquetes dominio, subtemas, paginas, menu, utils y volley. Dentro de menu se alojan los paquetes ajustes, glosario, compartir, login, contacto, perfil y temas. Se ha intentado seguir una nomenclatura de paquetes que hagan referencia a las acciones que realizan sus clases.

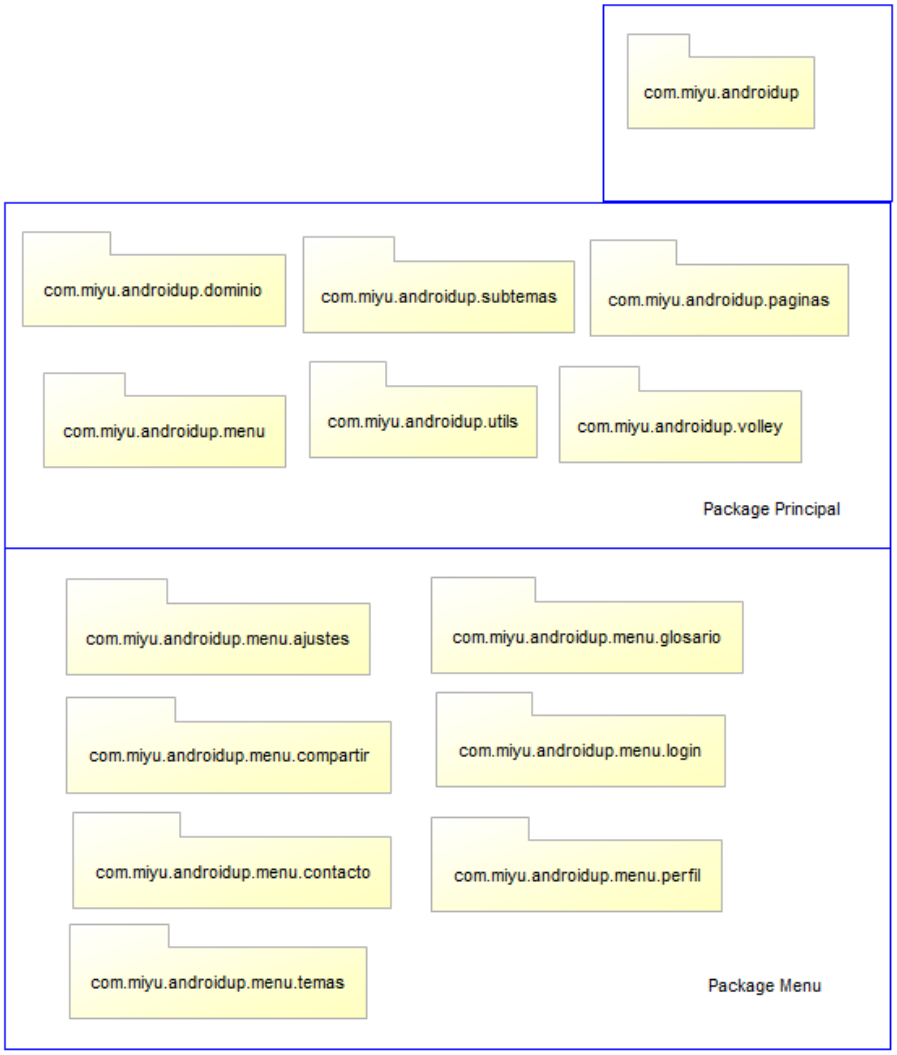

Fig. 7.4. Diagrama de paquetes AndroidUp

7.3. Gráfico de despliegue de clases Java

En éste apartado se muestra un despliegue de las clases de Java y los recursos *res* del proyecto:

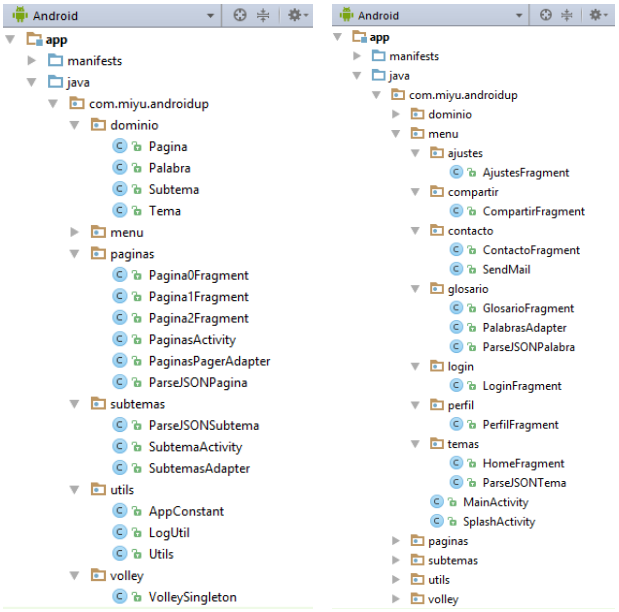

Fig. 7.5. Despliegue de clases en carpeta java

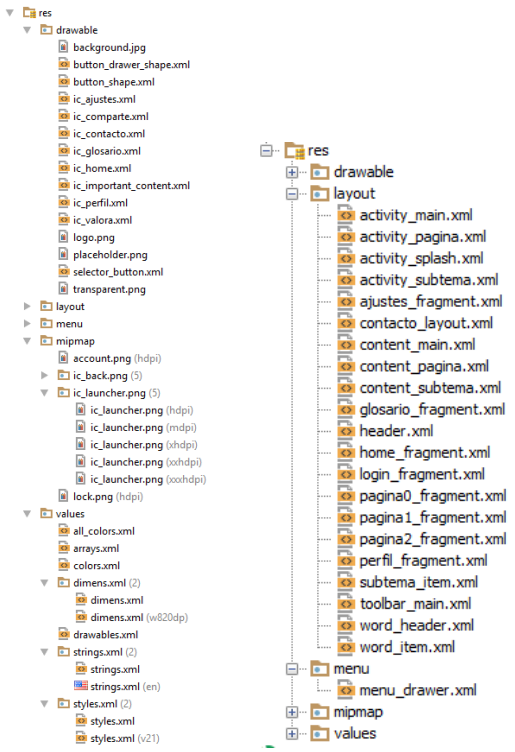

Fig. 7.6. Despliegue de recursos en carpeta *res*

7.4. Características y Funcionalidades

A continuación, se explica la implementación de las características y funcionalidades más destacadas de la aplicación. Para ver en contexto éste capítulo, se aconseja abrir el código en Android Studio.

7.4.1. NavigationView de MainActivity

La aplicación dispone de un NavigationView que funciona de menú lateral al apretar el botón con 3 líneas horizontales. Un NavigationView sólo se puede implementar con Fragments. Es la MainActivity la que controla todas las creaciones de Fragments dependiendo de la opción que se elija del menú:

```
public void displayView(int viewId) {
    Fragment fragment = null;
    String title = getString(R.string.app_name);

    switch (viewId) {
        case R.id.nav_home:
            fragment = new HomeFragment();
            this.homefragment = fragment;
            title  = getString(R.string.home_title);
            setCheckedAndChekacle(R.id.nav_home, true);
            break;

        case R.id.nav_glosario:
            fragment = new GlosarioFragment();
            setCheckedAndChekacle(R.id.nav_glosario, true);
            title = getString(R.string.glosario_title);
            break;

        case R.id.nav_perfil:
            if(signButton.getText().equals(getString(R.string.signout))) {
                fragment = new PerfilFragment();
                title = getString(R.string.perfil_title);
                setCheckedAndChekacle(R.id.nav_perfil, true);
            } else {
                Toast.makeText(this, R.string.noPerfil, Toast.LENGTH_LONG).show();
                setCheckedAndChekacle(R.id.nav_perfil, false);
            }
            break;

        case R.id.nav_ajustes:
            fragment = new AjustesFragment();
            setCheckedAndChekacle(R.id.nav_ajustes, true);
            title = getString(R.string.ajustes_title);
            break;

        case R.id.nav_contacto:
            fragment = new ContactoFragment();
            setCheckedAndChekacle(R.id.nav_contacto, true);
            title = getString(R.string.contacto_title);
            break;
               …

}
```

También es la encargada de abrir o cerrar el NavigationView, de actualizar la interfaz dependiendo de si el usuario está o no logueado y del sistema de Logout, explicado más adelante en el apartado 7.4.6.

7.4.2. Generación de temas en tiempo de ejecución

Para generar los temas en tiempo de ejecución, no pueden estar declarados dentro del layout, se han de instanciar por código. Esto significa que se le han de dar @id diferentes para que al apretarlos, el onClick() lleve al subtema correspondiente, es decir, sean objetos diferentes. En HomeFragment es dónde se van a crear, parsear y ordenar para ser mostrados. El método que los añade al layout es sortAndAddSections(), al que se le pasa una lista de objetos del dominio Tema ya parseados del JSON (esto lo hace la clase ParseJSONTema), es decir, los objetos Tema ya tienen dentro toda la información (índice, imagen y título). Así, se les aplica un sort ordenándolos por índice, y luego se recorren para crear cada imagen y título de cada tema, dentro de una ImageView y de una TextView respectivamente. Se destaca el hecho de que las ImageView tienen diferentes @id gracias al modifier *final*. Para colocar los temas en orden, se mira si el índice es impar o par, ya que se colocará en el LinearLayout de la izquierda si es impar o en el LinearLayout de la derecha si el índice es par:

```
private void sortAndAddSections(ArrayList<Tema> itemList) {
    //First we sort the array
    Collections.sort(itemList);

    int numTemas = itemList.size();

    //Loops through the list
    for(int t = 0; t < numTemas; t++) {

        final int index = itemList.get(t).getIndex();
        final ImageView imageTema = new ImageButton(getActivity());

        final Tema tema = new Tema(index, itemList.get(t).getTitulo(),
itemList.get(t).getUrl_image());

        //get imagen
        imageTema.setId(index);
        imageTema.setBackgroundColor(Color.TRANSPARENT);

Picasso.with(getActivity()).load(tema.getUrl_image()).error(R.drawable.placeholder).fit().i
nto(imageTema);

        //getContext
        txtTema = new TextView(getActivity());
        //get nombre tema
        txtTema.setText(tema.getTitulo());

        if (((index) % 2) != 0) { //IMPAR

            imageTema.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    //aquí en vez de preguntar el index, todos iran a parar a
SubtemaActivity, pero hay que pasarle el index para que sepa lo que tiene que mostrar
                    System.out.println("IMPAR CLICKED!!!!!");
                    Toast.makeText(getActivity(),"IMPAR: "+index,Toast.LENGTH_LONG).show();
                    Intent intent = new Intent(getActivity(), SubtemaActivity.class);
                    intent.putExtra("index_tema", index);
                    intent.putExtra("header_image", tema.getUrl_image());
                    intent.putExtra("header_title", tema.getTitulo());
                    startActivity(intent);
                }
            });
```

```
            impar.addView(imageTema);
            impar.addView(txtTema);

        } else { //PAR

            imageTema.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    Intent intent = new Intent(getActivity(), SubtemaActivity.class);
                    System.out.println("PAR CLICKED!!!!!");
                    Toast.makeText(getActivity(),"PAR: "+index,Toast.LENGTH_LONG).show();
                    intent.putExtra("index_tema", index);
                    intent.putExtra("header_image", tema.getUrl_image());
                    intent.putExtra("header_title", tema.getTitulo());
                    startActivity(intent);
                }
            });

            par.addView(imageTema);
            par.addView(txtTema);
        }

        int dimensionInDp = (int) TypedValue.applyDimension(TypedValue.COMPLEX_UNIT_DIP,
dimensionInPixel, getResources().getDisplayMetrics());
        int marginTop = (int) TypedValue.applyDimension(TypedValue.COMPLEX_UNIT_DIP, 50,
getResources().getDisplayMetrics());
        LinearLayout.LayoutParams layoutParams = new
LinearLayout.LayoutParams(dimensionInDp, dimensionInDp);
        layoutParams.setMargins(0, marginTop, 0, 0); //layoutParams.setMargins(left, top,
right, bottom);
        layoutParams.gravity = Gravity.CENTER;
        imageTema.setLayoutParams(layoutParams);

        LinearLayout.LayoutParams layoutParams2 = new
LinearLayout.LayoutParams(LinearLayout.LayoutParams.MATCH_PARENT,
LinearLayout.LayoutParams.WRAP_CONTENT);
        txtTema.setGravity(Gravity.CENTER);
        txtTema.setLayoutParams(layoutParams2);

        if (index == numTemas) {
            int marginBottom = (int) TypedValue.applyDimension(TypedValue.COMPLEX_UNIT_DIP,
25, getResources().getDisplayMetrics());
            layoutParams2.setMargins(0, 20, 0, marginBottom);
            txtTema.setLayoutParams(layoutParams2);
        }
    }
}
```

Se usa un RequestQueue de la clase VolleySingleton para poder parsear los objetos de la Base de Datos.

7.4.3. Generación de subtemas en tiempo de ejecución

La generación de subtemas en tiempo de ejecución es muy parecida a la generación de temas: tenemos sortAndAddSections() en SubtemaActivity que opera igual que en temas, con una lista de objetos del dominio Subtema que ya están parseados gracias a ParseJSONSubtema y que tienen @id diferentes para poder mostrar las páginas correspondientes en cada caso. Ahora bien, tienen una pequeña diferencia, y es que, al estar mostrados dentro de una

ListView, necesitan de un SubtemasAdapter para operar con el ArrayAdapter y personalizarlo:

```
public class SubtemasAdapter extends ArrayAdapter<Subtema> {

    LayoutInflater inflater;
    private Context context;
    ArrayList<Palabra> items;

    public SubtemasAdapter(Context context, ArrayList<Subtema> items) {
        super(context, 0, items);
        this.context = context;
        inflater = (LayoutInflater)
context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        Subtema subtema = getItem(position);

        convertView = inflater.inflate(R.layout.subtema_item, null);

        TextView tv = (TextView) convertView.findViewById(R.id.subtema_title);

        tv.setText(subtema.getTitulo());

        return convertView;
    }

}
```

A partir de aquí, se evocan todos los Subtemas en la ListView por orden de índice y se añade un ItemClickListener a la lista:

```
mList.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        Toast.makeText(getBaseContext(), "position: " + position + ", id: " +
subtemas.get(position).getIndex(), Toast.LENGTH_LONG).show();
        Intent intent = new Intent(getBaseContext(), PaginasActivity.class);
        intent.putExtra("index_tema", id_tema);
        intent.putExtra("subtema_title", subtemas.get(position).getTitulo());
        intent.putExtra("subtema_index", subtemas.get(position).getIndex());
        startActivity(intent);
    }
});
```

Con esto, llama a PaginasActivity pasándole por el Intent los extras del índice de tema y el índice de subtema para encontrar en la Base de Datos las páginas correspondientes (el índice del tema se lo ha pasado el tema a su subtema a su vez).

7.4.4. Generación de páginas en tiempo de ejecución

Igual que ocurre en la generación de subtemas en tiempo de ejecución, en las páginas sucede que en vez de tener una ListView, se tiene un ViewPager con un adaptador para crear las pestañas, que serán los contenedores de los Fragments Página.

Con el FragmentPagerAdapter, se puede saber en cualquier momento qué Fragment está en cada pestaña y el número total de pestañas del TabLayout:

```
public class PaginasPagerAdapter extends FragmentPagerAdapter {

    private int numTabs;
    private List<Fragment> tabList;

    public PaginasPagerAdapter(FragmentManager fm, int numTabs, List<Fragment> tabList) {
        super(fm);
        this.numTabs = numTabs;
        this.tabList = tabList;
    }

    @Override
    public Fragment getItem(int i) {
        // recibimos la posición por parámetro y en función de ella devolvemos el Fragment
correspondiente a esa sección.
        if (tabList.get(i) != null) {
            // Return a tab we created earlier..
            return tabList.get(i);
        }
        return null;
    }


    @Override
    public int getCount() {
        return numTabs;
    }

}
```

En sortAndReturnSections() creamos los objetos del dominio Página, ordenándolos por índice y en showJSON() se crean las pestañas del TabLayout. Para rellenar las pestañas, se pondrá el Fragment correspondiente por el orden de página, y mirando el tipo, se creará un Pagina0Fragment, un Pagina1Fragment o un Pagina2Fragment:

```
for(int i = 0; i < numTabs; i++) {
    Fragment tabFragment;
    System.out.println("ENTER FOR");
    System.out.println(items.get(i).getTitle());
    System.out.println(items.get(i).getContent1());

    //0 -> CODE
    if(items.get(i).getTipo() == 0) {
        //pasar al fragment title, content1, code, content2
        tabFragment = new Pagina0Fragment();
        myFragments.add(tabFragment);

    //1 -> IMAGE
    } else if(items.get(i).getTipo() == 1) {
        //pasar al fragment title, content1, url_image, content2
        tabFragment = new Pagina1Fragment();
```

```
            myFragments.add(tabFragment);

        //2 -> IMPORTANT
        } else {
            //pasar al fragment title, content1, content important
            tabFragment = new Pagina2Fragment();
            myFragments.add(tabFragment);
        }
    }
```

Con ésta implementación lo que se quiere conseguir es tener diferentes layouts que muestren contenido diferente, por ejemplo, Pagina0Fragment contiene el título de la página, 2 contenidos y contenido en forma de código, o Pagina1Fragment tendrá un título, 2 contenidos y una imagen, etc. De ésta manera, se pueden ir añadiendo tipos de Fragment i en tiempo de ejecución decidir cual se va a crear en ésa pestaña. A la larga, se debería implementar un patrón Factoría para controlar la creación de éstos Fragments.

7.4.5. Generar palabras del glosario en tiempo de ejecución

Al igual que en la generación de subtemas en tiempo de ejecución, se generan las palabras dentro de una ListView, pero ésta vez se añade una funcionalidad nueva: los headers. Se necesita de un PalabrasAdapter para saber si el dato que se recoge va a ser objeto de la ListView (palabra plana) o header (título que engloba un conjunto de palabras planas):

```
public class PalabrasAdapter extends ArrayAdapter<Palabra> {

   LayoutInflater inflater;
   private Context context;
   ArrayList<Palabra> items;

   public PalabrasAdapter(Context context, ArrayList<Palabra> items) {
      super(context, 0, items);
      this.context = context;
      inflater = (LayoutInflater)
context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
   }

   @Override
   public View getView(int position, View convertView, ViewGroup parent) {
      Palabra palabra = getItem(position);

      //If the cell is a section header we inflate the header layout
      if(palabra.isSectionHeader()) {
         convertView = inflater.inflate(R.layout.word_header, null);

         TextView header = (TextView) convertView.findViewById(R.id.category);

         header.setText(palabra.getCategory());

         convertView.setClickable(false);
      }
      else {
         convertView = inflater.inflate(R.layout.word_item, null);

         TextView name = (TextView) convertView.findViewById(R.id.word);
```

```
            TextView definition = (TextView) convertView.findViewById(R.id.definition);

            name.setText(palabra.getWord());
            definition.setText(palabra.getDefinition());

            convertView.setClickable(false);
        }
        return convertView;
    }
}
```

Entonces en GlosarioFragment, al hacer el sortAndAddSections, las palabras de tipo cell se añaden a la ListView dentro de su header, preguntando cuál es la categoría, y si ésa categoría no existe, se crea dinámicamente un header:

```
private ArrayList<Palabra> sortAndAddSections(ArrayList<Palabra> itemList) {

    ArrayList<Palabra> tempList = new ArrayList<Palabra>();

    //First we sort the array
    Collections.sort(itemList);

    //Loops through the list and add a section before each sectioncell start
    String header = "";
    for(int i = 0; i < itemList.size(); i++) {
        //If it is the start of a new section we create a new listcell and add it to our
array
        if(!header.equals(itemList.get(i).getCategory())){
            Palabra sectionCell = new Palabra("", "", itemList.get(i).getCategory());
            sectionCell.setToSectionHeader();
            tempList.add(sectionCell);
            System.out.println("Palabra: "+itemList.get(i).getWord()+ " Categoria: "+
itemList.get(i).getCategory());
            header = itemList.get(i).getCategory();
        }
        tempList.add(itemList.get(i));
    }

    return tempList;
}
```

De esta forma, se pretende poder crear categorías en tiempo de ejecución, además de obtener las palabras de la Base de Datos. Las palabras se ordenan por orden alfabético dentro de su header.

7.4.6. Implementación Login Google

En el apartado 7.4.1. se ha comentado que la MainActivity es la encargada del Logout de Google. Ahora bien, antes de poder hacer Logout, se ha de hacer Login. Para ello, tenemos la clase LoginFragment que implementa el GoogleApiClient, para obtener los datos del usuario y poder ponerlos en las SharedPreferences para que la MainActivity, el PerfilFragment y el ContactoFragment los recojan.

Se debe conectar con la API de Google:

```
//Initializing google api client
mGoogleApiClient = new GoogleApiClient.Builder(getActivity())
        .enableAutoManage(getActivity(), this)
        .addApi(Auth.GOOGLE_SIGN_IN_API, gso)
        .addApi(Plus.API)
        .build();

mGoogleApiClient.connect();
```

Al hacerlo, se obtiene la instancia que puede obtener los datos del usuario cuando apretamos el botón de Google Login y seleccionamos un email:

```
btnSignIn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {

        if(Utils.isNetworkAvailable(getActivity(), true, false)) {
            Intent signInIntent = Auth.GoogleSignInApi.getSignInIntent(mGoogleApiClient);
            startActivityForResult(signInIntent, RC_SIGN_IN);
        }
    }
});
```

La obtención de datos se debe hacer en el método onActivityResult(), dónde además los guardamos en las SharedPreferences:

```
//Getting google account
GoogleSignInAccount acct = result.getSignInAccount();

// We are signed in!
// Retrieve some profile information to personalize our app for the user.
try {
    String name = acct.getDisplayName();
    String email = acct.getEmail();
    String url = acct.getPhotoUrl().toString();

    editor = prefs.edit();
    editor.putString("name", name);
    editor.putString("email", email);
    editor.putString("url", url);
    editor.commit();

    Intent intent = new Intent(getActivity(), MainActivity.class);
    startActivity(intent);
    getActivity().finish();
...

}
```

Entonces, es la MainActivity la que se encarga de mostrar los datos del usuario en el header del NavigationView, volviendo al HomeFragment y cambiando la interfaz, por ejemplo, el texto del botón de “Sign In” pasa a ser “Sign Out”, etc.

Ésta parte de la aplicación ha sido realmente complicada de implementar, ya que, si operar con la API de Google de por sí ya es complicado, hacer el Login en un Fragment y el Logout en una Activity es otro nivel. La funcionalidad de "stay login" (que los datos del usuario aparezcan al cambiar y volver de otra activity que no sea la MainActivity o al salir y volver a entrar a la aplicación) funciona correctamente gracias a las SharedPreferences y al control que la MainActivity de la sesión de Google. Aquí se ha aprendido mucho sobre el ciclo de vida de una Activity y el ciclo de vida de un Fragment y cuán importante es tener claro que hace cada función del ciclo de vida (onCreate(), onResume(), onPause(), onActivityResult(), onStop(), onDestroy()…)

7.4.7. Vincular Perfil y Contacto con Login

Cuando el usuario se loguea, se pretende que tenga un Perfil en el que aparezca la imagen, el nombre y el email y dónde en una futura ampliación de la aplicación pueda ver su progreso en los temas y los logros que ha conseguido. Para pasar la información a ése PerfilFragment y de paso al ContactoFragment (formulario de contacto con la creadora de la aplicación), se obtienen los datos del SharedPreferences, que a su vez el que los ha colocado ha sido el LoginFragment. Entonces, si el usuario está logueado podrá ver sus datos en la pantalla de Perfil y su nombre y email se pondrán automáticamente en el formulario de Contacto, facilitándole el trabajo de rellenarlo. En el apartado *7.8. Problemas encontrados* se expondrá cómo se accede a las SharedPreferences desde el código.

7.4.8. Formulario y envío de correo en Contacto

Cuando el usuario aprieta la opción "Contacto" del NavigationView, se crea un nuevo Fragment que contiene el siguiente método:

```
private void send() {
    //Getting content for email
    String name = feedback_name.getText().toString();
    String email = feedback_email.getText().toString();
    String feedback = feedback_body.getText().toString();
    String feedbackType = feedback_type.getSelectedItem().toString();
    boolean bRequiresResponse = feedback_response.isChecked();

    StringBuilder sb = new StringBuilder();
    sb.append("Tipo de feedback: " + feedbackType + "\n\n");
    sb.append("De parte del usuario: " + name + "\n");
    sb.append("con email: " + email + "\n\n");
    sb.append(feedback);
    if(bRequiresResponse) {
```

```
        sb.append("\n\nRequiere una respuesta.");
    }

    String subject = "AndroidUp Feedback";
    String message = sb.toString();

    //Creating SendMail object
    SendMail sm = new SendMail(getActivity().getApplicationContext(), subject, message);

    //Executing sendmail to send email
    sm.execute();

    Toast.makeText(getActivity().getBaseContext(),"Feedback sent",
Toast.LENGTH_SHORT).show();
    showHomeFragment();
}
```

El método send() recoge la información del contacto_layout.xml y monta un email que será enviado con la ayuda de la librería Java Mail a andup.info@gmail.com.

7.4.9. Compartir la aplicación

En un principio se pensó de implementar ésta funcionalidad sólo dando la opción de compartir la aplicación por Whatsapp, creando un Fragment específico que tenía la función siguiente y atacaba directamente la API de Whatsapp:

```
public void sendMessage(View v) {

    String whatsAppMessage = message.getText().toString();

    Intent compartirIntent = new Intent();
    compartirIntent.setAction(Intent.ACTION_SEND);
    compartirIntent.putExtra(Intent.EXTRA_TEXT, whatsAppMessage);
    compartirIntent.setType("text/plain");

    try { //Abrir Whatsapp
        compartirIntent.setPackage("com.whatsapp");
        startActivity(compartirIntent);
    } catch(Exception e) {
        Toast.makeText(getActivity(), "Aplicación Whatsapp no encontrada en el
dispositivo.", Toast.LENGTH_LONG).show();
    }
}
```

Pero más tarde, se consideró la posibilidad de añadir directamente un *chooser* cuando el usuario aprieta la opción “Comparte” del NavigationView, que hiciera aparecer todas las aplicaciones que el usuario tenga en el móvil y que se consideren automáticamente aptas para compartir la aplicación:

```
case R.id.nav_comparte:
//                fragment = new CompartirFragment();
```

```
//                setCheckedAndChekacle(R.id.nav_comparte, true);
//                title = getString(R.string.comparte_title);
                Intent compartirIntent = new Intent(Intent.ACTION_SEND);
                compartirIntent.setType("text/plain");
                compartirIntent.putExtra(Intent.EXTRA_TEXT, "Instala ya la aplicación de
tutoriales Android: AndroidUp! (URL here)");
                startActivity(Intent.createChooser(compartirIntent,
getString(R.string.comparte_title)));
                break;
```

Con ésta solución, si el usuario no tiene una aplicación concreta potencialmente considerada para compartir, no le aparecerá en el chooser, así que no se han de controlar excepciones.

7.4.10. Valorar la aplicación

Para implementar ésta funcionalidad, simplemente se crea un Intent en la MainActivity cuando el usuario aprieta la opción “Valora” del NavigationView, que parsea una URL de Play Store y hace que aparezca el market delante de la aplicación con el siguiente código:

```
case R.id.nav_valora:
    /* context.getPackageName() -> com.miyu.androidup */
    Uri uri = Uri.parse("market://details?id=" + AppConstant.PACKAGE_NAME_EXAMPLE);
    Intent goToMarket = new Intent(Intent.ACTION_VIEW, uri);

    goToMarket.addFlags(Intent.FLAG_ACTIVITY_NO_HISTORY |
            Intent.FLAG_ACTIVITY_NEW_DOCUMENT |
            Intent.FLAG_ACTIVITY_MULTIPLE_TASK);
    try {
        startActivity(goToMarket);
    } catch (ActivityNotFoundException e) {
        startActivity(new Intent(Intent.ACTION_VIEW, Uri.parse(
                "http://play.googageNamele.com/store/apps/details?id=" +
AppConstant.PACKAGE_NAME_EXAMPLE)));
    }
```

La URL en éstos momentos apunta al package del juego Clash of Clans, ya que la aplicación AndroidUp no está subida al market y se quería poner un ejemplo para ver la funcionalidad. En el *release* de la aplicación, sólo se debe cambiar el package de AppConstant a com.miyu.androidup:

```
public static final String PACKAGE_NAME_EXAMPLE = "com.supercell.clashofclans";
```

7.4.11. Idiomas de dispositivo

Para poder cambiar los títulos de la aplicación dependiendo del idioma que esté usando en ése momento el usuario, se debe crear otro recurso string.xml específico de idioma que contenga todas los name-string con el valor traducido al nuevo idioma.

default string.xml:

```
<string name="glosario_title">Glosario</string>
<string name="perfil_title">Perfil</string>
<string name="ajustes_title">Ajustes</string>
<string name="contacto_title">Contacto</string>
<string name="comparte_title">Comparte</string>
<string name="valora_title">Valora</string>
```

en-string.xml:

```
<string name="glosario_title">Glossary</string>
<string name="perfil_title">Profile</string>
<string name="ajustes_title">Settings</string>
<string name="contacto_title">Contact</string>
<string name="comparte_title">Share</string>
<string name="valora_title">Rate</string>
```

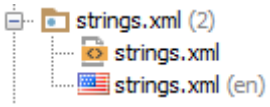

Fig. 7.7. Recurso res/values/strings.xml

7.4.12. Icono e imágenes de la aplicación

Con la ayuda de Edna Aniceto de Aniceto Studio [7], se ha creado un icono de la aplicación y unas imágenes para los temas de la pantalla principal.

Fig. 7.8. Icono AndroidUp

El icono se ha transformado en 5 .png, para las distintas densidades de pantalla de Android, para adaptarse lo máximo posible al dispositivo, que se escoge automáticamente:

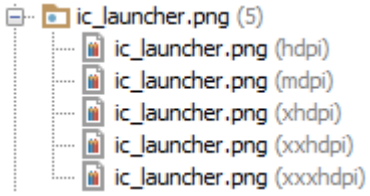

Fig. 7.9. Recurso res/mipmap/ic_launcher.png

Imágenes de los temas de AndroidUp:

Fig. 7.10. Imágenes de los temas de AndroidUp

Las imágenes se han subido a TinyPic [8], para poder vincular su URL en la Base de Datos y obtenerlas de la API.

7.5. Base de Datos

MongoDB, al crear el objeto, crea un índice de tipo ObjectID, los cuales utiliza de forma interna. Además de éstos índices, se han incluido otros índices únicos para posicionar el tema. En el caso de las palabras, el índice único es el nombre. Hasta el momento no se ha conseguido poner índices únicos a los subtemas y a las páginas, ya que sus valores se pueden repetir al estar dentro de la misma colección (el tema 1 puede contener un subtema 1, a la vez que el tema 2 puede contener un subtema 1), por lo tanto, se pueden poner dos subtemas con index = 2 en el mismo tema (sería un error).

Se tendría que buscar otra forma de evitar la repetición de subtemas dentro del mismo tema, o la repetición de páginas dentro del mismo subtema.

A continuación, se muestra cómo crear en MongoDB un índice único:

```
db.palabra.createIndex{{"nombre":1},{unique:true}}
```

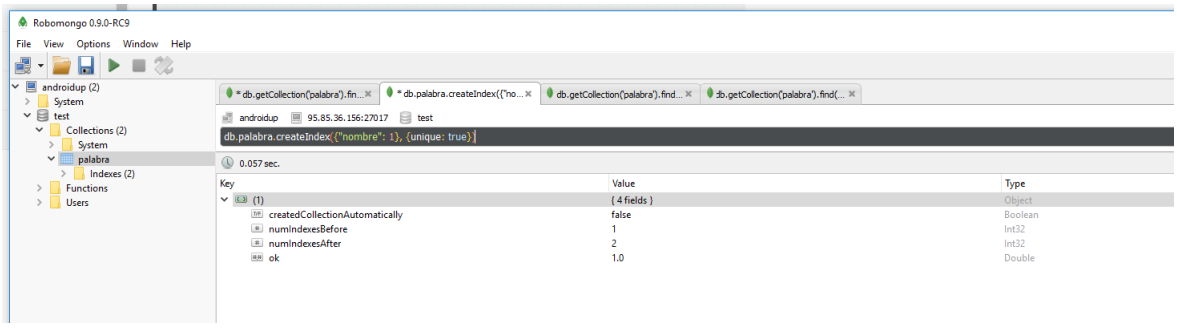

Fig. 7.11. Creación de índice único en MongoDB

Las colecciones de Usuario y Logro no se usan por el momento, ya que la implementación de la aplicación tal y como está en estos momentos no las requiere.

Se mostrarán en la siguiente página, las colecciones “tema” y “palabra” vistas desde la interfaz de Robomongo:

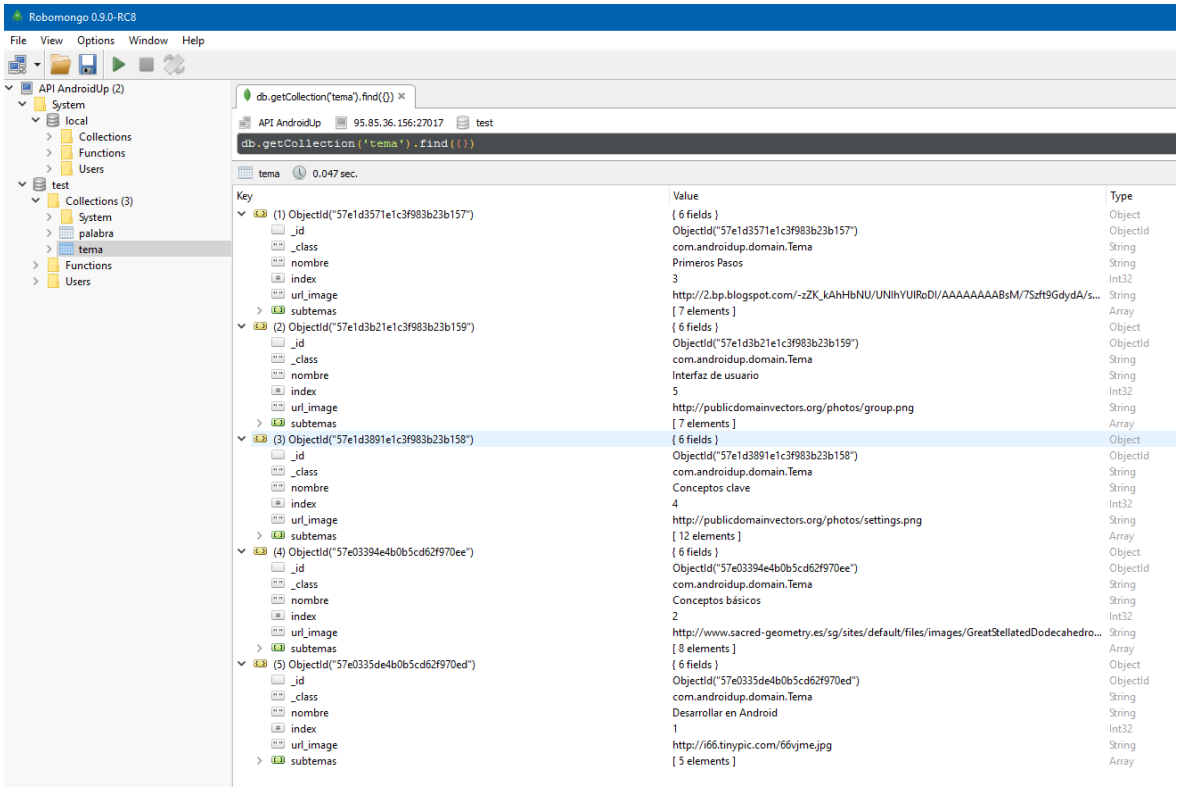

Fig. 7.12. Colección “tema” en Robomongo

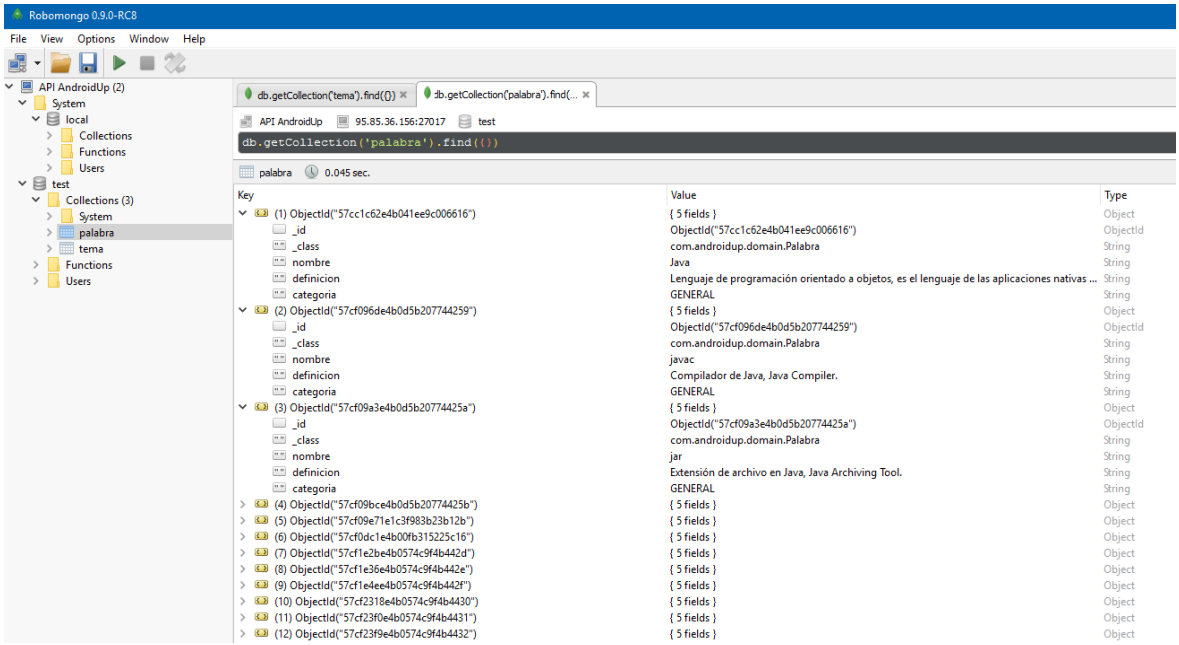

Fig. 7.13. Colección “palabra” en Robomongo

La siguiente imagen muestra la interfaz de Postman:

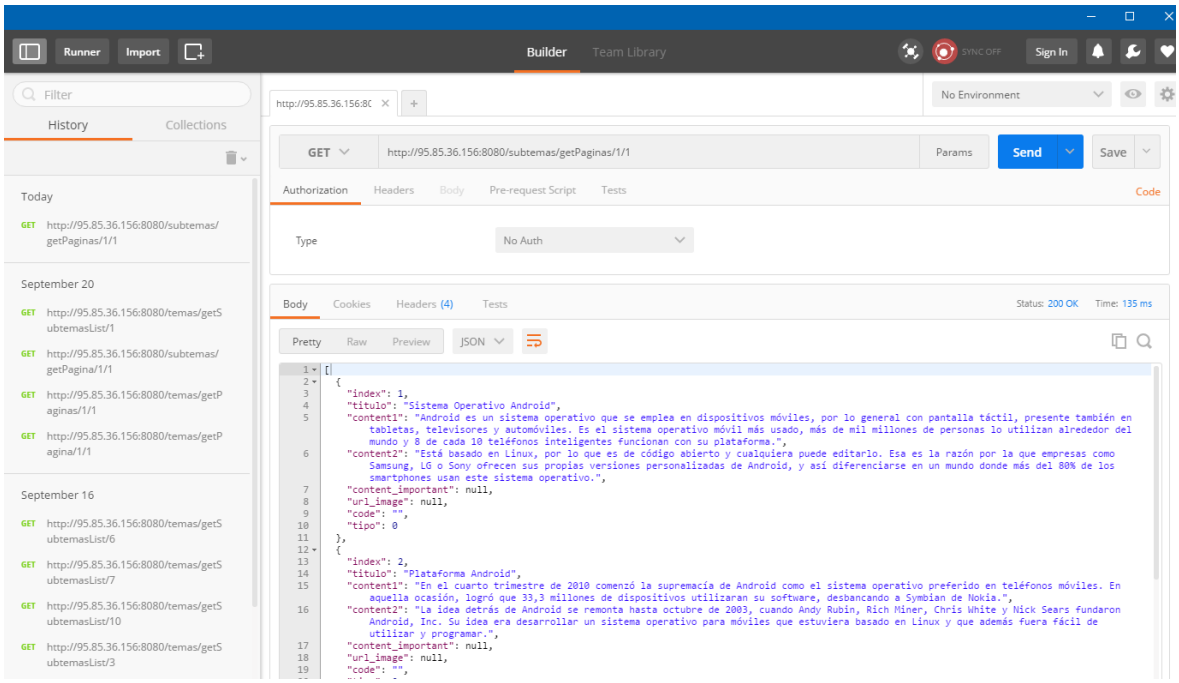

Fig. 7.14. Interfaz Postman

Se listarán a continuación, las operaciones GET que se ejecutan desde el código de la aplicación para llamar a la API REST y que ésta devuelva la información de la Base de Datos MongoDB:

- /temas/getTemasList (lista de temas)
- /temas/getSubtemasList/{id_tema} (lista de subtemas de un tema concreto)
- /palabras/getPalabras (lista de palabras)
- /subtemas/getPaginas/{id_tema}/{id_subtema} (lista de páginas de un subtema concreto)

7.6. Implementación API REST

La API ha utilizado los repositorios de datos de Spring. Estos repositorios proporcionan capacidades CRUD (Create, Read, Update, Delete) para las clases del dominio. A su vez, los repositorios gestionan las transacciones con la Base de Datos (abren y cierran conexión, buscan, editan, borran, etc.):

Estos repositorios permiten crear métodos específicos al implementar una interfaz ya proporcionada (en este caso, MongoRepository). Añadiendo la acción y el campo por el cual realizarla, al compilar la API se generan en tiempo de compilación los métodos necesarios. Por ejemplo:

Es necesario crear un método que encuentre un tema por su posición o índice (diferente al índice privado de MongoDB). Para ello, se declara el método "findByIndex(int index)":

```
1  package com.androidup.repositories;
2
3  import java.util.List;
4
5  import org.springframework.data.mongodb.repository.MongoRepository;
6  import org.springframework.data.repository.query.Param;
7  import com.androidup.domain.Tema;
8
9  //@RepositoryRestResource(collectionResourceRel = "temas", path = "temas")
10 public interface TemaRepository extends MongoRepository<Tema, String> {
11
12     List<Tema> findByNombre(@Param("nombre") String nombre);
13     Tema findByIndex(@Param("index") int index);
14
15     Tema deleteTemaByIndex(@Param("index") int index);
16
17 }
18
```

Fig. 7.15. Código findByIndex(int index) de Tema (API)

De esta forma, las interacciones básicas con la base de datos se pueden crear fácilmente. [9]

Además de los repositorios, se han implementado controladores REST que permiten adaptar las interacciones con la Base de Datos. Éstos controladores utilizan la notación de inyecciones (@Autowired) para acceder a dependencias tales como los repositorios. De ésta forma, sólo se ha tenido que implementar las operaciones CRUD con requerimientos específicos (obtener un listado de temas sin sus subtemas u obtener un listado de subtemas sin sus páginas):

```java
package com.androidup.controllers;

import java.util.ArrayList;

@RestController
@RequestMapping(value = "/temas")
public class TemaController {

    @Autowired
    TemaRepository temaRepository;

    @RequestMapping(value = "/getTemasList", method = RequestMethod.GET)
    public @ResponseBody List<Tema> getTemas() {
        List<Tema> list = temaRepository.findAll();
        for (Tema t : list){
            t.setSubtemas(null);
        }
        return list;
    }

    @RequestMapping(value = "/getSubtemasList/{tema}", method = RequestMethod.GET)
    public @ResponseBody List<Subtema> getSubtemasListConIndexTema(@PathVariable int tema) {
        Tema t = temaRepository.findByIndex(tema);
        List<Subtema> subtemas = t.getSubtemas();
        for (Subtema s : subtemas) {
            s.setPaginas(null);
        }
        return subtemas;
    }

    @RequestMapping(value = "/getSubtemas/{tema}", method = RequestMethod.GET)
    public @ResponseBody List<Subtema> getSubtemasConIndexTema(@PathVariable int tema) {
        Tema t = temaRepository.findByIndex(tema);
        return t.getSubtemas();
    }

    @RequestMapping(value = "/removeTema/{tema}", method = RequestMethod.GET)
    public @ResponseBody Tema removeTema(@PathVariable int tema){
        // TODO: Remove "tema" from "seguimiento" of all the users that had this "tema"
        return temaRepository.deleteTemaByIndex(tema);
    }

    @RequestMapping(value = "/removeSubtema/{tema}/{subtema}", method = RequestMethod.GET)
    public @ResponseBody Subtema removeSubtema(@PathVariable int tema , @PathVariable int subtema){
        Tema t = temaRepository.findByIndex(tema);
        List<Subtema> subtemas = t.getSubtemas();
        Subtema sub = null;
        for (Subtema s : subtemas) {
            if (s.getIndex() == subtema) {
                sub = s;
                subtemas.remove(s);
            }

        }
        t.setSubtemas((ArrayList<Subtema>) subtemas);
        temaRepository.save(t);
        return sub;
```

Fig. 7.16. Código Controlador de Tema (API)

Por último, la API contiene unas clases del dominio necesarias para poder encapsular los datos y generar la salida JSON:

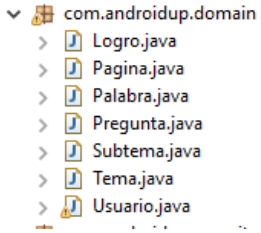

Fig. 7.17. Clases del dominio (API)

7.7. Control de Versiones

El control de versiones de la aplicación se ha hecho con GitHub. Concretamente, se ha usado el cliente de escritorio GitHub Desktop, que termite usar GitHub de forma más gráfica y sencilla. A continuación, se explicará cómo funciona éste cliente.

En el caso del código de la aplicación de AndroidUp, el repositorio es privado:

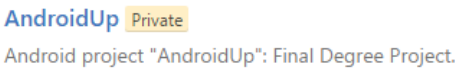

Fig. 7.18. Repositorio privado AndroidUp en GitHub

Y cuando en GitHub Desktop se vincula la cuenta GitHub personal, podemos clonar los repositorios o incluso crear nuevos desde aquí, tal como se muestra en la Fig. 7.19.:

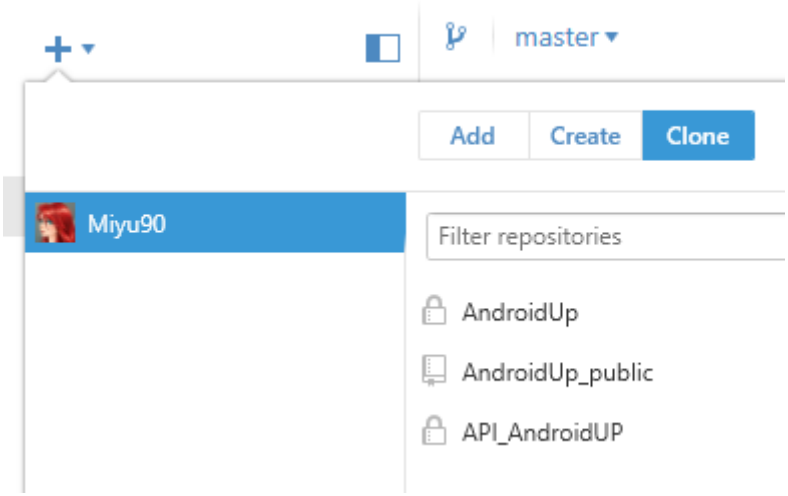

Fig. 7.19. Clonar repositorio de GitHub en GitHub Desktop

Se dispone de un historial de versiones, dónde igual que en la web de GitHub, se pueden ver los cambios efectuados entre commit y commit como, por ejemplo, clases Java añadidas, líneas de código borradas o añadidas, etc.

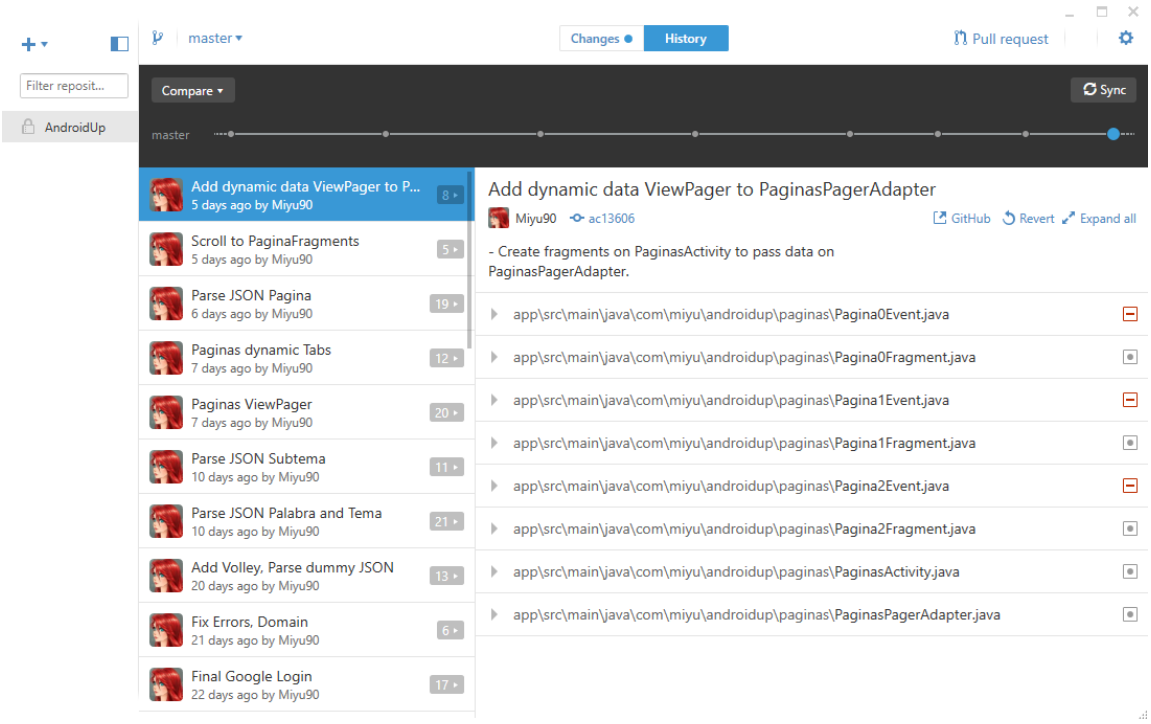

Fig. 7.20. Pestaña de Historial de versiones de AndroidUp en GitHub Desktop

En la Fig. 7.20. se puede ver los detalles del último commit de AndroidUp hasta la fecha, con lo que se ha modificado, añadido o eliminado y quién ha sido el autor del commit.

En la parte derecha, se aprecia el botón Sync, que funciona como un Pull, extrayendo el código del repositorio en la nube y actualizando el repositorio local con el último commit. Es potencialmente peligroso si no se quiere descartar los cambios locales, porque puede generar un conflicto de merge si no se ha trabajado con ramas, que es el caso.

No hace falta trabajar con ramas en éste proyecto, ya que es de una sola persona y no tendría mucho sentido.

Algunas veces se ha tenido que arreglar algún problema con el último commit realizado, y la siguiente instrucción:

```
git reset --hard HEAD~1
```

ha sido de vital importancia a la hora de deshacer el último commit del repositorio local, que se puede ejecutar en la Shell de GitHub Desktop, con la opción “Open in Git Shell”.

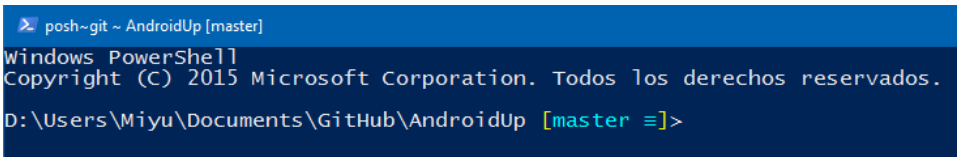

Fig. 7.21. Git Shell de AndroidUp (master)

Cuando se quiere hacer un commit, se debe ir a la pestaña “Changes”, dónde aparecen los cambios locales que hay respecto al repositorio en la nube. Como muestra la Fig. 7.22., en la parte inferior izquierda, se debe poner un título y una descripción para el commit, y luego apretar el botón “Commit to master”. Esto simplemente guarda los cambios, pero, además, se debe apretar el botón “Sync” de la parte superior derecha, que ésta vez funcionará como Push y subirá el código al repositorio en la nube:

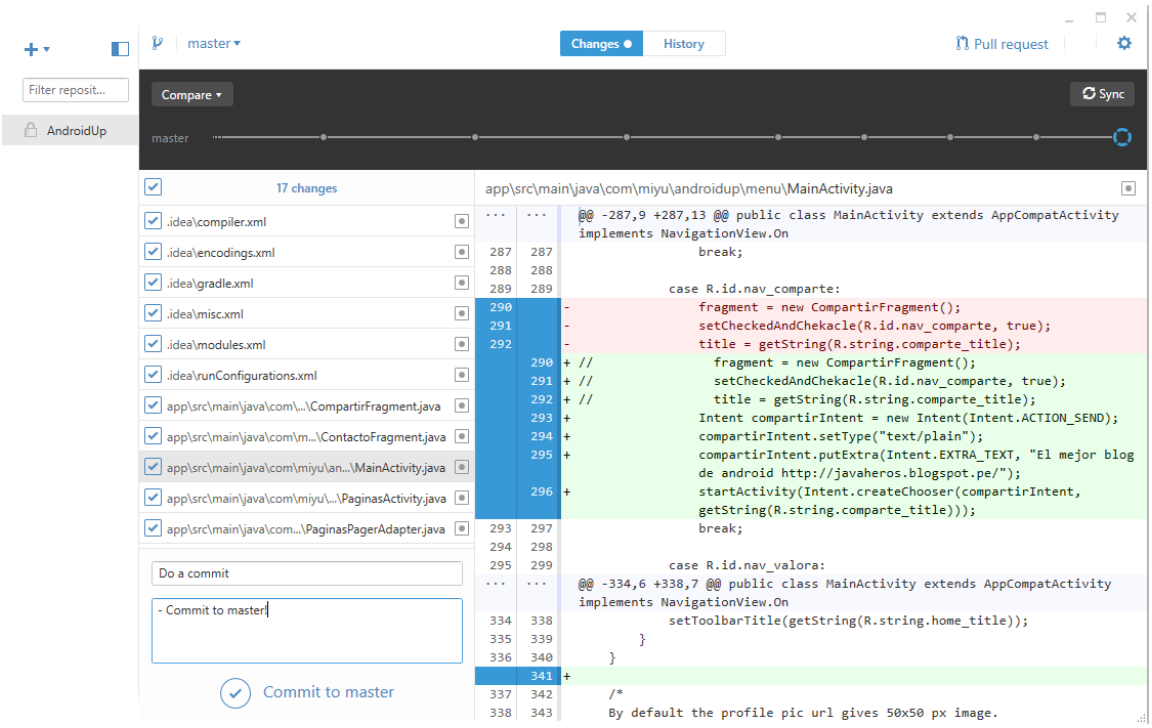

Fig. 7.22. Pestaña de cambios locales en el código de AndroidUp en GitHub Desktop

7.8. Problemas encontrados

Se muestran en éste apartado los problemas de implementación que se han generado a lo largo del proceso de desarrollo, y la forma en la que han sido solucionados.

7.8.1. Problema 1: Login de Google

Estado: Abierto

Sin duda, el mayor problema que se ha enfrentado en éste proyecto ha sido la implementación del Login con Google.

Primero, se pensó en hacerlo extrayendo directamente los datos del usuario de la API de Google y pasándolos del LoginFragment a la MainActivity con un EventBus. La idea no acabó de salir bien, ya que la información se mostraba, pero luego había problemas con el Logout y con la funcionalidad del "stay login" y al final se acabó descartando.

Luego se intentó que la MainActivity implementara una interface del Fragment y teniendo una clase GoogleClient que implementaba a su vez GoogleApiClient de Google, y pasaba lo mismo que con el EventBus: problemas en el Logout y el "stay loging".

Se llegó incluso a programar un patrón Observer con Estados (conectado, conectándose, desconectado) pero era muy complicado de mantener y aunque el Logout funcionaba, el stay login seguía sin funcionar y daba muchos problemas al salir de la aplicación, ya que los suscriptores se quedaban sin suscripción al volver a entrar.

La implementación final como se comentaba más arriba, se ha hecho haciendo que el LoginFragment sea el que implementa directamente la GoogleApiClient de Google y es él el que con su ciclo de vida controla el login y el que pone los datos en las SharedPreferences, para que la MainActivity recoja los datos de ahí y los ponga en el header de la NavigationView. Es mucho más sencillo que estar interactuando entre LoginFragment y MainActivity, además de hacer que la funcionalidad de "stay login" funcione perfectamente. Es la MainActivity la que sabe si se está conectado o no, y que desconecta al apretar el "Sign Out" y cambia la interfaz según el estado.

SharedPreferences en LoginFragment:

```
//Getting google account
GoogleSignInAccount acct = result.getSignInAccount();

// We are signed in!
// Retrieve some profile information to personalize our app for the user.
try {
    String name = acct.getDisplayName();
    String email = acct.getEmail();
    String url = acct.getPhotoUrl().toString();

    editor = prefs.edit();
    editor.putString("name", name);
    editor.putString("email", email);
    editor.putString("url", url);
    editor.commit();
...
```

Se tuvo que sacar el menú que le pregunta al usuario si acepta o rechaza las condiciones de Google al hacer login en la aplicación, ya que daban muchos problemas y se considera que, a la hora de instalar la aplicación, se muestran los permisos de cuentas y el usuario los ha aceptado con anterioridad. Estos primeros errores se han solventado, pero la implementación del Login de Google sigue abierta porque en la fase de Testing se encontraron 2 problemas, referentes al tratamiento de la imagen de usuario de Google en la interfaz gráfica:

- Si la cuenta de Google no tiene imagen, se genera un error: la aplicación hace como que se ha conectado pero los datos no aparecen en la NavigatorView, ni la interfaz cambia (ni siquiera va a la Home), dejando el botón en “Sign In”:

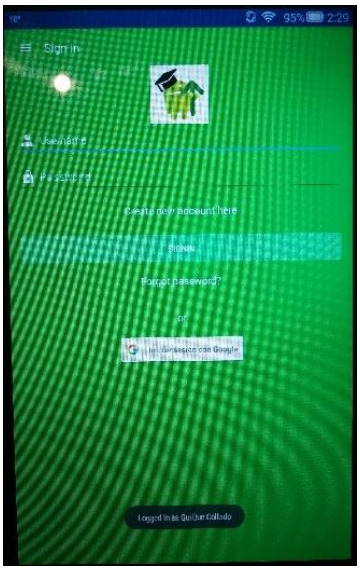

Fig. 7.23. Imagen de Error Login con Google cuando no hay imagen de perfil

Se tiene que investigar la causa del error, ya que, en la MainActivity, cuando se recoge la url de las SharedPreferences se llama a un método que con Picasso carga la imagen, y se especifica que, en caso de error, se ponga un placeholder:

```
Picasso.with(getApplicationContext())
        .load(profile_pic)
        .fit()
        .placeholder(R.drawable.placeholder)
        .error(R.drawable.placeholder)
        .into(profile_image);
```

- En algunas versiones anteriores de Android, al hacer Logout, se queda la imagen de Google del usuario puesta en la CircleImageView:

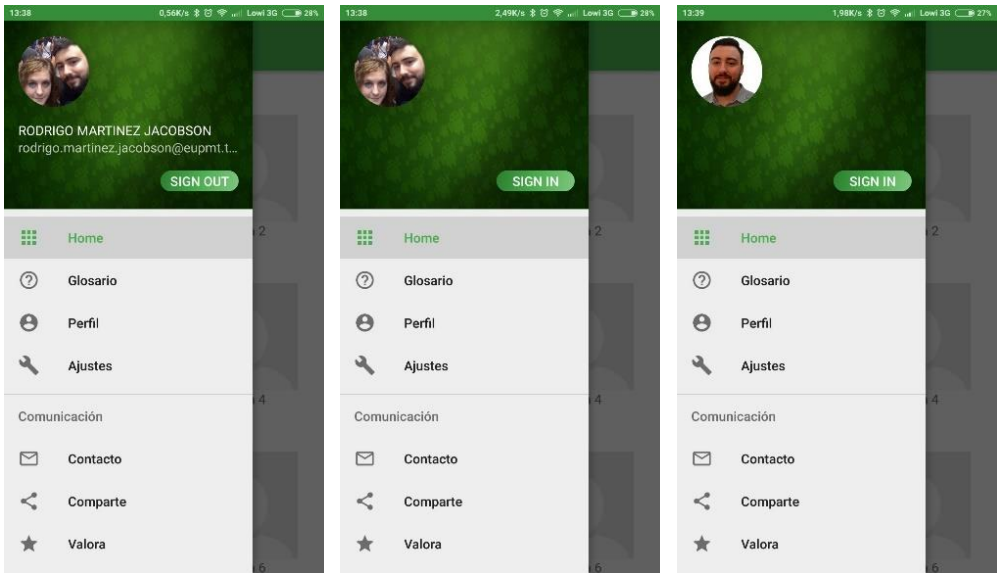

Fig. 7.24. Imágenes de error Login con Google en API 21

Se tiene que investigar la causa del error, ya que, en la MainActivity, cuando hace un update de la UI, se cambia la imagen de Google por una trasparente, una solución que si funciona con el nivel de API 22:

```
int id = getResources().getIdentifier("@drawable/transparent.png", null, null);
profile_image.setImageResource(id);
```

La parte de captura de la imagen cuando la cuenta no posee imagen deja abierto el problema con el Login de Google.

7.8.2. Problema 2: Vincular Perfil y Contacto con datos de usuario

Estado: Solventado

El problema con esta funcionalidad surgía a la par que con la del Login de Google. Cuando se implementó con EventBus, también se pasaba un EventBus al PerfilFragment y al ContactoFragment para que recibieran la imagen, el nombre y el email del usuario en Perfil y el nombre y el email en Contacto. Al cambiar a un Observer, estas dos Fragments se volvieron también subscriptores. Se destaca que funcionaba la actualización de datos, pero al hacer Logout había problemas. Con la implementación final del Login de Google con SharedPreferences, ahora éstas dos clases extraen los datos de ahí, y como el Perfil no se puede ver si el usuario no está conectado y en Contacto, si están vacías las SharedPreferences, se coloca una String vacía, el problema queda resuelto.

PerfilFragment:

```
profile_name.setText(prefs.getString("name", ""));
profile_email.setText(prefs.getString("email", ""));
setProfilePic(prefs.getString("url", ""));
```

ContactoFragment:

```
feedback_name.setText(prefs.getString("name", ""));
feedback_email.setText(prefs.getString("email", ""));
```

7.8.3. Problema 3: Java Mail

Estado: Solventado

Ocurrió un atasco de implementación: la librería Java Mail no funcionaba. El código no daba ningún error en tiempo de ejecución, y al depurarlo llegaba a la línea de envío del email y lo enviaba, pero el correo nunca llegaba. Buscando por internet, se consiguió hacer que funcionara modificando la seguridad de la sesión del correo electrónico al que llegan los emails de AndroidUp (andup.info@gmail.com), permitiendo el acceso de aplicaciones menos seguras:

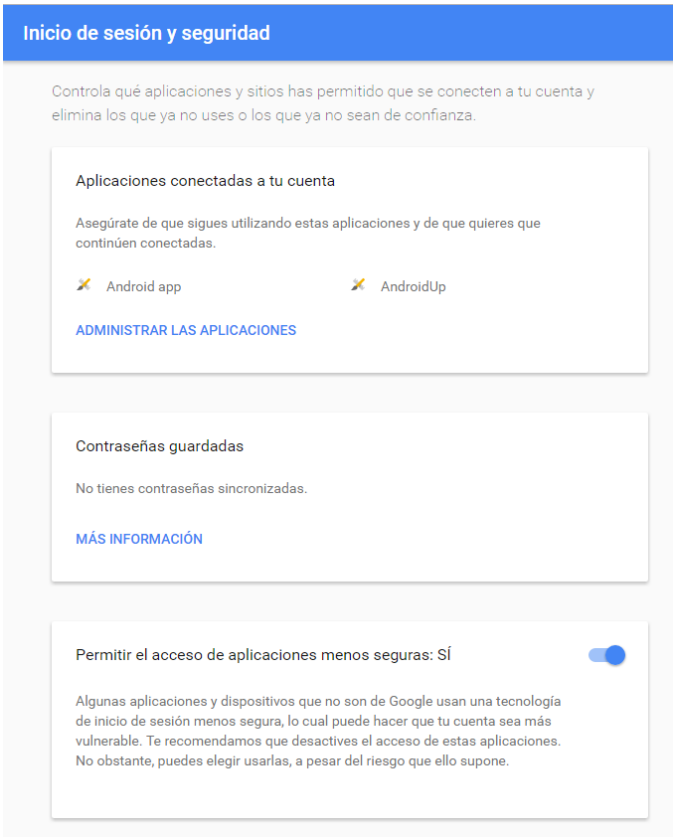

Fig. 7.25. Seguridad de la sesión andup.info@gmail.com

7.8.4. Problema 4: Crear dinámicamente una CircleImageView

Estado: Cerrado

Cuando se empezó a implementar la parte de crear las imágenes y títulos de los temas dinámicamente en el HomeFragment, hubo un problema con las CircleImageView: en tiempo de ejecución no se podían crear, seguramente relacionado con que deban ser final para tener diferentes ids y ser diferentes objetos, ya que al clicarlos deben responder de forma diferente, cada uno con su id correspondiente. Se han sustituido por ImageView normales por el momento.

El caso se ha cerrado y no solventado para dar la posibilidad de reabrirlo más adelante si de verdad se ve necesaria la implementación de CircleImageViews dinámicas por temas de diseño.

7.8.5. Problema 5: Obtención, parseado y exposición de datos con la API

Estado: Solventado

Había que decidir cómo se importaría y se trataría el JSON que se recibe de la API. Siguiendo un tutorial, se decidió crear una clase ParseJSON- por cada clase del dominio a parsear, para que, al llamarla, ella transformara los datos buscando las key de los JSON y devolviera una colección de objetos de ése tipo del dominio ya parseados, para así en la activity o fragment principal tratarlos cómo se necesitara.

7.8.6. Problema 6: Carga de imágenes externas

Estado: Solventado

Hay infinidad de soluciones para cargar imágenes externas en Android. La más extendida antes era con BitMap y Bitmap Factory, o implementando una clase específica con la que cargar las imágenes. El problema era decidir cómo se iba a hacer la implementación, pero se encontró la librería Picasso, que facilita mucho la tarea de carga de imágenes externas, y todo el trabajo que daba antes ésta funcionalidad.

7.8.7. Problema 7: Primera Pestaña de la Página

Estado: Abierto

Al entrar por primera vez en la actividad PaginasActivity, la primera pestaña que contiene el primer PaginaFragment, no se actualiza. Una vez pasamos a la segunda pestaña, y volvemos a la primera, la información aparece, actualizándose el Fragment. Éste problema no se ha podido resolver por el momento, se deberían de hacer pruebas con el ViewPager y el PaginasPagerAdapter para buscar una solución a éste error de actualización.

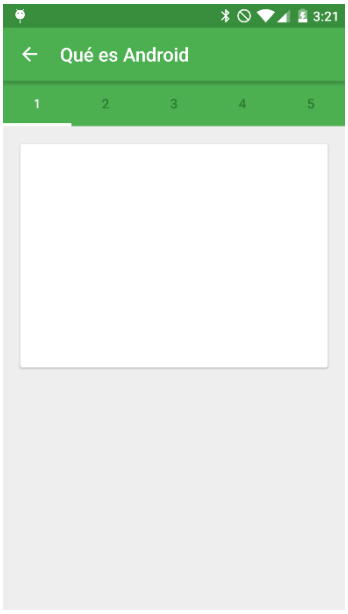

Fig. 7.26. Problema con Primera Pestaña de la Página

7.8.8. Problema 8: Formulario de Contactar

Estado: Abierto

En la tableta Teclast X98 Plus, el formulario del ContactoFragment no aparece a pantalla completa:

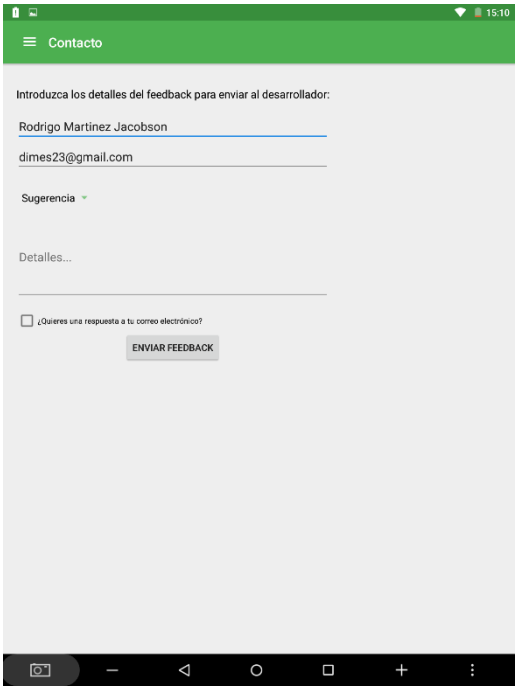

Fig. 7.27. Problema con Formulario de Contactar

Esto se puede deber al tamaño de la pantalla, al tener más pulgadas de que los dispositivos móviles, es probable que se deba hacer una carpeta dimen.xml específica para pantallas más grandes.

7.8.9. Problema 9: Pantalla de Login y teclado

Estado: Abierto

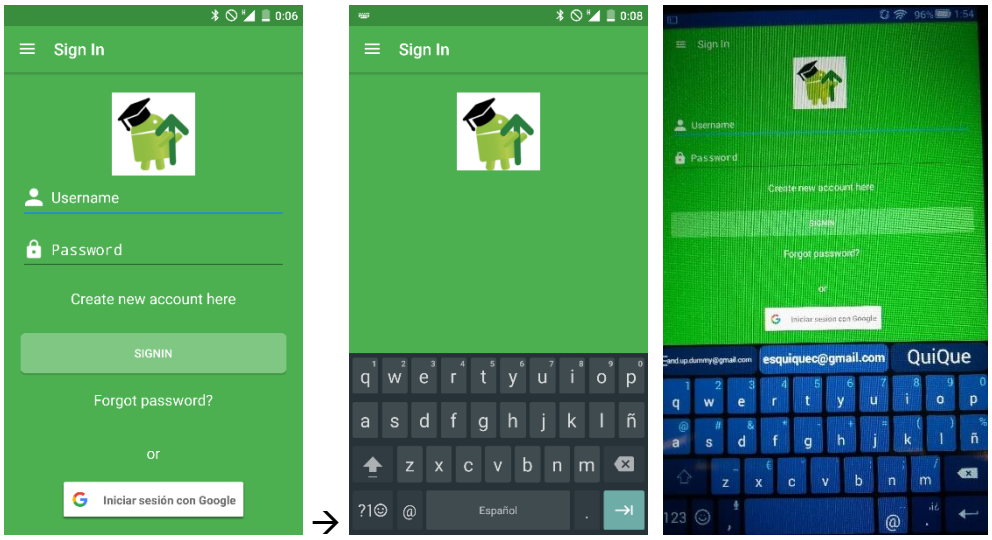

Fig. 7.28. Problema con teclado en LoginFragment

7.8.10. Problema 10: Controladores en la API

Estado: Solventado

El mayor problema que se encontró con la API fue estructurar los controladores, a causa de que la Base de Datos es MongoDB. Debido a la estructura documental, para acceder a un subtema hay que acceder primero al tema, por lo tanto, el controlador de subtemas requiere dependencias con el controlador de temas, y lo mismo pasa con las páginas, ya que una página es contenida por un subtema que a su vez es contenido por un tema. Los controladores al final se han estructurado, para separar las funciones de cada uno y dejar el código más repartido.

8. Testing

Se ha probado la aplicación en un total de 4 dispositivos diferentes, todos con versiones de Android diferentes, dos con API 22, uno con API 21 y otro con API 19. Se han podido detectar así, errores como el de la imagen de Google (apartado 7.8.1.), el error de dimensión en formulario de Contacto (apartado 7.8.8) y el error de teclado en la pantalla de Login (apartado 7.8.9). Se debería seguir testeando, sobre todo con APIs más antiguas, ya que en Android existen muchos problemas de compatibilidad y se debe intentar minimizarlos lo máximo posible. Se resalta el hecho de que el testing se ha hecho con versiones anteriores de código y que en la última versión de la aplicación las pantallas pueden haber cambiado el color de fondo de las Activities, los iconos de la pantalla principal, etc.

8.1. One Plus One X (Android 5.1.1, API 22)

El testing se ha hecho con éste móvil. Para no repetir imágenes se hará referencia al apartado *6.1. Diseño de la Interfaz*. Como se ha visto en el punto 6.1., los layouts se acoplan perfectamente al dispositivo, de 5 pulgadas y resolución 1920x1080 (441ppp). En este dispositivo no sale el error con la imagen de Google (apartado 7.8.1), ésta se borra cuando hacemos Logout. La prueba restante a comprobar es que desde ContactoFragment envíe un email de Feedback a andup.info@gmail.com, como se muestra a continuación:

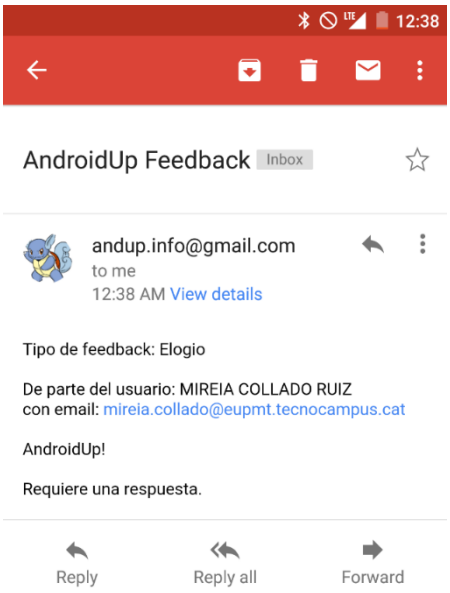

Fig. 8.1. Correo de Feedback recibido en andup.info@gmail.com

8.2. Teclast X98 Plus (Android 5.1, API 22)

En la Tablet Teclast X98 Plus de 9.7 pulgadas y resolución 2048x1536, la aplicación funciona perfectamente (a excepción del error del apartado 7.8.8), como puede verse a continuación en las capturas de pantalla realizadas en el dispositivo:

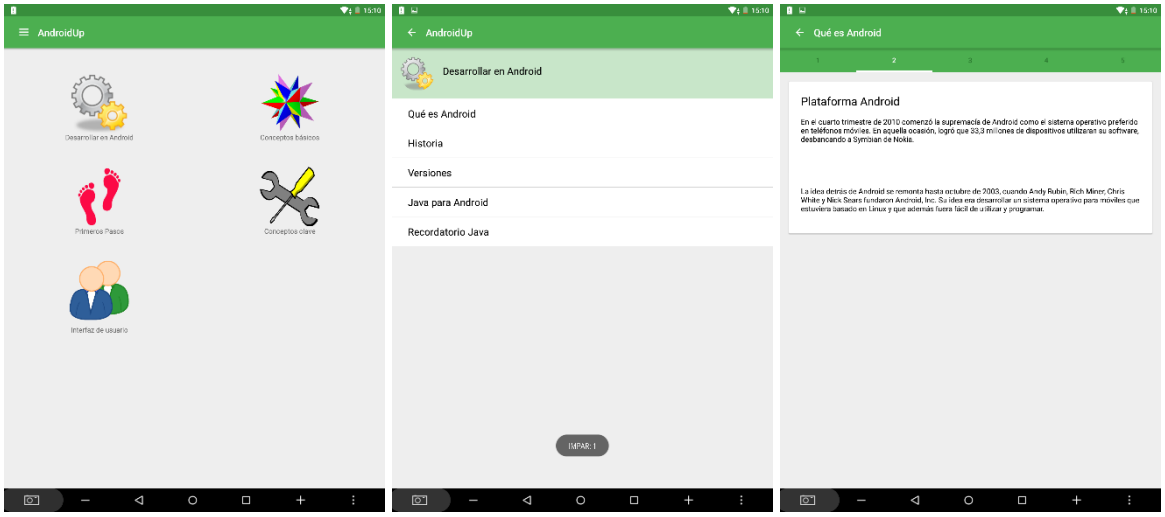

Fig. 8.2. Pantallas de Tema, Subtema y Página en Teclast X98 Plus (Testing)

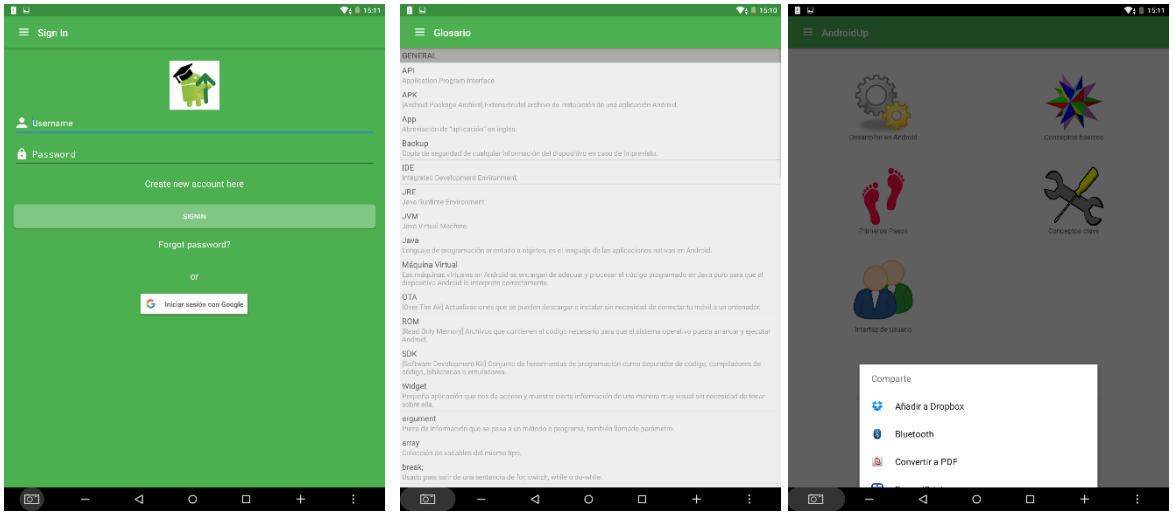

Fig. 8.3. Pantallas de Login, Glosario y Compartir en Teclast X98 Plus (Testing)

8.3. Xiaomi MI NOTE Pro (Android 5.0.2, API 21)

Se puede ver a continuación cómo se ven las pantallas en el Xiaomi MI NOTE Pro de 5.7 pulgadas y resolución 2560x1440 (515 ppi):

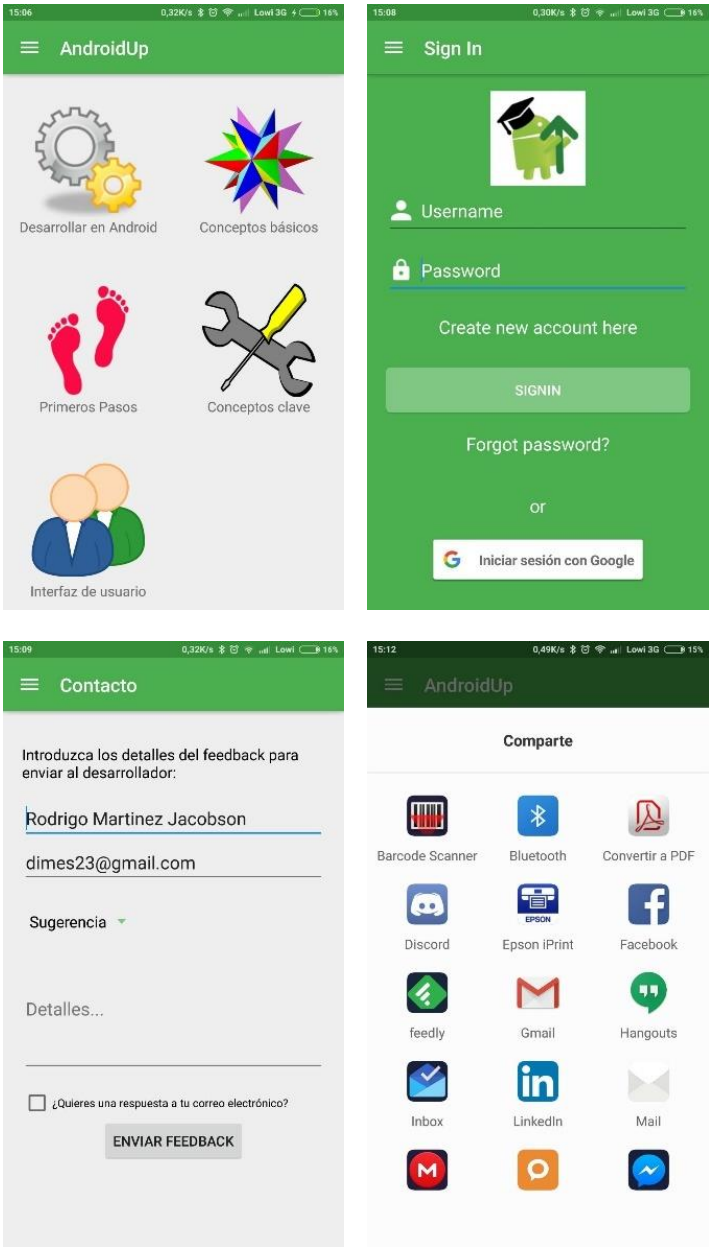

Fig. 8.4. Pantallas de Tema, Login, Contacto y Compartir en Xiaomi MI NOTE Pro (Testing)

En éste dispositivo se ha detectado uno de los errores del apartado 7.8.1: cuando se hace Logout, la imagen de Google sigue persistiendo en la aplicación.

8.4. HUAWEI T1 7.0 (Android 4.4.2, API 19)

A continuación, se aprecian las pantallas de la aplicación AndroidUp en el dispositivo Huawei T1 7.0, de 7 pulgadas y resolución 1024x600. El dispositivo no podía guardar imágenes, así que se han hecho fotos directamente a la pantalla:

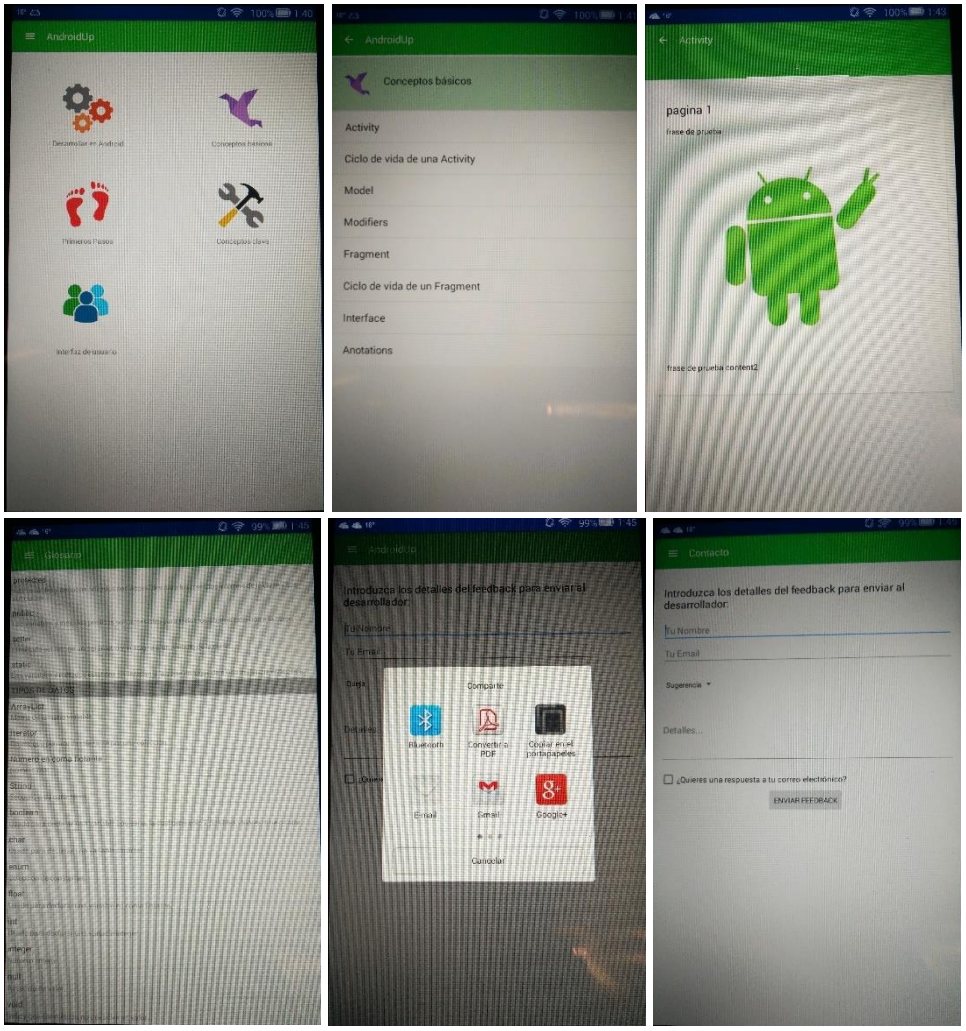

Fig. 8.5. Pantallas de Tema, Subtema, Página, Glosario, Compartir y Contacto en HUAWEI T1 7.0 (Testing)

En éste dispositivo se ha detectado uno de los errores del apartado 7.8.1: cuando se hace login con una cuenta que no tiene imagen de Google, el login no funciona.

9. Ampliaciones

Se detallan en éste apartado las posibles ampliaciones a realizar en la aplicación. Se destaca el hecho de que, por falta de tiempo, no se han podido llevar a cabo, pero posiblemente en las siguientes iteraciones de implementación de nuevo contenido se puedan añadir.

9.1. Ampliación idiomas de dispositivo

Ahora mismo tenemos 2 archivos string.xml, uno en español y otro en inglés, que hacen que cuando se entra a la app, elija automáticamente el español si el dispositivo tiene como idioma por defecto el español o inglés si tiene el idioma del dispositivo en inglés. Se debería ampliar el repertorio de idiomas disponibles, para tener los menús y temas en diferentes idiomas, con el fin de llegar a más usuarios.

9.2. Temario en varios idiomas

Fig. 9.1. Imagen Idiomas

La aplicación cuenta ahora únicamente de tutoriales en castellano, y se debería plantear la posibilidad de traducirlos al menos al inglés, para así conseguir más usuarios. Se debería implementar entonces un ajuste de idiomas en el apartado Ajustes, para que el usuario elija el idioma que más le convenga.

9.3. Notificaciones push

Implementación de notificaciones push en la aplicación, que recuerden al usuario cada cierto tiempo que abra la aplicación, para así mantener alto la ratio de tiempo de uso del aplicativo, con la opción para el usuario de habilitarlas y deshabilitarlas en el apartado Ajustes.

9.4. Sonidos in-app

Implementación de sonidos in-app, como por ejemplo al seleccionar un tema, al escoger un subtema, al navegar por las páginas, al escoger una respuesta correcta o incorrecta, entre otros, con su correspondiente ajuste que dé la posibilidad al usuario de habilitarlos o deshabilitarlos.

Fig. 9.2. Imagen Sonidos

9.5. Efectos de transición de pantalla

Añadir efectos de transición a las pantallas de la aplicación, con el fin de hacerla más dinámica y atractiva a la vista.

9.6. Justificación y estilos de texto en TextViews

Buscar una manera de justificar los textos en las TextViews, el método comentado de usar WebViews parece factible, además de poder añadir estilos al texto tales como resaltado, cursiva, color, etc. a través del texto en formato HTML guardado en la Base de Datos y recuperado a través de la API.

9.7. Poner Páginas de preguntas

Crear un tipo nuevo de páginas con preguntas sobre Android, para que el usuario las conteste. Sería la pregunta con una checkbox con 3-4 respuestas, sólo una correcta, y que se indique al usuario cuando éste apreté un botón si su respuesta es correcta o no.

9.8. Seguimiento del usuario

Implementar un sistema de seguimiento de temas y subtemas para el usuario, tanto en la API como en la aplicación, para así tener los datos del usuario en la base de datos y que éste pueda llevar un control sobre los temas y subtemas que ha terminado y por dónde se ha quedado en el tutorial una vez cierra y vuelve a abrir la aplicación. Se vería reflejado

Fig. 9.3. Imagen Seguimiento

tanto en el perfil del usuario como indicado con colores en las imágenes del tema y subtemas, con la posibilidad de incluir el total de subtemas de un tema y los ya superados en el menú principal, debajo del nombre del tema con el formato [número de temas superados] / [temas totales] y/o con una ProgressBar. Además, se estudiaría la opción de poner un sistema de bloqueo de temas, que se desbloquearían únicamente si el tema anterior ha sido superado, dándole a la aplicación potencial para poner micro pagos (pagar por desbloquear el siguiente tema, por ejemplo).

9.9. Sistema de Logros

Implementar un sistema de logros para hacer más ameno el aprendizaje. Contaría con un seguimiento de los mismos, para que el usuario pueda ver los logros superados y los que le quedan por conseguir.

Fig. 9.4. Imagen Logros

Los logros se podrán consultar en el perfil del usuario.

9.10. Implementación de un foro Q/A

Ésta es la ampliación definitiva: el contacto del usuario con otros usuarios que usan también la aplicación. Se podría implementar un foro de preguntas y respuestas en el menú lateral, dónde un usuario haga una pregunta y otros puedan responderle, con valoración de preguntas y respuestas, y reputación de usuario, parecido al sistema que tiene actualmente stackoverflow [10]. Está opción sería interesante de cara a ampliar el número de usuarios y la actividad dentro de la aplicación, convirtiéndola no sólo en una aplicación de aprendizaje, sino en una red social, así, el proyecto puede crecer e incluso llegar a más usuarios.

Fig. 9.5. Imagen Foro Q/A

10. Conclusiones

A continuación, se explicarán las conclusiones sacadas después de haber terminado el proyecto, como el estado final de desarrollo en el que se encuentra, la valoración de las herramientas usadas y la valoración personal de ésta experiencia.

10.1. Estado final del desarrollo

El estado final de desarrollo del proyecto en este momento es satisfactorio, y muy probablemente con un par de ajustes estaría listo para subirlo a Google Play. Se ha llegado muy lejos en cuánto a implementación. Aun así, es tan sólo un prototipo funcional por el momento, que hay que testear en más dispositivos para hacer algunos ajustes de versiones o arreglo de errores y añadir, en la medida de lo posible, alguna ampliación más antes de su lanzamiento definitivo, sobretodo de carácter estético.

10.2. Valoración de las herramientas usadas

En general, las herramientas usadas han dado resultados muy apropiados. Android Studio ha sido básico para programar en Android con Java, así como sus diferentes versiones de SDK. Aunque alguna vez ha tenido sus errores, no se puede negar su potencia y ayuda a la hora de escribir código. Gradle ha facilitado mucho la tarea de añadir librerías externas, siendo un builder muy estable y fácil de usar. Volley ha sido clave para la obtención de datos con la API, ha facilitado mucho la tarea. Librerías como Butter Knife, Picasso, CardView, CircleImageView, Java Mail, etc. han enriquecido el proyecto y lo han dotado de personalidad y facilidad en cuanto a implementación se refiere. GitHub ha sido una de las herramientas clave para el control de versiones, muy importante en este tipo de proyectos, que al estar sujetos a una fecha límite, no hay posibilidad de volver a empezar de nuevo si se pierde el código. Para el almacenamiento de la memoria se ha usado Google Drive, que no se ha considerado necesario explicarlo en el apartado de herramientas utilizadas. Aunque Asana como herramienta de planificación no ha sido apropiada para una sola persona, ha sido una experiencia, y da enfoque de futuro profesional, ya que muchas empresas utilizan éste tipo de sistema de tickets y se considera bueno haberlo estudiado. GitBook es muy

potente, no se ha podido sacar todo el potencial al no disponer de una libre exposición de la información de la documentación del proyecto. MongoDB ha dado resultados increíbles en cuanto a almacenamiento, haciendo exactamente lo que se necesitaba: guardar estructuras de datos en documentos que pueden llegar a tener diferente estructura. Robomongo ha facilitado la tarea con la Base de Datos y Postman ha ayudado a inyectar los JSON en la Base de Datos con su método POST, y con el GET para saber en todo momento qué iba a devolver la API, dando la posibilidad de probar sus consultas. En todo el tiempo que se ha utilizado Digital Ocean, el servidor no ha caído ni una sola vez. La API se ha realizado en Eclipse siguiendo tutoriales de Spring y usando Maven como builder, con una experiencia que deja buena impresión y disipa miedos en cuanto a crear una API REST sencilla.

10.3. Valoración personal

Se puede decir que la experiencia con este proyecto ha sido del todo satisfactoria. Es un proyecto complejo, dónde se partía de la base de tener únicamente conocimientos en Java, sin conocer nada del mundo Android. Personalmente, ésta experiencia ha ampliado los conocimientos adquiridos a lo largo del grado, no sólo en cuanto a programación, sino también en nuevas herramientas que serán útiles en el mercado laboral, por la gran demanda de empresas informáticas en crecimiento que las usan.

Y al final del proceso, el producto resultante es una aplicación móvil funcional y compatible con la plataforma Android, que permite al usuario la visualización de los tutoriales en temas y subtemas.

La sensación final es de haber aprendido muchas cosas. El recorrido ha sido largo y difícil, pero en éste entorno informático, son precisamente las situaciones complicadas las que te hacen crecer como programadora y en su defecto, como persona.

11. Referencias

[1] Android Developers, Dashboards, https://developer.android.com/about/dashboards/index.html

[2] Android Developers, Starting an Activity, https://developer.android.com/training/basics/activity-lifecycle/starting.html

[3] Android Developers, Activities, https://developer.android.com/guide/components/activities.html

[4] Android Developers, Fragments, https://developer.android.com/guide/components/fragments.html

[5] Tim Buchalka, Learn Android App Development With Java Step By Step (Udemy) https://www.udemy.com/android-marshmallow-java-app-development-course/

[6] Moqups, https://moqups.com/

[7] Edna Aniceto, Aniceto Studio, http://anicetostudio.com/

[8] TinyPic, http://es.tinypic.com/

[9] Spring docs, MongoDB repositories, http://docs.spring.io/spring-data/mongodb/docs/1.2.0.RELEASE/reference/html/mongo.repositories.html

[10] Stackoverflow community, http://stackoverflow.com/

Escola Universitària Politècnica de Mataró

Centre adscrit a:

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Grado en Ingeniería Informática

AndroidUp

Estudio económico

Mireia Collado Ruiz
PONENT: Catalina Juan Nadal

PRIMAVERA 2016

TecnoCampus Mataró-Maresme

Índice

1. Coste del prototipo

Se ha hecho un estudio económico aproximado de los gastos relacionados con el desarrollo del proyecto AndroidUp, que se detallará a continuación. El estudio considera los costes de diseño e implementación del proyecto (horas/hombre de un ingeniero, programador o diseñador dependiendo el caso), costes indirectos como luz, internet o datos y las amortizaciones de las herramientas usadas.

1.1. Costes de recursos humanos

Los costes de recursos humanos comprenden las horas de los profesionales que han intervenido en el proyecto. En éste caso, sólo hay una persona que ha hecho de ingeniero, programador y diseñador, y podría considerarse la posibilidad de poner un precio/hora único de freelance, pero para respetar las diferentes disciplinas, los cálculos se han hecho con los respectivos salarios, comentados a continuación.

1.1.1. Horas de Ingeniero

Las horas de ingeniero son aquellas relacionadas con el estudio de mercado y la definición de requerimientos, casos de uso y clases del dominio.

Considerando que un ingeniero ha de cobrar unos 40€/hora, salen reflejadas las horas dedicadas a cada parte y las totales, con sus respectivos costes y el coste final en la siguiente tabla:

Actividad	Horas dedicadas	Coste
Estudio de mercado	6 h	240 €
Definición de requerimientos	3 h	120 €
Definición de casos de uso	9 h	360 €
Definición de clases	2 h	80 €
Elección de las tecnologías a usar	5h	200 €
Total	**25 h**	**1.000 €**

Tabla 12.1. Tabla de Horas de Ingeniero

1.1.2. Horas de Desarrollo

Las horas de desarrollo están relacionadas con la programación del aplicativo, la API y la base de datos, además de la corrección de bugs y el testing. El programador se dedica también a documentar el proyecto.

Considerando que un programador ha de cobrar unos 30€/hora, salen reflejadas las horas dedicadas a cada parte y las totales, con sus respectivos costes y el coste final en la siguiente tabla:

Actividad	Horas dedicadas	Coste
Programación de la aplicación	320 h	9.600 €
Formación en herramientas y lenguajes	100 h	3.000 €
Programación de la API	8 h	240 €
Diseño de la Base de Datos	2 h	60 €
Generación temario	8 h	240 €
Inserción de contenido a la Base de Datos	6 h	180 €
Corrección de bugs	30 h	900 €
Testing	8 h	240 €
Documentación	160 h	4.800 €
Total	**642 h**	**19.260 €**

Tabla 12.2. Tabla de Horas de Desarrollo

1.1.3. Horas de Diseño

Las horas de diseño son las relacionadas con el diseño de la interfaz y las pantallas, la creación del logotipo de la aplicación entre otras imágenes y la aplicación de las consignas del Material Design de Google.

Considerando que un diseñador ha de cobrar unos 25€/hora, salen reflejadas las horas dedicadas a cada parte y las totales, con sus respectivos costes y el coste final en la siguiente tabla:

Actividad	Horas dedicadas	Coste
Diseño de la interfaz	5 h	125 €
Creación logotipo de la aplicación	1 h	25 €
Aplicación de Material Design	6 h	150 €
Creación de otras imágenes	3 h	75 €
Total	**15 h**	**375 €**

Tabla 12.3. Tabla de Horas de Diseño

1.1.4. Costes directos

Al final, nos salen los costes directos de Recursos Humanos. Los costes directos son aquellos vinculados al proceso de creación del producto, como son las horas de análisis, programación y diseño del aplicativo.

Concepto	Horas dedicadas	Coste
Horas de ingeniero	25 h	1.000 €
Horas de desarrollo	642 h	19.260 €
Horas de diseño	15 h	375 €
Total	**682 h**	**20.635 €**

Tabla 12.4. Tabla de Costes directos de RRHH

1.2. Amortizaciones de equipo y software

Es el material necesario que se ha utilizado para realizar el desarrollo de la aplicación Android. La mayoría de gastos son relacionados con equipos informáticos y licencias de software, añadiendo los costes mensuales del servidor Digital Ocena y el coste que supondría subir la app a Google Play.

Concepto	Precio	Vida útil	Tiempo usado	Amortización
Equipo informático	800 €	24 meses	6 meses	200 €
Dispositivo móvil Android	280 €	24 meses	6 meses	70 €
Servidor Digital Ocean	10 € / mes	-	4 meses	40 €
Licencia Google Play	25 €	-	pago único	25 €
Total				**335 €**

Tabla 12.5. Tabla de Costes en Amortizaciones

Equipo informático → 800/24 * 6 = 200

Dispositivo móvil Android → 280/24 * 6 = 70

1.3. Costes indirectos

Los costes indirectos afectan al proceso de creación del aplicativo. En este caso particular, se pueden considerar costes indirectos la electricidad, la conexión a internet y wifi y los datos móviles, todos ellos indispensables para poder crear el producto, además del uso de las instalaciones, como el alquiler de un local. Aquí concretamente se ha considerado poner un 23% de porcentaje sobre el coste total, ya que cualquier coste de luz o internet ha sido como se comentaba indispensable.

1.4. Coste de fabricación del Prototipo

Así, el coste final de fabricación del prototipo es el siguiente:

Concepto	Coste
Costes RRHH	20.635 €
Costes Amortizaciones	335 €
Subtotal	20.970 €
Costes indirectos (23%)	~ 4.823 €
Total	**25.793 €**

Tabla 12.6. Tabla de Coste de fabricación del Prototipo

2. Precio de venta en el mercado

La aplicación será de descarga gratuita. Se esperan tener entre 5.000 y 10.000 descargas para empezar, ya que el tema de la aplicación es un nicho de mercado muy reducido: el de los programadores Java que quieran pasarse a programar en móviles Android, pero sería un comienzo, hasta llegar a las 100.000 descargas que es el objetivo final. Se podría pensar en poner publicidad, considerando que sea lo menos invasiva posible, para no distraer al usuario de los tutoriales (un anuncio al entrar, uno por cada subtema superado, etc.).

Cuando se implemente la parte de temas bloqueados, se podrán desbloquear haciendo un micro pago, lo que dará mayor beneficio si los usuarios pagan por ello.

3. Conclusiones Estudio Económico

Las conclusiones que podemos sacar del Estudio Económico son que desarrollar una aplicación nativa en Android es muy costoso en cuanto a presupuesto y esto se debe a la gran dedicación en horas de análisis y desarrollo. Además, se debe destacar que el margen del coste total puede reducirse si tenemos en cuenta que la persona que ha desarrollado todo el proyecto es una programadora junior, y por lo tanto se han tenido que considerar horas de formación, más las horas de documentación para entregar una Memoria, además de las múltiples dificultades que se han encontrado en el camino de la implementación a causa precisamente de ser la primera vez que se trabaja con Android.

Escola Universitària
Politècnica de Mataró

Centre adscrit a:

UNIVERSITAT POLITÈCNICA
DE CATALUNYA

Grado en Ingeniería Informática

AndroidUp

Anexos

Mireia Collado Ruiz
PONENT: Catalina Juan Nadal

PRIMAVERA 2016

TecnoCampus
Mataró-Maresme

Índex.

Anexo I. Glosario del Aplicativo

GENERAL

Java
Lenguaje de programación orientado a objetos, es el lenguaje de las aplicaciones nativas en Android.

javac
Compilador de Java, Java Compiler.

jar
Extensión de archivo en Java, Java Archiving Tool.

jdb
Java Debugging Tool.

JRE
Java Runtime Environment.

JVM
Java Virtual Machine.

IDE
Integrated Development Environment.

argument
Pieza de información que se pasa a un método o programa, también llamado parámetro.

break;
Usado para salir de una sentencia de for, switch, while o do-while.

continue;
Termina con la iteración actual.

exception
Error o anomalía que se produce cuando un programa se está ejecutando.

array
Colección de variables del mismo tipo.

package
Grupo de tipos de clases similares o agrupadas por una razón concreta.

API
Application Program Interface.

ROM
[Read Only Memory] Archivos que contienen el código necesario para que el sistema operativo pueda arrancar y ejecutar Android.

SDK
[Software Development Kit] Conjunto de herramientas de programación como depurador de código, compiladores de código, bibliotecas o emuladores.

APK
[Android Package Archive] Extensión del archivo de instalación de una aplicación Android.

App
Abreviación de “aplicación” en inglés.

Backup
Copia de seguridad de cualquier información del dispositivo en caso de imprevisto.

OTA

[Over The Air] Actualizaciones que se pueden descargar e instalar sin necesidad de conectar tu móvil a un ordenador.

root
Acceso a partes del software del teléfono a los que normalmente no se puede acceder, permitiendo modificar ciertas partes del software o instalar aplicaciones y ROMs.

Widget
Pequeña aplicación que nos da acceso y muestra cierta información de una manera muy visual sin necesidad de tocar sobre ella.

Máquina virtual
Las máquinas virtuales en Android se encargan de adecuar y procesar el código programado en Java puro para que el dispositivo Android lo interprete correctamente.

POO

instance
Cada objeto es llamado una instancia de una clase.

constructor
Métodos especiales que son usados para inicializar objetos y que se invocan al tiempo de una creación de objeto.

method
Colección de sentencias que son agrupadas para ejecutar una operación. También llamado function (función).

encapsulation
Empaquetar datos y métodos en un solo componente.

inheritance
Proceso mediante el cual una clase hereda los miembros y métodos de otra clase.

polymorphism
Al llamar a un método miembro, causará que otro método diferente se ejecute en función del tipo de objeto que invoca el método.

abstract class
Si una clase es declarada como abstracta, no puede ser instanciada y se debe heredar de otra clase. Cualquier clase que contenga un método abstracto debe definirse como abstracta.

interface
Clase completamente abstracta que sólo contiene métodos abstractos.

extends
Utilizado para heredar una subclase de otra clase.

implements
Utilizado para implementar clases abstractas.

getter
El método get devuelve el valor de un atributo de la clase.

setter
El método set recoge un parámetro y lo asigna a un atributo de la clase.

static
Las variables y métodos estáticos pertenecen a la clase en lugar de a una instancia específica.

public
Las variables y métodos públicos son accesibles para todos los objetos externos a la clase.

private
Las variables y métodos privados son accesibles sólo para los métodos internos de la clase.

protected

Las variables y métodos privados son accesibles para los métodos internos de la clase y sus subclases.

member

Las variables y métodos miembro son asociadas a un objeto específico y accesibles para todos los métodos del objeto.

Variable local

Variables que sólo existen dentro de un método o función y que dejan de existir una vez que la función es ejecutada. Se vuelven a crear cada vez que se llama o se ejecuta a la función. No son accesibles para otras funciones o para el programa principal.

Variable global

Variables declaradas en el cuerpo principal del código fuente del programa que existen en cualquier parte del código, incluso dentro de los métodos y funciones.

Pila

Antes de ejecutar un método, el programa añade todas las variables locales declaradas en ese método en la pila, en orden inverso en que han sido declaradas. Al llamar al método, las variables locales se quitan de la pila en orden inverso, la variable más reciente es la primera en ser eliminada.

TIPOS DE DATOS

int

Usado para declarar una variable integer.

float

Usado para declarar una variable en coma flotante.

boolean

Usado para declarar una variable booleana, la cual tiene sólo 2 posibles valores: true o false.

char

Usado para declarar una variable carácter.

String

Secuencia de caracteres.

void

Indica que ése método no devuelve un valor.

null

Ausencia de valor.

integer

Número entero.

Número en coma flotante

Número real.

enum

Colección de constantes.

ArrayList

Matriz de tamaño variable.

Iterator

Objeto que permite desplazarse por una colección.

Anexo II. Temario del Aplicativo

Tema 1: Desarrollar en Android

Qué es Android

P1 - Sistema Operativo Android

Android es un sistema operativo que se emplea en dispositivos móviles, por lo general con pantalla táctil, presente también en tabletas, televisores y automóviles.
Es el sistema operativo móvil más usado, más de mil millones de personas lo utilizan alrededor del mundo y 8 de cada 10 teléfonos inteligentes funcionan con su plataforma.

Está basado en Linux, por lo que es de código abierto y cualquiera puede editarlo. Esa es la razón por la que empresas como Samsung, LG o Sony ofrecen sus propias versiones personalizadas de Android, y así diferenciarse en un mundo donde más del 80% de los smartphones usan este sistema operativo.

P2 - Plataforma Android

Es una plataforma creada por Google. Aplicaciones como YouTube, Google Maps, Google Mail y Google Search son incluidas de serie con el sistema operativo, lo que le permitió a Google expandir mucho sus servicios.

Android tiene una tienda llamada Google Play, con más de 1 millón y medio de aplicaciones y una creciente oferta de libros, música, películas y revistas.

Historia

P1 - Historia

En el cuarto trimestre de 2010 comenzó la supremacía de Android como el sistema operativo preferido en teléfonos móviles. En aquella ocasión, logró que 33,3 millones de dispositivos utilizaran su software, desbancando a Symbian de Nokia.

La idea detrás de Android se remonta hasta octubre de 2003, cuando Andy Rubin, Rich Miner, Chris White y Nick Sears fundaron Android, Inc. Su idea era desarrollar un sistema

operativo para móviles que estuviera basado en Linux y que además fuera fácil de utilizar y programar.

P2 - Historia

Las empresas dominantes de la época eran Nokia y BlackBerry, por lo que cuando Google anunció en 2005 que compraba Android, nadie sabía qué planeaba Google en el sector móvil.

Andy Rubin pasó a ser jefe de la división móvil de Google, y cuando en junio de 2007 Apple lanzó el iPhone, todos esperaban que Google creara su propio teléfono para competir con Apple y los fabricantes de la época. Sin embargo, Google eligió otro camino.

P3 - Historia

En noviembre de 2007, la empresa anunció la creación de la Open Handset Alliance, un convenio de compañías tecnológicas que hoy tiene cerca de 80 miembros y que en su momento decidieron apoyar el proyecto Android. La primera versión del sistema operativo móvil fue anunciada ese mismo mes, y recibió el nombre de Apple Pie.

La primera empresa que lanzó un teléfono con Android fue HTC, con su modelo Dream, sacado a la venta en septiembre de 2008. Sólo dos años le tomó a Google superar a Symbian, Blackberry OS y iOS, mandando a Nokia y a Blackberry al borde de la desaparición, catapultando a Samsung al trono como el fabricante móvil más vendido del mundo.

P4 - Historia

Un dato curioso: el robot verde mascota de Android se llama Andy y fue diseñado por Irina Block en 2005, poco después de que Android Inc. fuera adquirido por Google.
IMAGEN ANDROID

Versiones

P1 - Versiones

13 versiones de Android han sido distribuidas hasta la fecha desde que apareció Apple Pie en 2007, serán 14 con la llegada de Nougat (Android 7.0) este año. Todas reciben el nombre de un postre en inglés y en orden alfabético.

P2 - Versiones

Apple Pie (Android 1.0)

Anunciado en 2007 y lanzada de forma comercial con el HTC Dream en septiembre de 2008.

Banana Bread (Android 1.1)

Lanzado en febrero de 2009.

Cupcake (Android 1.5)

Lanzado en abril de 2009.

Donut (Android 1.6)

Lanzado en septiembre de 2009, significó la llegada de equipos como el Sony Ericsson Xperia X10.

Eclair (Android 2.0)

Lanzado en octubre de 2009, fue el sistema operativo de teléfonos como el Motorola Droid.

Froyo (Android 2.2)

Lanzado en mayo de 2010, vino acompañado de equipos como el HTC Inspire, el LG Optimus M y el primer Samsung Galaxy S.

Gingerbread (Android 2.3)

Lanzado en diciembre de 2010, trajo smartphones como el LG Optimus One y el Samsung Google Nexus S.

P3 - Versiones

Honeycomb (Android 3.0)

Lanzado en febrero de 2011, fue un sistema exclusivo para tabletas y se estrenó en la Motorola Xoom.

Ice Cream Sandwich (Android 4.0)

Lanzado en octubre de 2011, fue uno de los cambios más importantes de Android, y fue compatible con tabletas y smartphones.

Jelly Bean (Android 4.1)
Anunciado en junio de 2012, y su primer dispositivo fue el Nexus 7, si bien después llegaron el Nexus 4 de LG y el Nexus 10 de Samsung.

KitKat (Android 4.4)
Lanzado en octubre de 2013, trajo consigo a smartphones como el Galaxy S5, el Motorola Moto X y el Zony Xperia Z2.

Lollipop (Android 5.0)
Lanzado en noviembre de 2014, introdujo Material Design, un nuevo diseño en las aplicaciones de Android.

Marshmallow (Android 6.0)
Lanzado en septiembre de 2015, continúa la propuesta de Android Lollipop.

Nougat (Android 7.0)
En proceso de lanzamiento.

Java para Android

P1

Hay muchos lenguajes para desarrollar aplicaciones de Android, pero Java es el lenguaje que Google anima a los desarrolladores a utilizar.

Android no usa Java "puro". Aunque la escritura y el desarrollo de una aplicación Android es familiar para los desarrolladores Java con experiencia, la familiaridad termina cuando se compila y ejecuta el código. La razón es la forma que Android maneja sus aplicaciones durante el proceso de compilación.

P2

El mayor atractivo de Java es su capacidad de "escribe una vez, ejecuta en todas partes". En vez de traducir el programa en código máquina directamente como la mayoría de lenguajes, el compilador de Java traduce el programa en una forma intermedia llamada Bytecode. Se

crean un conjunto de instrucciones similares al código máquina, pero que se ejecutan en una máquina virtual (VM) y no en una estructura específica.

El uso de la máquina virtual significa que mientras se pueda leer e interpretar las instrucciones en Bytecode, el programa funcionará en la plataforma acogida, lo que garantiza la compatibilidad entre plataformas. Esto explica por qué la mayoría de programas de Java piden que descargues el Java Runtime Environment (JRE), ya que es el valor predeterminado de máquina virtual para la mayoría de plataformas.

P3

Compilar una aplicación en Android se hace de forma parecida, pero cuando la aplicación se instala en un dispositivo, el Bytecode de la aplicación se convierte en código máquina que está optimizado para el dispositivo Android concreto, mejorando el rendimiento en tiempo de ejecución. A éste proceso se le conoce como compilación Ahead of Time (AoT) y se hace posible gracias a la máquina virtual Android Runtime (ART).

AoT sólo ocurre en Android KitKat (4.4) o versiones superiores, aunque proporciona compatibilidad para versiones anteriores, que se basaban en otra máquina virtual conocida como Dalvik. Como ART, Dalvik hizo cambios en el Bytecode de Java para convertirlo en una forma específica para optimizar las apps en dispositivos Android de bajo rendimiento. A diferencia de ART, Dalvik no compila el Bytecode a código máquina hasta el tiempo de ejecución denominada compilación Just in Time (JIT), proceso más cercano al utilizado por Java Virtual Environment (JVE) en un PC.

P4

Es la conversión del Bytecode de Java lo que hace que escribir en Android sea menos “puro”. Los cambios en el Bytecode limitan la portabilidad de la aplicación, negando así una de las grandes promesas del lenguaje Java “escribir una vez, ejecutar en todas partes”.

Otra forma en la que Android diverge de Java es la disponibilidad de bibliotecas estándar. Si Java es tan portátil es porque se basa en un conjunto estandarizado de bibliotecas que se pueden utilizar en varias plataformas, como las bibliotecas de red o de interfaz de usuario.

Android en cambio ofrece un subconjunto de lo que Java proporciona, y además lo proporciona única y exclusivamente para Android.

P5

Android adopta el paradigma de la programación orientada a objetos de Java, y está diseñado para adaptarse a los conceptos de encapsulación, herencia y polimorfismo. Se utiliza todo esto cuando se programa apps.

Todos los objetos en Android heredan de la clase Object de alguna forma, construyendo sobre sus funciones para dotar de un comportamiento especializado y de características propias al objeto. Si miramos algunos objetos disponibles en la API de Android, veremos cuál es la jerarquía de la que hereda cada objeto, y todas acaban heredando Object.

Recordatorio Java

P1

Implementación de clases

```
public class MyClass {
	public String mString;
	private int mInt;
	// More member variables

	// Constructor for class
	public MyClass() {
		mString = “Hi”;
		mInt = 10;
	}
	// More methods…
}
```

P2

Métodos y funciones

```
public void doIt() {
	// task
}
```

```
public int doIt() {
	return 0;
}

public int doIt(int a) {
	return a;
}

public int doIt(int a, int b) {
	return a + b;
}
```

P3

Crear y usar una instancia de objeto

```
SomeObject a = new SomeObject();
a.getMemberVariable();
a.doIt();
a.doIt(1);
a.doIt(2, 3);
```

P4

Declarar variables

```
double doubleVar = 1.0
doubleVar = 2.0

int intVar = 1;
String stringVar = “Hi”;

Boolean truth = true;
```

P5

Control de flujo

```
Boolean condition = true;
if (condition) {
} else {
}
```

```
while (condition) {
	// condition = false;
}

for (int i = 0; i < 5; i++) {
}

int val = 2;
switch (val) {
	case 1:
		break;
	case 2:
		break;
	default:
		break;
}
```

P6

Ejemplos de String

```
String personOne = "Ray";
String personTwo = "Brian";

String combinedString = personOne + ": Hello, " + personTwo + "!";
String tipString = "2499.00";
Int tipInt = Integer.parseInt(tipString);
double tipDouble = Double.parseDouble(tipString);
```

P7

Ejemplos de array

```
String personOne = "Ray";
String personTwo = "Brian";

String[] arrayPerson = {personOne, personTwo};

for (String person : arrayPerson) {
	Log.d("person: ", person);
}
```

```
String arrayZero = array[0];
```

Tema 2: Conceptos básicos

Activity [FALTA]

Ciclo de vida de una Activity [FALTA]

Model

P1

Los modelos te permiten crear objetos que consisten en estructuras de datos y funcionalidades que son fundamentales para escribir un programa en Android. Se encuentran en clases separadas de las clases de interfaz de usuario. Ésta separación no sólo ayuda a mantener la aplicación organizada, sino que se ajusta a la idea de encapsulación en Programación Orientada a Objetos.

En los modelos no se menciona una vista o actividad, sólo hay estructuras de datos, tipos de datos en bruto y diversas funciones. De hecho, sólo es un Plain Old Java Object (POJO), un objeto plano de Java.

P2

Un modelo típico es algo así:

```
public class ReminderList {

  private ArrayList mReminder;

  public ReminderList() {
    mReminder = new ArrayList<>();
  }

  public void setReminder(ArrayList reminder) {
    this.mReminder = reminder;
  }

  public ArrayList getReminder () {
    return mReminder;
```

```
  }

  public void removeLastItemFromList() {
    if (!mReminder.isEmpty()) {
        mReminder.remove(mReminder.size() - 1);
    }
  }
}
```

P3

En Android es convencional poner el prefijo “m-“ en el nombre de una variable global para indicar que además es no pública y no estática. Es un buen hábito empezar a nombrarlas convencionalmente, ya que así está hecho en el código fuente de Android.

Los getters y setters de variables miembro son un elemento básico de desarrollo Android, ya que proporcionan al resto de la aplicación una manera rápida de obtener o modificar una variable miembro.

Modifiers

P1

Algo que ayuda al trabajo de los getters y setters son los modificadores de acceso. Volvamos a echar un vistazo en el siguiente fragmento de código del modelo anterior, prestando atención a las palabras “private” y “public”:

```
private ArrayList mReminder;

public void setReminder (ArrayList reminder) {
  this.mReminder = reminder;
}
```

P2

“Private” y “public” son modificadores de acceso y son responsables de especificar qué elementos del modelo son accesibles a otras clases, lo que ayuda a encapsular sus objetos.

Public

Accesible por todos los demás objetos fuera de la clase. Los métodos y variables públicos forman la API de la clase.

Private

Sólo se puede acceder desde el mismo objeto, ni siquiera sus subclases pueden acceder a él.

Protected

Accesible desde el mismo objeto y sus subclases, aunque no está disponible para ninguna otra clase exterior.

P3

Las variables miembro siempre deben establecerse como private o protected. El acceso a los objetos private o protected desde fuera de la clase debe ser a través de getters y setters públicos.

Si hacemos una subclase del modelo anterior y queremos que la subclase acceda a mReminder, hay que cambiar su modificador de acceso a protected, que permitirá el acceso de esa variable a la subclase, además de personalizar aún más o trabajar con ésa variable:

```
protected ArrayList mReminder;
```

Fragment [FALTA]

Las actividades son excelentes para la gestión de todo el contenido de la pantalla, pero no les gusta compartir. Por suerte, hay una gran manera de romper la interfaz de usuario en fragmentos más pequeños llamados Fragments.

Los fragmentos son como las actividades, pero con la ventaja añadida de ser capaces de incrustarse en actividades como si fueran vistas. Tienen métodos de ciclo de vida similares al onCreate() de una actividad y pueden recibir input igual que una actividad.

Un layout para un fragmento se ve exactamente igual de diseño que el de una actividad, conteniendo algunas declaraciones de las vistas. Incluso se enlazan de la misma forma en el

código, declarando una vista como una variable y buscando la vista a través de un identificador que has proporcionado en el layout.

```
public class EmbeddedFragment extends Fragment {

  @Override
  public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {
    // Inflate the layout for this fragment
    return inflater.inflate(R.layout.fragment_embedded, container, false);
  }
}
```

Ciclo de vida de un Fragment [FALTA]

Interface

P1

Imaginemos una situación donde una actividad aloja a 3 fragmentos, cada uno haciendo sus propias operaciones, pero con la necesidad de informar a otros fragmentos lo que sucede en momentos específicos.

En teoría, los fragmentos sólo deben preocuparse por su propia finalidad y no tienen por qué saber que existen junto a otros, ya que la actividad es la que manda y la única que tiene acceso a todos los fragmentos y sabe lo que hacen cada uno.

Esta situación exige una característica conocida como Java Interfaces.

P2

Una interfaz es como una clase, pero sin la implementación. En su lugar define la API pública. Las clases pueden implementar estas interfaces y su código ya no se basa en las implementaciones de una clase concreta.

Las interfaces permiten a los objetos trabajar indirectamente con otros objetos sin exponer su funcionamiento interno.

P3

En Android, las interfaces son útiles para facilitar al fragmento la comunicación con la actividad o de fragmento a fragmento. Funciona así:

```
public class EmbeddedFragment extends Fragment {
  // 1
  OnItemInListSelectedListener mCallback;

  // 2
   // Container Activity must implement this interface
  public interface OnItemInListSelectedListener {
    public void onItemInListSelected(int position);
  }

// This makes sure that the container activity has implemented the callback
interface. If not, it throws an exception
  @Override
  public void onAttach(Activity activity) {
    super.onAttach(activity);
    // 3
    try {
      mCallback = (OnItemInListSelectedListener) activity;
    }
    catch (ClassCastException e) {
      throw new ClassCastException(activity.toString() + " must implement
OnItemInListSelectedListener");
    }
  }

  @Override
  public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {
    return inflater.inflate(R.layout.fragment_embedded, container, false);
  }
}
```

P4

1. En el fragmento se declara una variable miembro para almacenar un objeto personalizado que implementa OnItemInListSelectedListener y lo llamas mCallback.

2. Crear la interfaz y declarar los métodos necesarios (la interfaz puede tener los métodos que quieras).

3. En onAttach(), un método de ciclo de vida del fragmento, se comprueba si la actividad a la que el fragmento está asociado implementa la interfaz OnItemInListSelectedListener. Si no es así, entonces no sirve para ése propósito y tenemos un problema. El ClassCastException describe el problema en tiempo de ejecución. Lo mejor es siempre poner la Exception, para darte cuenta si falla.

P5

El siguiente paso es para hacer que la actividad utilice la interfaz:

```
public class MainMenuActivity extends Activity implements
EmbeddedFragment.OnItemInListSelectedListener {
  ...
  public void onItemInListSelected(int position) {
      // Received a message from the Fragment, you'll do something neat here
  }
}
```

La palabra clave “implements” en la definición de la clase indica que esta clase implementará la interfaz especificada. Se tendrá que proporcionar implementaciones para todos los métodos de la interfaz. En nuestro ejemplo sólo hay un método (onItemInListSelected).

P6

Ya hemos hecho la parte difícil, pero aún queda establecer la comunicación fragmento a fragmento. Definimos un segundo fragmento, ListDetailFragment, con identificador fragment_list_detail y un método showListItemDetails(int). Para establecer la comunicación fragmento a fragmento, se hace a través de la actividad asociada, ya que dos fragmentos nunca deberían comunicarse directamente:

```
public void onItemInListSelected(int position) {
  ListDetailFragment listDetailFragment = (ListDetailFragment)
getFragmentManager.findFragmentById(R.id.fragment_list_detail);
  listDetailFragment.showListItemDetails(position);
}
```

Annotations

P1

Las palabras que comienzan por una @ son anotaciones. Proporcionan información adicional en las diferentes etapas de una aplicación. Le dan información al compilador en tiempo de ejecución o incluso generan código extra:

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

La anotación más común en Android es @Override, que existe para informar al compilador que ese método específico debe sobrescribir al de la super clase.

P2

Otra anotación conocida es @TargetApi, que existe para indicar a los métodos que son para uso de una versión específica de Android. Usar un método con una anotación @TargetApi mayor a la versión final del dispositivo donde se quiere usar la app, causará que el compilador se queje de que estás usando funcionalidades que no están disponibles en la versión del dispositivo. Al ser sólo un warning, todavía se puede ejecutar la aplicación, pero casi seguro acabará causando un error a largo plazo.

P3

Cuando estamos construyendo una app para Android necesitaremos otras anotaciones. Si vamos a la documentación de la API de Android encontraremos más anotaciones e incluso se pueden crear anotaciones propias para automatizar tareas específicas, generar código, etc.

Si visitamos la biblioteca AndroidAnnotations encontraremos una gran variedad de anotaciones personalizadas que simplifican funciones multi-línea en una sola línea con una anotación y otras características útiles: http://androidannotations.org/

Próximo Temario:

Tema 3: Primeros pasos

- **Instalando Android Studio**
- **Android SDK Manager**
- **Creando nuestro primer proyecto**
- **Estructura del proyecto y archivos**
- **Run en el emulador**
- **Run en un dispositivo**
- **Importar un proyecto existente**

Tema 4: Conceptos clave

Tema 5: Interfaz de usuario

Anexo III. Contenido del CD-ROM.

- Documentación del proyecto en Word (Memoria, Estudio Económico, Anexos)
- Documentación del proyecto en PDF (Memoria, Estudio Económico, Anexos)
- Proyecto Java de AndroidUp
- APK de AndroidUp
- Proyecto API REST de AndroidUp