iMosso

**2018**

David

[Escriba el nombre de la compañía]

05/10/2018



Introducción

**¿Cuál es la idea?**

Ayudar a la gente que quiera opositar para el cuerpo de policía de los Mossos d’esquadra.

**¿Por qué?**

Hace años oposité y aunque no salieron plazas me hubiera gustado tener algo así.

**¿Qué voy a hacer?**

Una aplicación para Android.

Estudio mercado:

**Análisis DAFO:**

**Análisis de aplicaciones con funciones similares a la nuestra en el mercado.**

**¿Tengo competencia?**

Sí. Solamente una app que existe en el Store de Android es iGub 2.0 la cual tiene un módulo de Mossos con un coste de 20 € anuales.

**¿Público?**

Toda la gente que cumpla las bases de los mossos d’esquadra o que en el futuro quiera opositar y quiera empezar a prepararse la oposición.

**¿Tengo competencia?**

Sí, iGub.

Precio de solo del módulo de Mossos 20 € anuales.

**Diferencia entre competencia y lo mío**

iMosso es una app totalmente gratuita que se mantiene gracias a Adsense.

**Diseño**

El diseño será a partir de Vistas.

Cronograma

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **CRONOLOGÍA DE PROYECTO FIN DE CURSO** | | | | | | | | | | | | | | | | | |  |  |  |
|  |  |  |
| **ACTIVIDADES / SEMANA ASIGNADA** | | **SEPTIEMBRE** | | | | **OCTUBRE** | | | | **NOVIEMBRE** | | | | **DICIEMBRE** | | | |  |  |  |
| 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 | 1 | 2 | 3 | 4 |  |  |  |
| 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 6 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 7 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 8 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 9 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 10 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 11 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 12 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 13 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 14 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 15 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 16 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 17 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 18 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 19 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 20 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

Requisitos

Funcionales y

no funcionales

**Requisitos funcionales**

|  |  |
| --- | --- |
| **Acceder** | **Pulsar botón de Acceder** |
|  |  |

**Requisitos no funcionales:**

|  |  |
| --- | --- |
| **Requerimientos del producto:** | Android : 5.0 Lolipop (o superior).  Nivel de api: 21 (o superior).  Espacio en disco: PENDIENTE |
| **Requerimientos organizacionales:** | Entrega a finales Enero 2019 |
| **Requerimientos externos:** | Tener internet para descargar la guía de Mossos. |
| **Requisitos de eficiencia:** | Trabajar en local siempre que sea posible. |
| **Legales:** | Cumplir las leyes vigentes que puedan implicar problemas legales en la actualidad. |
| **Horas para el proyecto:** | 400 Horas |
| **Económicos:** | Los mínimos imprescindibles para que la app pueda generar dinero más fácilmente. Por ello, trabajaremos con todo de forma local. |

**Diseño de BBDD**

Diseño

Sí. Solamente una app que existe en el Store de Android es

**O en git hub realizando un gitLog**

Y ahora nos vamos a la vista de manera que nos quede algo así:

1) Intro

2) Cronograma

2) Diseño

diagramas caso de uso

diagrama de BBDD

3) Código fuente

4) Manual usuario

5) Powerpoint

Intro:

¿Qué voy a hacer?

¿Cuál es la idea?

¿Por qué?

Estudio mercado:

¿Público?

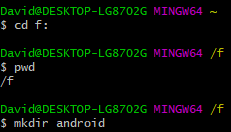
Diferencia entre competencia y lo mío

Diseño

**Inicializando el control de versiones Git para el proyecto**

Utilizaremos Git para controlar de una manera más inteligente nuestro proyecto. Git nos permitirá realizar con mayor eficiencia una “copia” de varios puntos de la aplicación y nos permitirá retroceder a un punto determinado para ver el estado de la aplicación en aquel entonces o avanzar hasta ver el resultado final.

Para inicializar git vamos a la ruta donde vamos a crear el proyecto en este caso en mi unidad f: y creamos un directorio llamado android:



Una vez creado, nos situamos en dicho directorio:

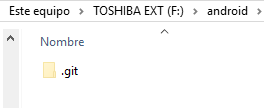


E inicializamos el git para poder trabajar con él:



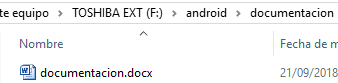
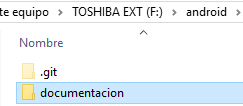
**Revisando el directorio**

Si finalmente ahora abrimos el directorio que hemos creado anteriormente, podemos ver que tras realizar el git init. Git nos ha creado automáticamente un subdirectorio que contiene todos los archivos necesarios del repositorio (el esqueleto de un repositorio Git).

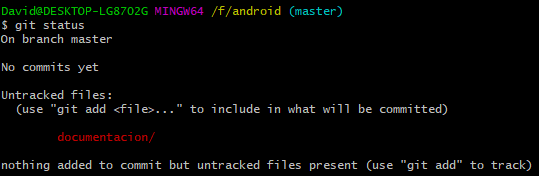


**Añadiendo contenido a nuestro proyecto**

Bien lo siguiente será crear una carpeta documentación y añadir este subdirectorio el documento de Word para que esté dentro del repositorio. Para ello, movemos el archivo al directorio f:/android/documentación e introducimos el documento de word:



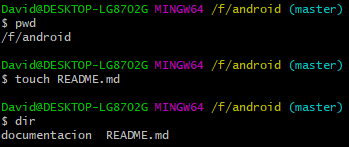
Si ahora visualizamos el directorio desde la consola de comandos de git hacemos git status podemos comprobar que aún no hemos realizado ningún commit. Un commit significaría realizar una copia “una foto” del estado de nuestro proyecto para poder retroceder en caso que lo deseemos más tarde.



**Creado el readme.md**

Vamos a ver cómo realizar un readme.md y para que nos va a servir.

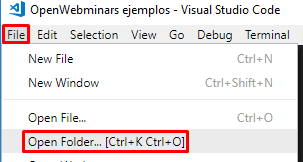
Un readme es el archivo que va a contener la información principal del proyecto. En español y durante la época de los 90, era más característico realizar un leeme.txt está representa su evolución. Si os fijáis tiene la extensión .md que significa markdown. Y normalmente suele escribirse con mayúsculas README.md. En git hub, donde iremos subiendo las diferentes versiones de nuestro proyecto git si un repositorio tiene un archivo README.md este será convertido en html y presentado como página principal dado esto su importancia.



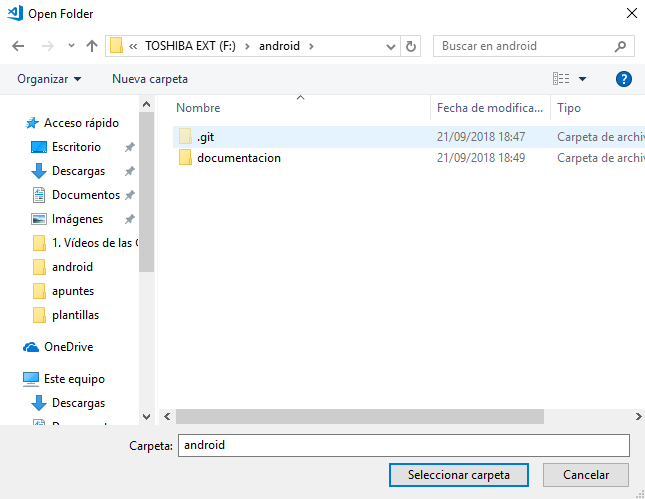
Si hacemos un touch README.md ya hemos creado el archivo y si ahora hacemos un dir podemos ver que tenemos el directorio que hemos creado anteriormente y el archivo README.md.

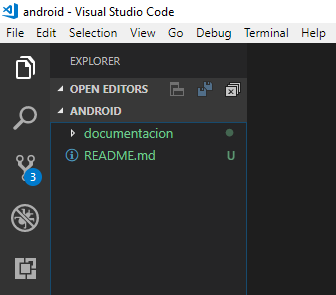
Bien para trabajar con nuestro archivo README.md vamos a utilizar Visual Studio Code. Uno de los beneficios que nos ofrece dicho IDE es que nos permitirá entender mejor que es el README.md y gestionar mejor los commits de git (las fotos con el estado actual).

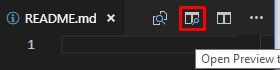
Para ello, vamos a File > Open Folder…



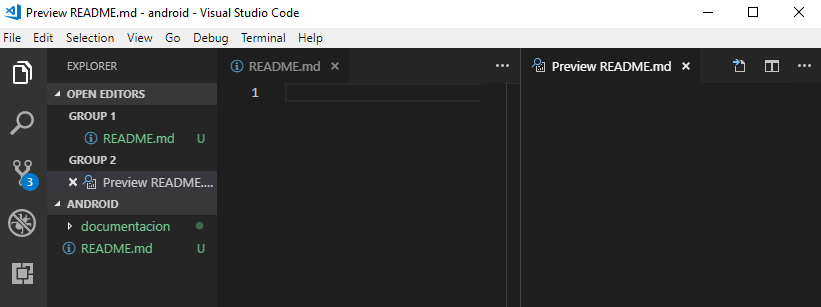
Y seleccionamos el directorio de nuestro proyecto.

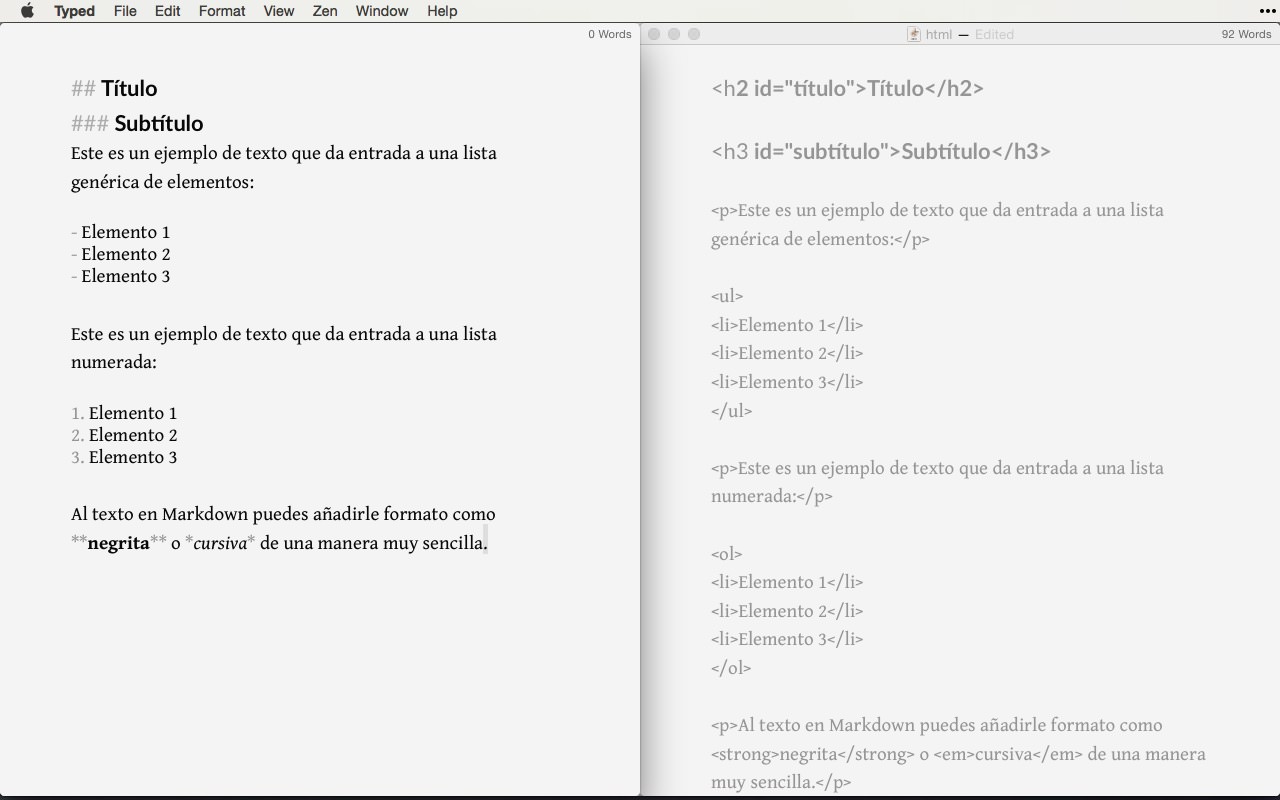


Si abrimos el fichero README.md y le damos a open preview:



Tenemos la división del readme tanto para verlo en formato markdown como en formato que se visualizará en Git:



Un ejemplo para que nos quedé claro del todo lo que es markdown es un html simplificado el ejemplo de la derecha es en markdown y el de la izquierda el equivalente en html.

Bien pues vamos a empezar a redactar el contenido básico de nuestro README.md

**# iMosso**

**## 1.Información del proyecto**

- Dispositivos compatibles: Android

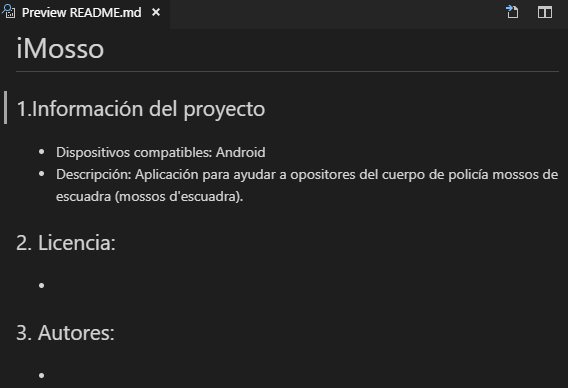
- Descripción: Aplicación para ayudar a opositores del cuerpo de policía mossos de escuadra (mossos d'escuadra).

**## 2. Licencia:**

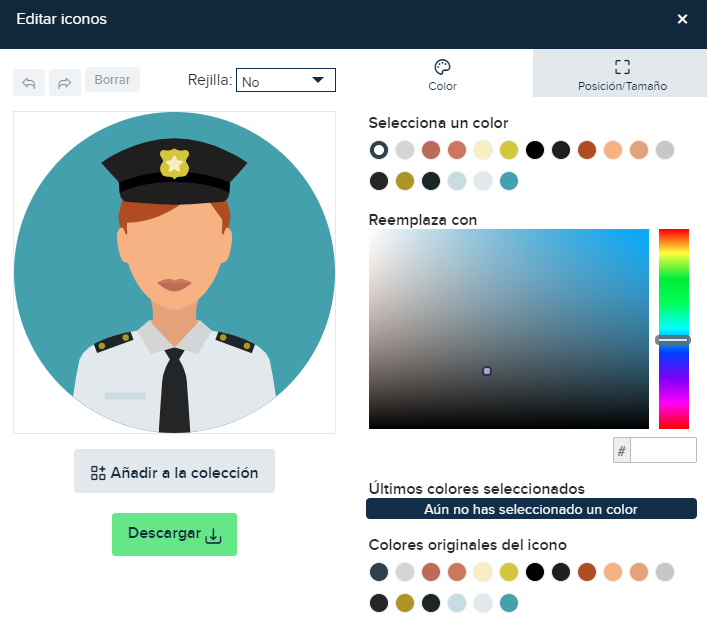
-

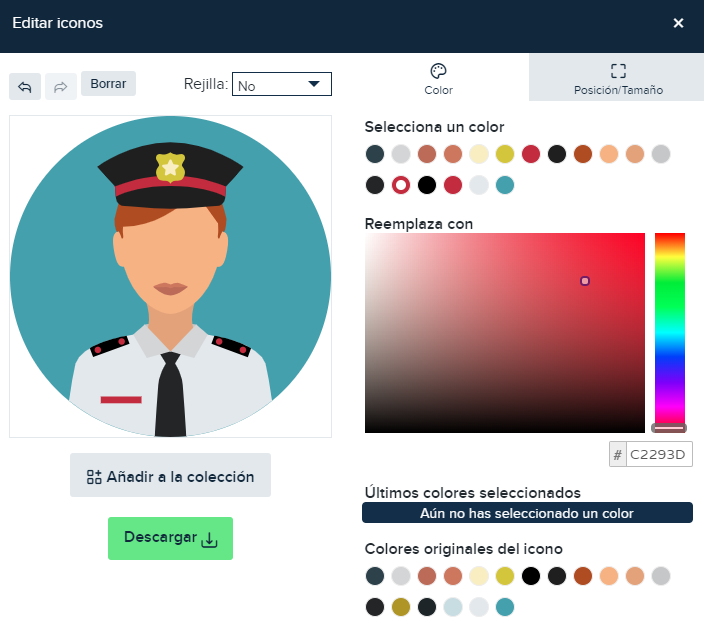
**## 3. Autores:**

-

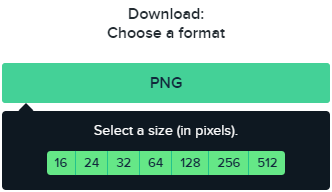
Bien pues una vez escrito si visualizamos el contenido:

Para realizar el icono de la app nos hemos metido en <https://www.flaticon.es/> y basándome meticulosamente en los colores de los uniformes del cuerpo de policía del que vamos a hacer la oposición y con herramientas para sacar el color, hemos buscado el siguiente:

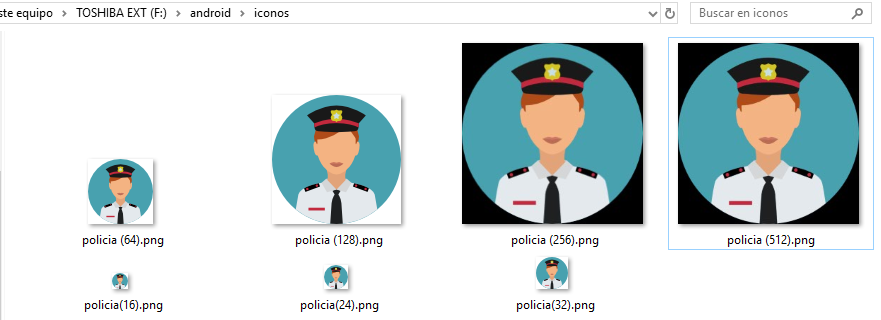




Y una vez tenemos la imagen la vamos a descargar y guardar en todos los formatos



Y vamos a crear una carpeta en el proyecto y vamos a poner los iconos con las diferentes resoluciones.



Y ahora que ya tenemos el icono vamos a añadirle dicho icono a la documentación para mejorar la estructura del README.md mediante a ![](iconos/policia(256).png)

**# iMosso**

![](iconos/policia(256).png)

**## 1.Información del proyecto**

- Dispositivos compatibles: Android

- Descripción: Aplicación para ayudar a opositores del cuerpo de policía mossos de escuadra (mossos d'escuadra).

**## 2. Licencia:**

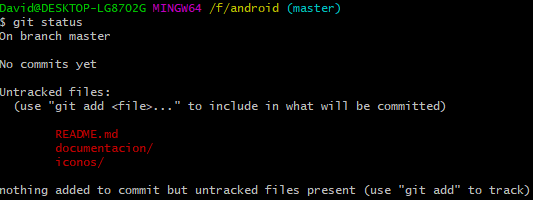
-

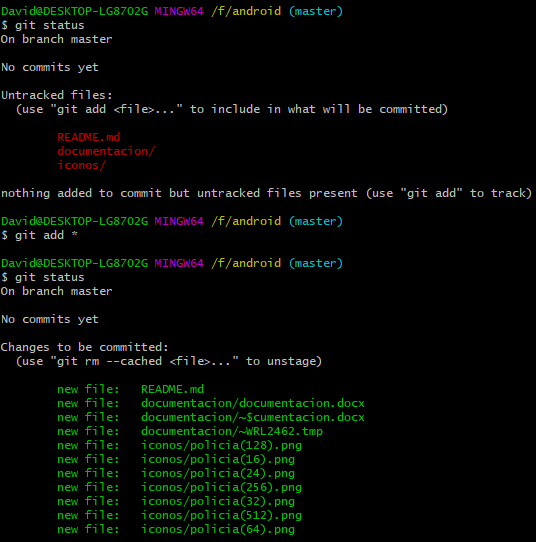
**## 3. Autores:**

-

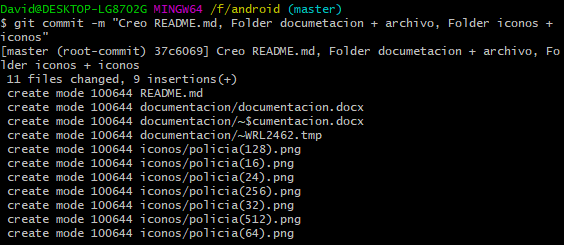
El resultado como podéis comprobar hace que el archivo tenga una presentación mucho más elegante.

Bien llega la hora de realizar el primer commit para guardar el estado de todo el trabajo que hemos realizado hasta ahora.

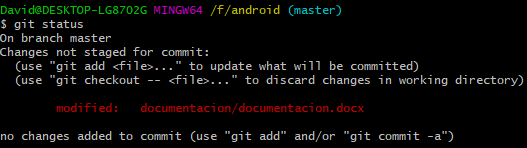




También podríamos hacer un git add . que es equivalente a git add \*

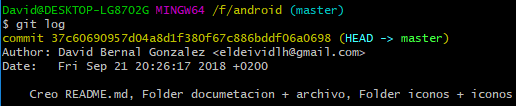


Si ahora compruebo el git status:



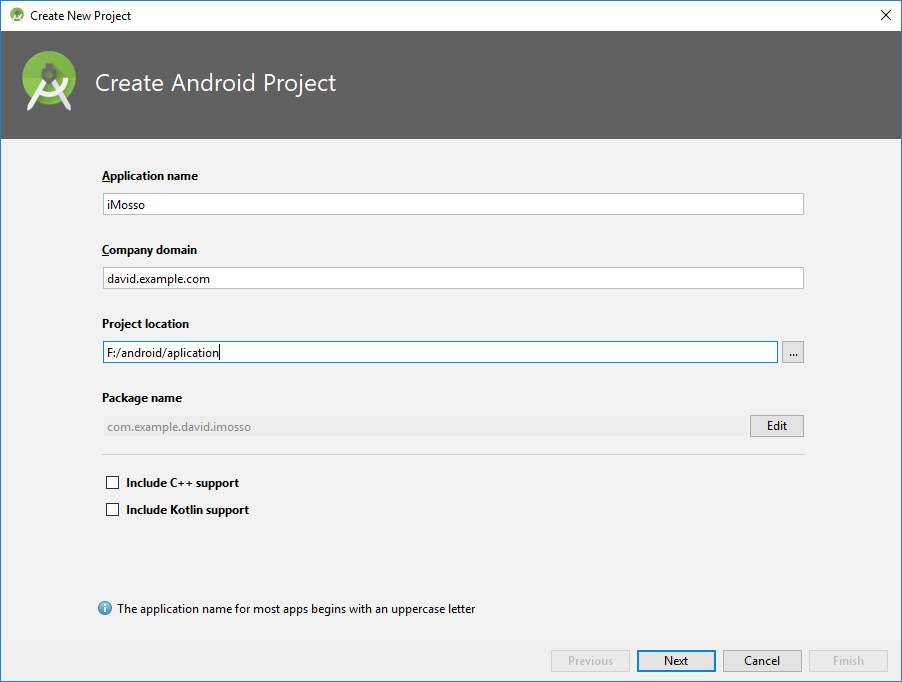
Puedo comprobar que ya han desaparecido todos los archivos a excepción de la documentación que estoy editando conforme realizo el proyecto.

Y aquí puedo ver que he hecho un commit con todo para poder volver a dicho punto cuando quiera.

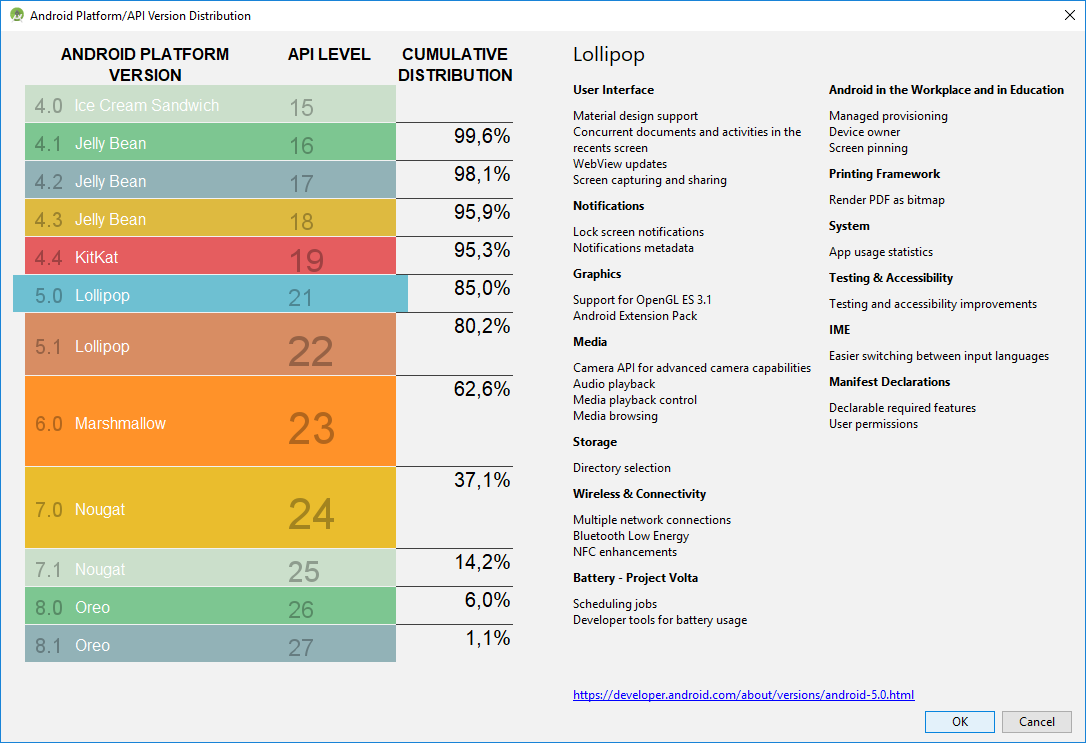


**Creando el proyecto de Android**

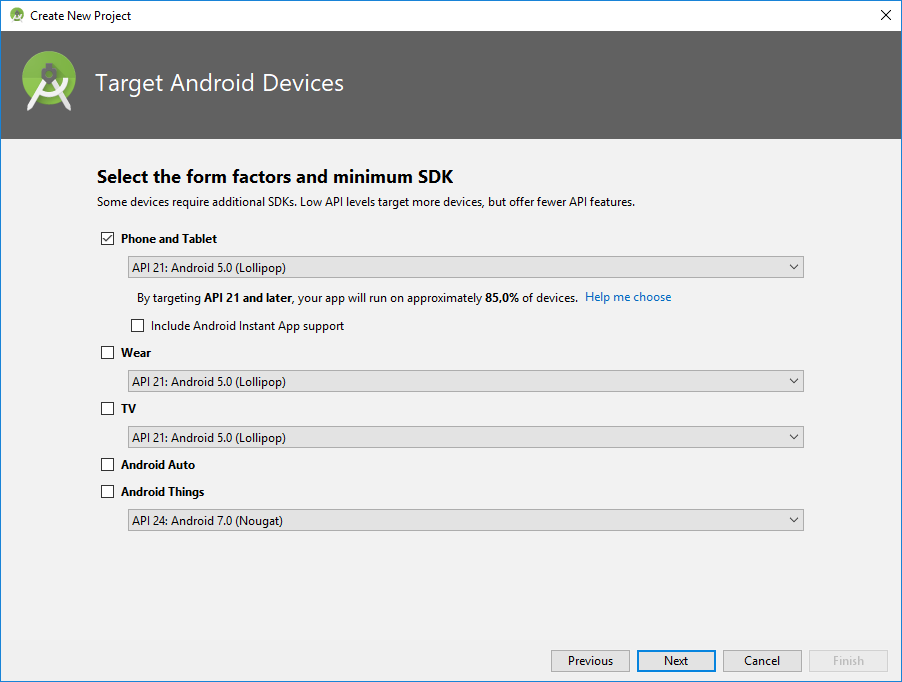
Llega la hora de ponernos ya con Android studio, lo primero que vamos a hacer es crear el proyecto con File > New > Proyect e indicar la ruta de nuestro proyecto git y concretamente se guardará en el subdirectorio aplication.

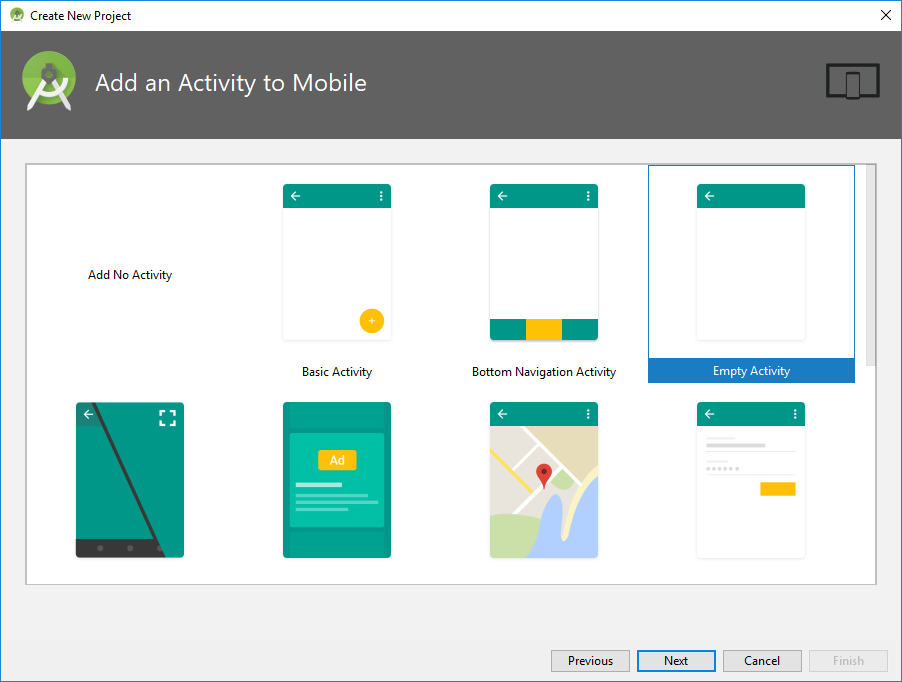


**Seleccionando una versión Android para el proyecto**

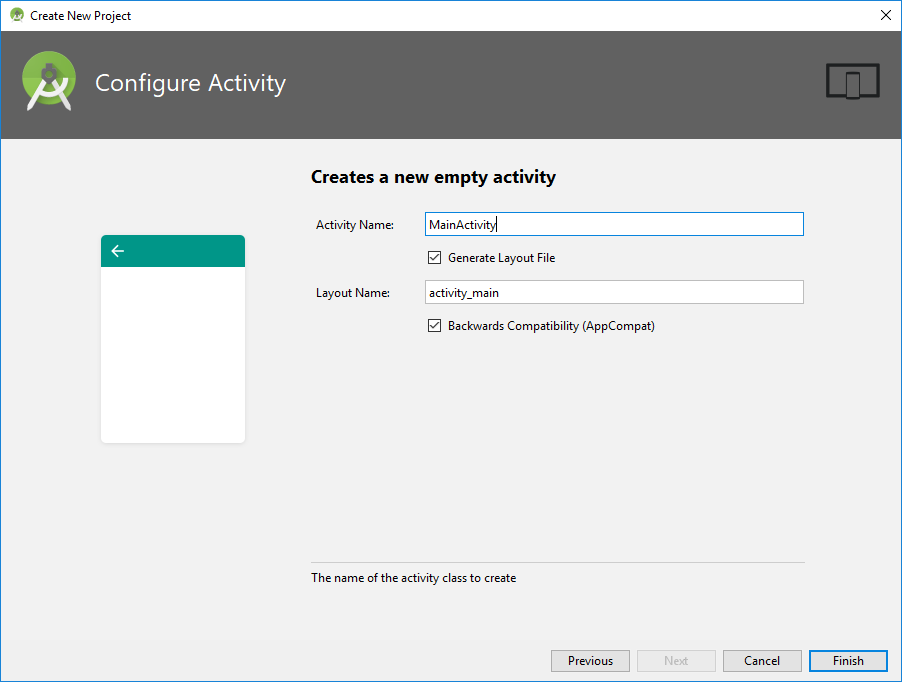
Bien, llega la hora uno de los momentos más difíciles, el momento de elegir una versión de android. La versión afortunada en este caso es la 5.0 Lolipop. Con un 85 % de compatibilidad sobre todos los dispositivos Android, deja muy pocos dispositivos fuera sobre todo pensando que el público objetivo de la aplicación es de entre 18 años y 35 años aproximadamente. Ya que han quitado desde hace poco el máximo de edad pero ese es nuestro público objetivo.

Por lo tanto, esos usuarios, suelen tener dispositivos que normalmente no superan los 3 – 4 años de media y por tanto prefiero trabajar con una api algo más avanzada para que la aplicación sea lo más estable posible. Por tanto, y a modo de resumen api nivel 21 con una compatibilidad de un 85 % pero debido a que gran parte de nuestros usuarios son millennials, creemos que la aproximación será de un 90 – 92 % aproximadamente.





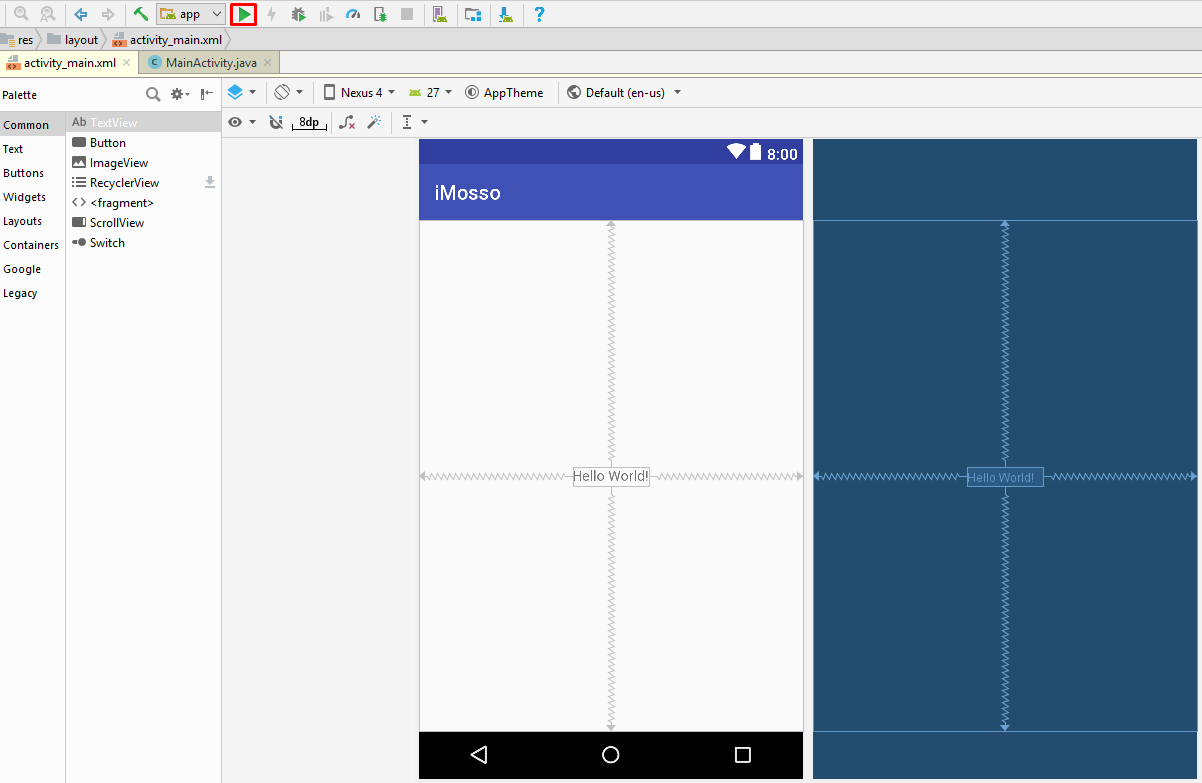
Y finalmente, creamos la actividad principal con el nombre de MainActivity.

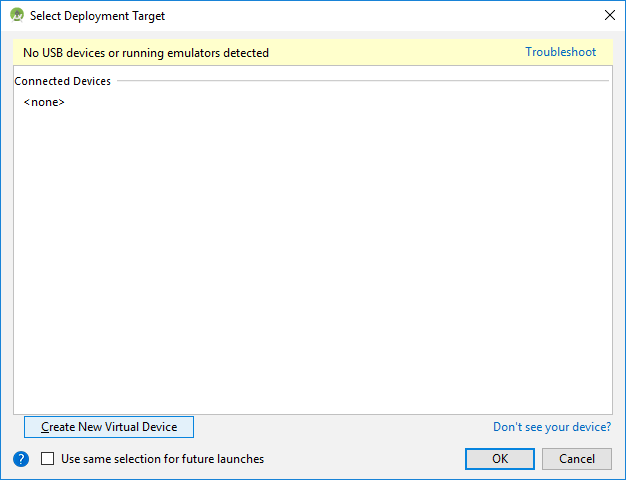


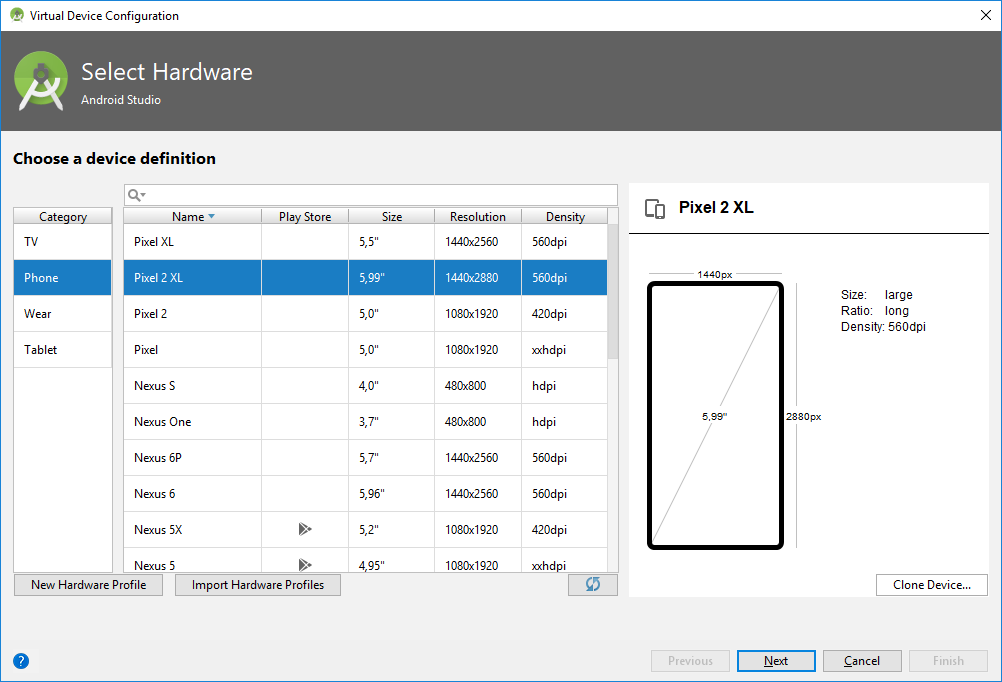
Al finalizar la creación del proyecto, nos aparece un mensaje de si queremos que VCS Version Control System no Visual Studio Code pero en este caso preferimos pulsar sobre ignorar y gestionar todo desde git por línea de comandos.

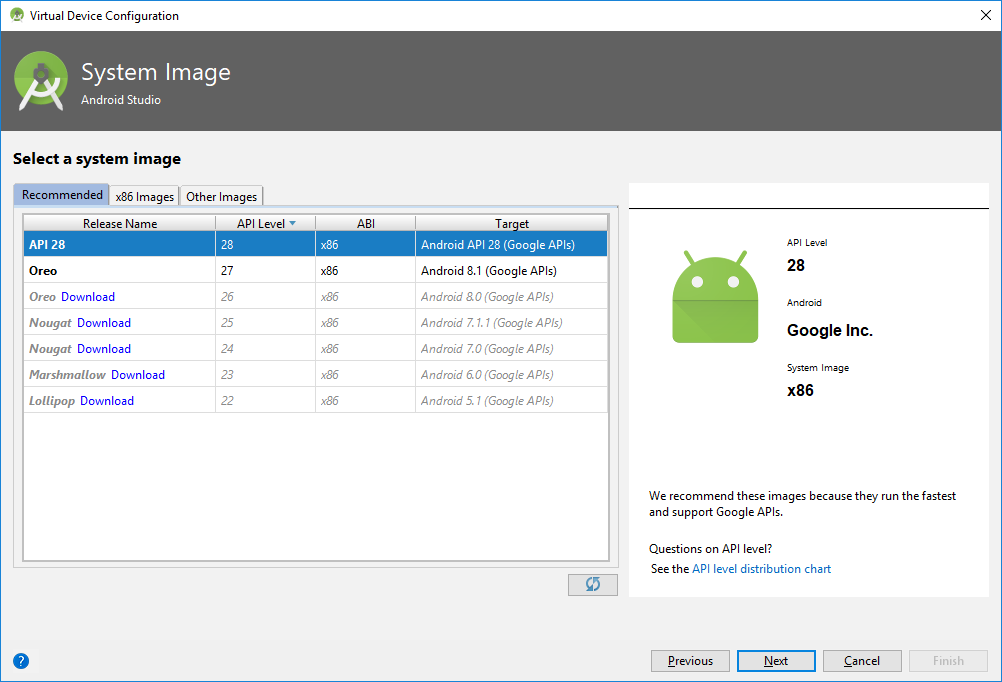


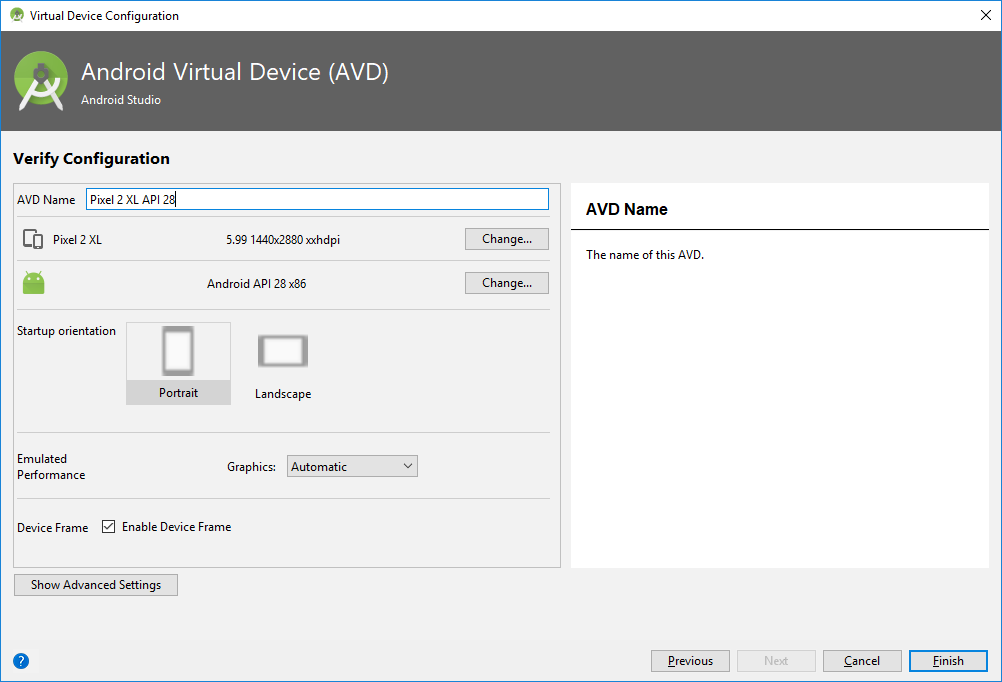
**Creando la máquina virtual y la primera vista previa del nuestra aplicación**

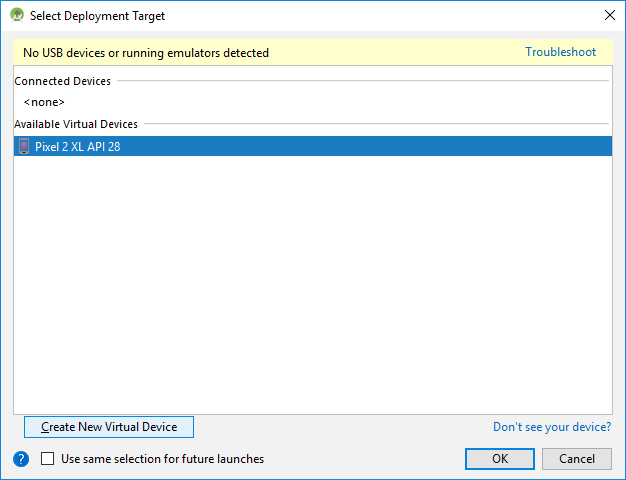


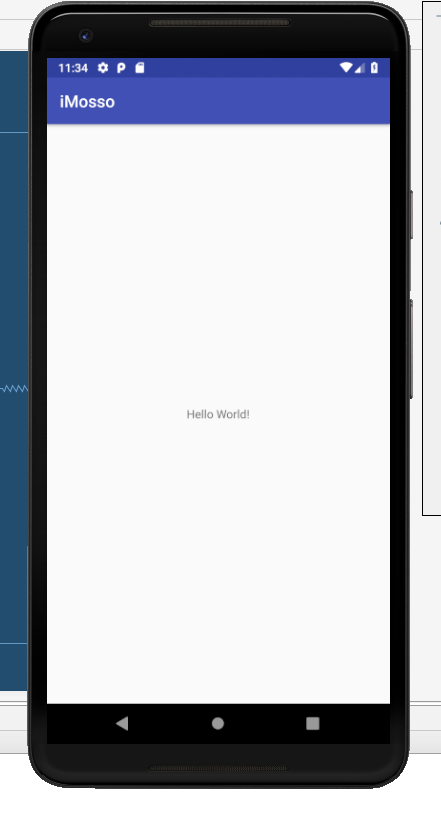












**Añadiendo información al README.md**

Aprovechamos para añadir toda la información que vamos conociendo al README.md

**# iMosso**

![](iconos/policia(256).png)

**## 1.Información del proyecto**

- Dispositivos compatibles: Android

- Compatibilidad:

- Versión mínima de Android : 5.0 Lolipop

- Nivel de api mínimo: api 21

- Compatibilidad mínima: 85 % (90-92 % estamación real).

- Descripción: Aplicación para ayudar a opositores del cuerpo de policía mossos de escuadra (mossos d'escuadra).

**## 2. Licencia:**

- Software Comercial

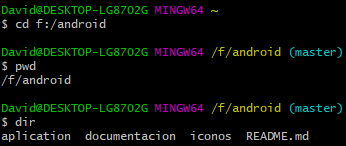
**## 3. Autor:**

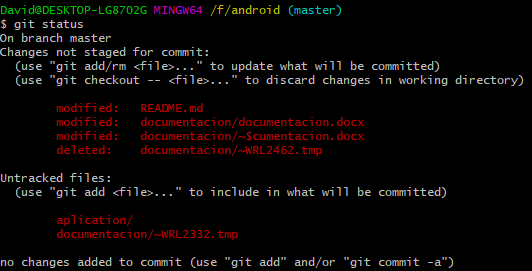
- David Bernal Gonzalez

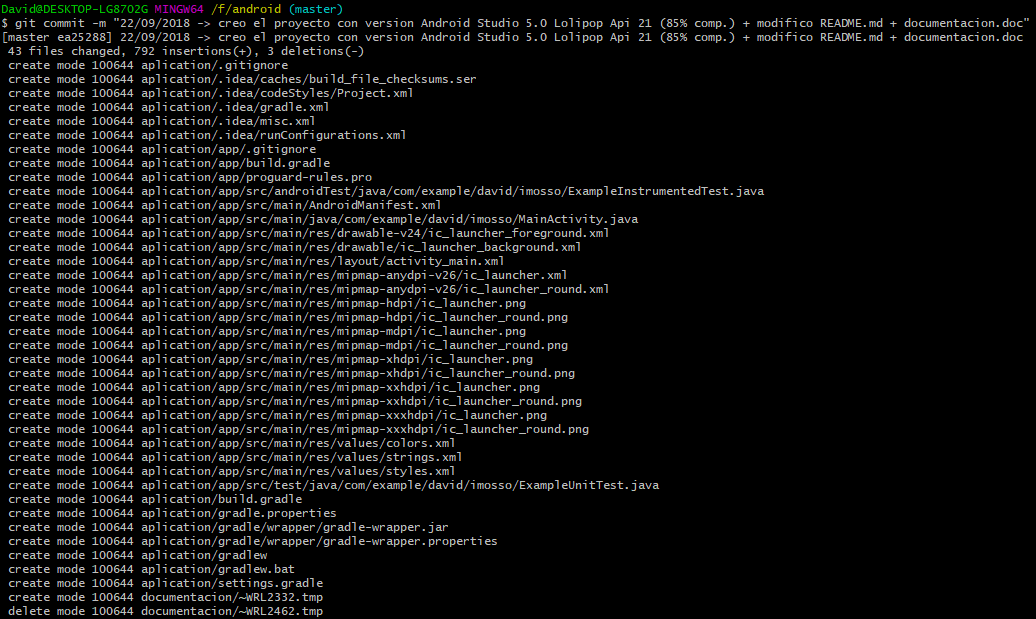


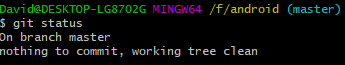
**Realizando un commit de nuevo**

Bien una vez tenemos el proyecto Android creado, considero que no nos encontramos en un buen momento para crear una copia de Git.

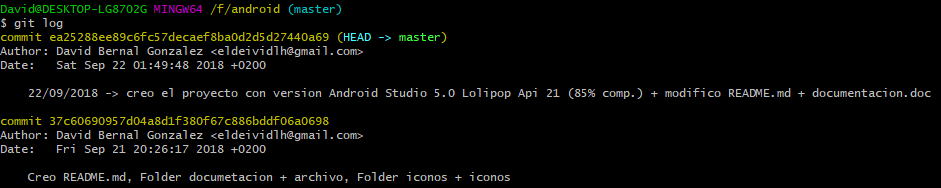




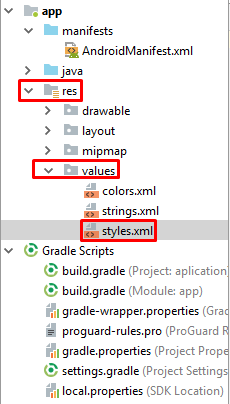


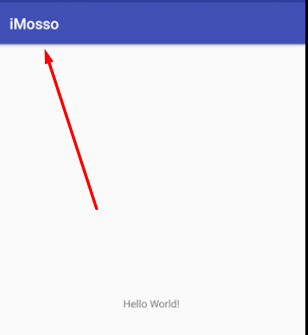


Y ya podemos ver que tenemos dos puntos guardados en el git:



**Eliminando la barra superior iMosso:**

Lo primero que vamos a hacer en el proyecto es eliminar la barra superior iMosso de color azul para disponer de más espacio en nuestra aplicación y a su vez porque considero que es algo antiestético.



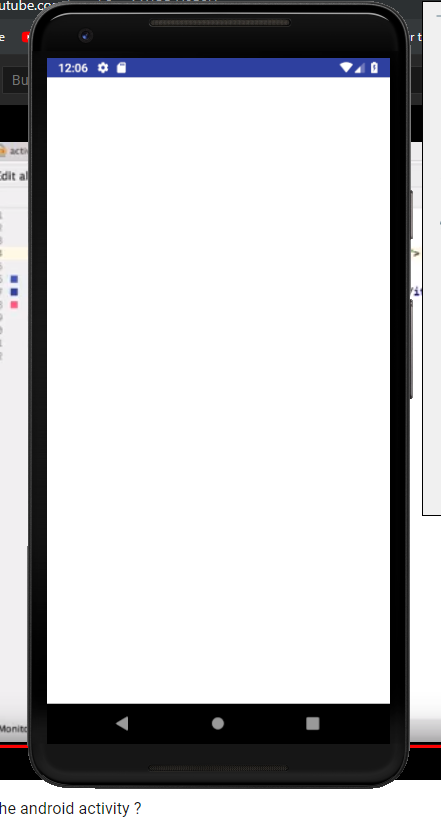
Para ello, vamos a:

Y tenemos el siguiente archivo XML:

<**resources**>  
  
 *<!-- Base application theme. -->* <**style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar"**>  
 *<!-- Customize your theme here. -->* <**item name="colorPrimary"**>@color/colorPrimary</**item**>  
 <**item name="colorPrimaryDark"**>@color/colorPrimaryDark</**item**>  
 <**item name="colorAccent"**>@color/colorAccent</**item**>  
 </**style**>  
  
</**resources**>

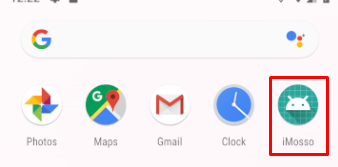
Eliminamos del parent del style **Light.DarkActionBar** y lo substituimos por **NoActionBar** de tal manera que nos quede de la siguiente manera:

<**resources**>  
  
 *<!-- Base application theme. -->* <**style name="AppTheme" parent="Theme.AppCompat.NoActionBar"**>  
 *<!-- Customize your theme here. -->* <**item name="colorPrimary"**>@color/colorPrimary</**item**>  
 <**item name="colorPrimaryDark"**>@color/colorPrimaryDark</**item**>  
 <**item name="colorAccent"**>@color/colorAccent</**item**>  
 </**style**>  
  
</**resources**>

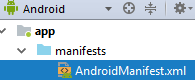


**Cambiando el icono de la aplicación:**

Si nos metemos en la aplicación podemos ver que el icono de iMosso es horroroso. Si queremos crearnos nuestra propia marca, debemos definir un icono que nos diferencie del resto y no podemos utilizar el que viene por defecto en cualquier aplicación de android.



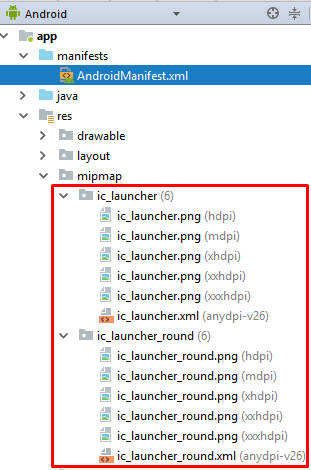
Por ello, vamos a utilizar el icono que habíamos creado anteriormente. Bien para cambiar el icono, lo primero que tenemos que hacer es abrir el fichero de android manifest.



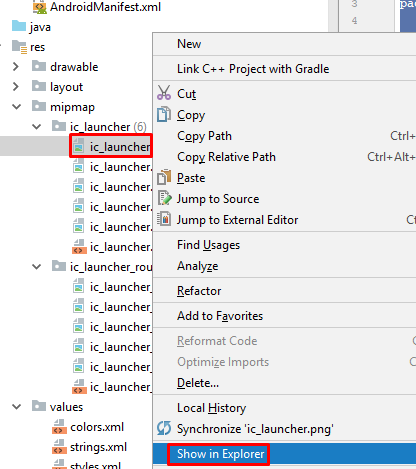
*<?***xml version="1.0" encoding="utf-8"***?>*<**manifest xmlns:android="http://schemas.android.com/apk/res/android"  
 package="com.example.david.imosso"**>  
  
 <**application  
 android:allowBackup="true"  
 android:icon="@mipmap/ic\_launcher"  
 android:label="@string/app\_name"  
 android:roundIcon="@mipmap/ic\_launcher\_round"  
 android:supportsRtl="true"  
 android:theme="@style/AppTheme"**>  
 <**activity android:name=".MainActivity"**>  
 <**intent-filter**>  
 <**action android:name="android.intent.action.MAIN"** />  
  
 <**category android:name="android.intent.category.LAUNCHER"** />  
 </**intent-filter**>  
 </**activity**>  
 </**application**>  
  
</**manifest**>

En android manifest, podemos comprobar que tenemos dos iconos y cambiaremos ambos.



Para ello, vamos a:

Donde si os fijáis tenemos varias resoluciones del icono. Por lo que pondremos



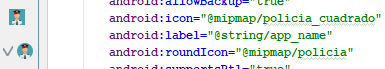
Lo teníamos de está manera:

*<?***xml version="1.0" encoding="utf-8"***?>*<**manifest xmlns:android="http://schemas.android.com/apk/res/android"  
package="com.example.david.imosso"**>  
  
<**application  
 android:allowBackup="true"  
 android:icon="@mipmap/ic\_launcher"  
 android:label="@string/app\_name"  
 android:roundIcon="@mipmap/ic\_launcher\_round"  
 android:supportsRtl="true"  
 android:theme="@style/AppTheme"**>  
 <**activity android:name=".MainActivity"**>  
 <**intent-filter**>  
 <**action android:name="android.intent.action.MAIN"** />  
  
 <**category android:name="android.intent.category.LAUNCHER"** />  
 </**intent-filter**>  
 </**activity**>  
</**application**>  
  
</**manifest**>

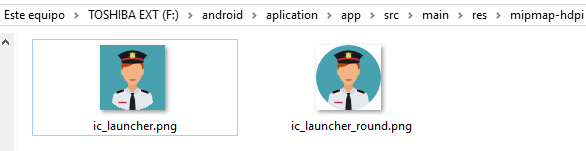
Y lo que vamos a hacer es cambiar los nombre y substituir los iconos:

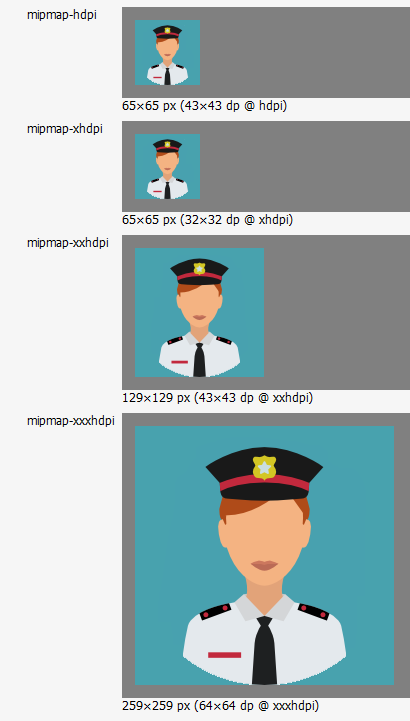
*<?***xml version="1.0" encoding="utf-8"***?>*<**manifest xmlns:android="http://schemas.android.com/apk/res/android"  
package="com.example.david.imosso"**>  
  
<**application  
 android:allowBackup="true"  
 android:icon="@mipmap/policia\_cuadrado "  
 android:label="@string/app\_name"  
 android:roundIcon="@mipmap/policia "  
 android:supportsRtl="true"  
 android:theme="@style/AppTheme"**>  
 <**activity android:name=".MainActivity"**>  
 <**intent-filter**>  
 <**action android:name="android.intent.action.MAIN"** />  
  
 <**category android:name="android.intent.category.LAUNCHER"** />  
 </**intent-filter**>  
 </**activity**>  
</**application**>  
  
</**manifest**>

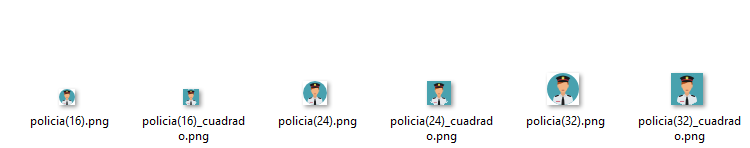
Y una vez cambiados nos aparece en las imágenes de tal manera:

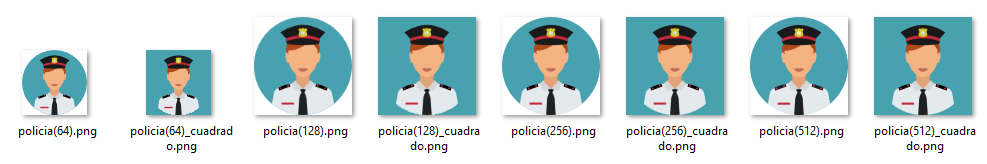


Y lo que vamos a ir haciendo es eliminar las dos imágenes y substituirlas por las de policia.png y policia\_cuadrado.png.

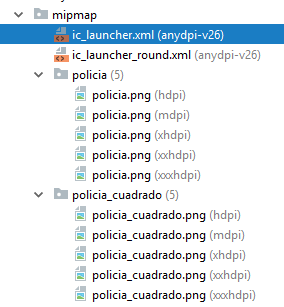
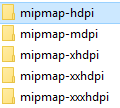
 Esto lo haremos en todas las resoluciones para que quede de la siguiente manera.

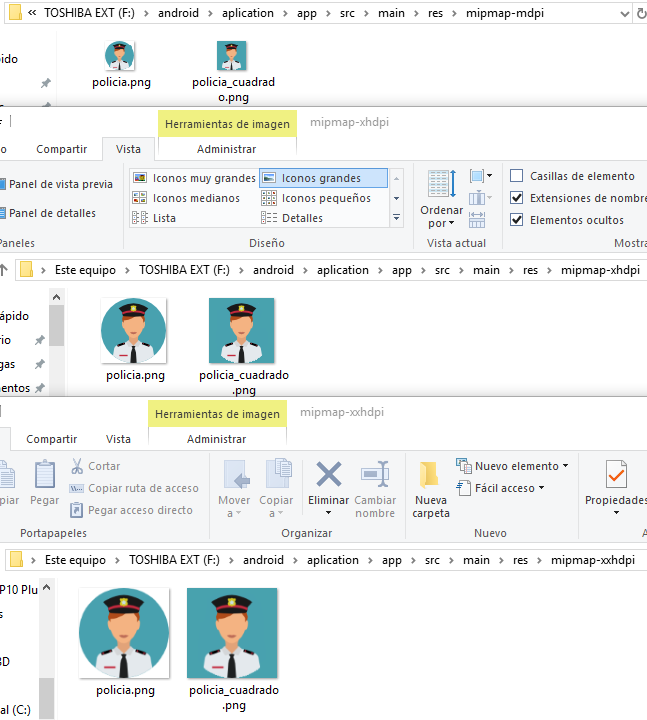


Por cierto también hemos tenido que editar el icono para que en el caso de que sea el icono cuadrado se vea con dicho formato de todas las imágenes con photoshop.

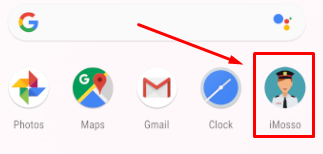


Y en cada resolución tenemos los dos iconos adaptados a esa medida el redondo y el cuadrado.

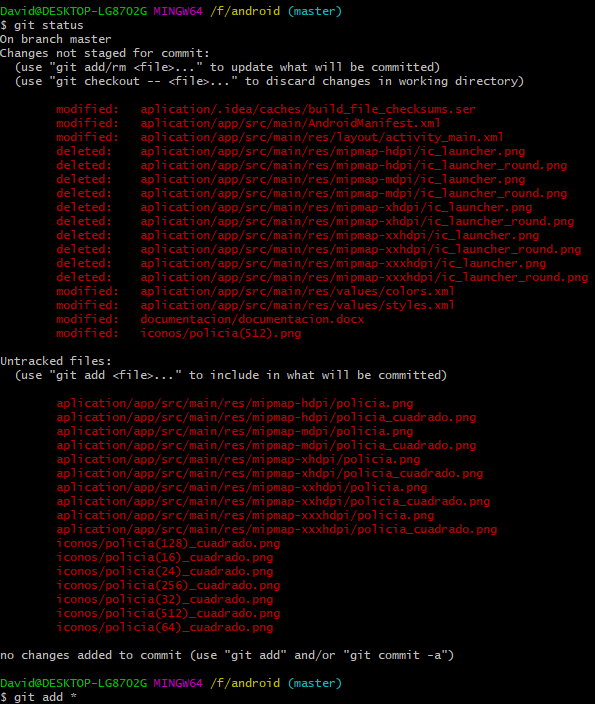
 

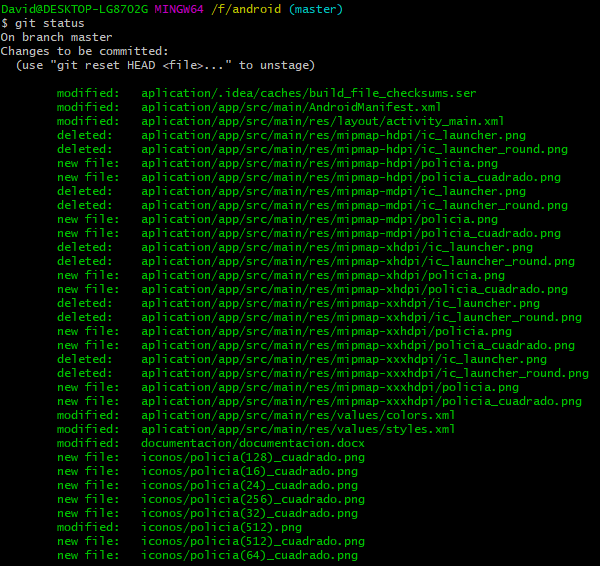


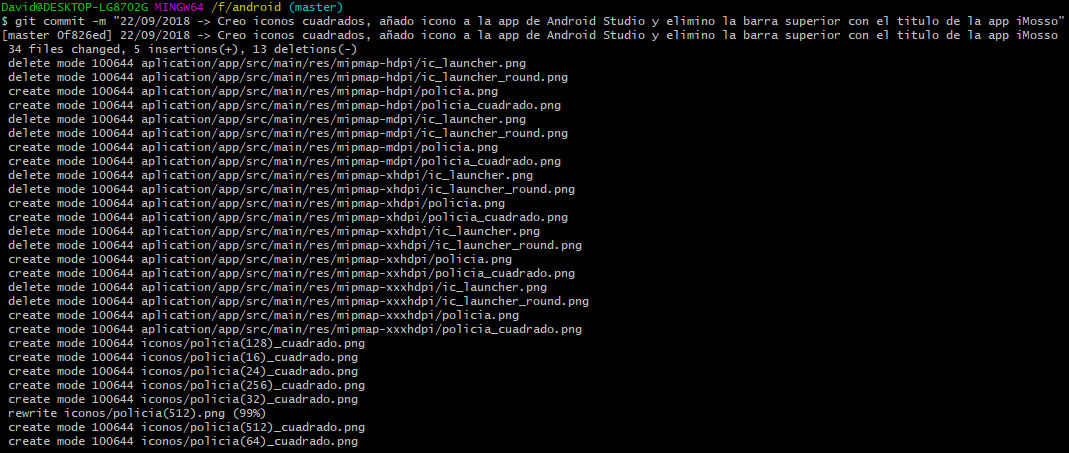
Y finalmente, ya tenemos nuestro icono:

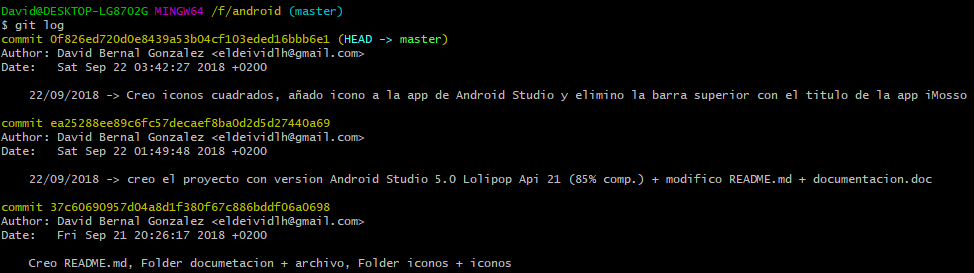


**Realizando un commit de nuevo**





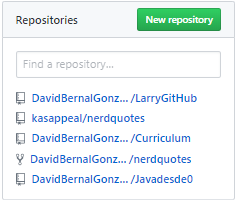


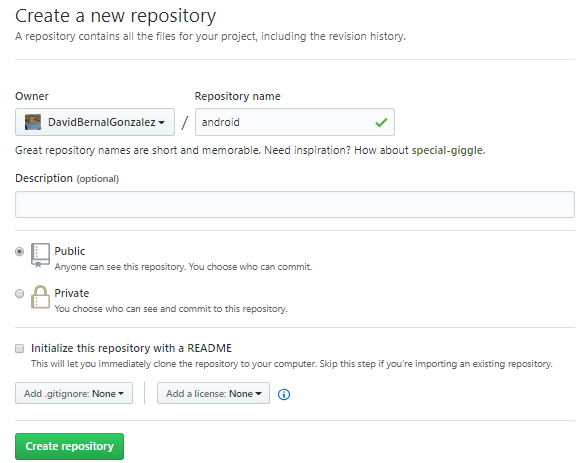


**Creando un proyecto en Git Hub y subiéndolo con git pull**

En este caso queremos compartir en Git Hub nuestra primer repositorio y de esta manera tener seguro el proyecto para ello tenemos que realizar un push.

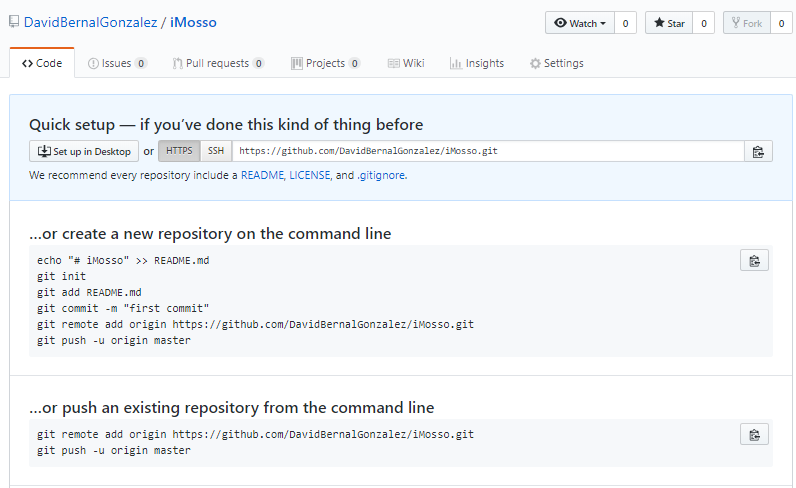
Para poder realizar el push, necesitamos previamente crear un proyecto para eso vamos a new repository:



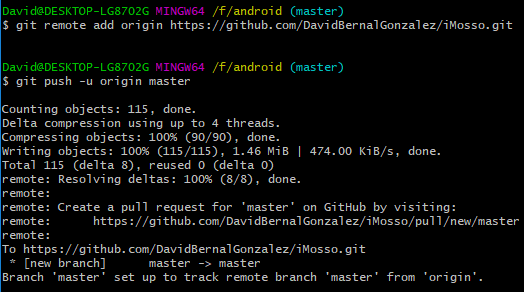


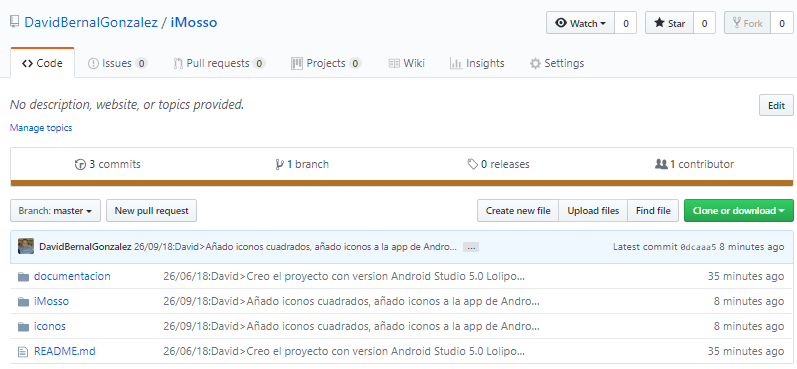
En este caso, solo se puede poner la modalidad Publica si haces la privada requiere un pago de 7$ mensuales. Git Hub, podría decirse que es la red social más popular del mundo de código.

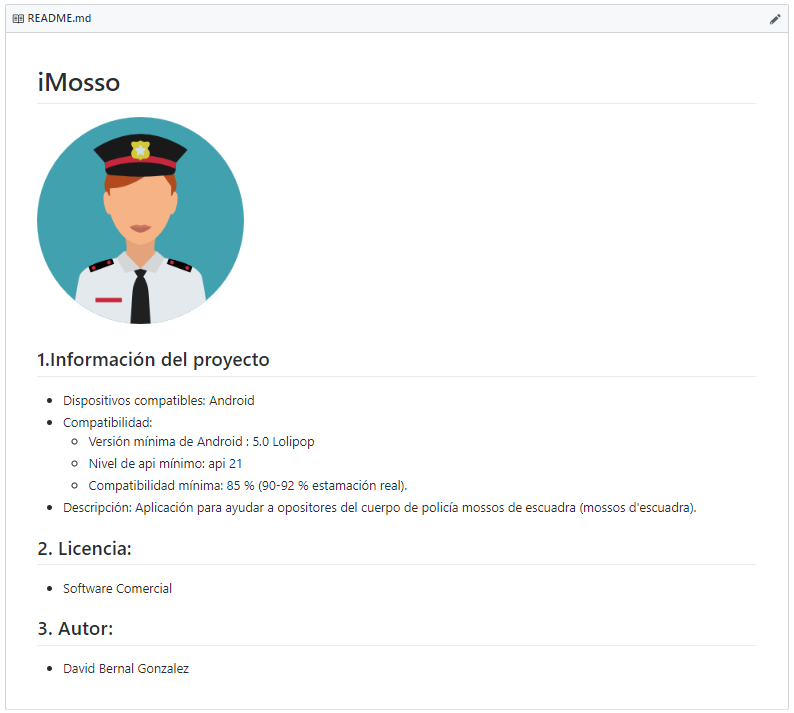
Bien una vez tenemos el proyecto, tenemos que seguir los pasos subrayados en azul.



Si copiamos y pegamos en la consola de Git Bash



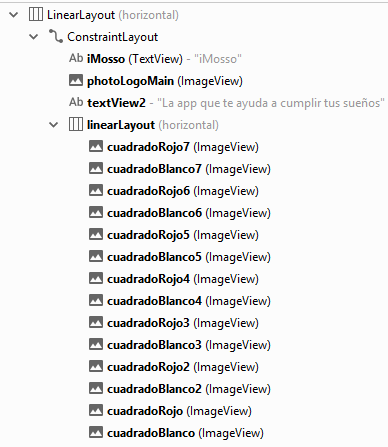




**Dándole un lavado de cara a nuestra Main Activity**

Este es el resultado por el momento:

****

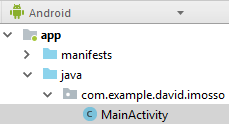


**Barra de estado/navegación transparente:**

Bien, vamos a hacer que la barra de estado sea transparente por una cuestión estética y que coincida con el color de la app.



Para modificar el color del status bar debemos ir a:



**package** com.example.david.imosso;  
  
**import** android.support.v7.app.AppCompatActivity;  
**import** android.os.Bundle;  
**import** android.view.Window;  
**import** android.view.WindowManager;  
  
**public class** MainActivity **extends** AppCompatActivity {  
  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_main***);  
 }  
}

Y añadimos:

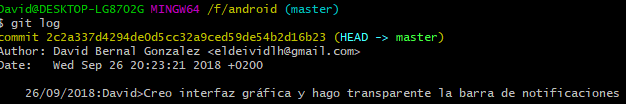
**package** com.example.david.imosso;  
  
**import** android.support.v7.app.AppCompatActivity;  
**import** android.os.Bundle;  
**import** android.view.Window;  
**import** android.view.WindowManager;  
  
**public class** MainActivity **extends** AppCompatActivity {  
  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 Window barratop = getWindow();  
 barratop.setFlags(WindowManager.LayoutParams.***FLAG\_TRANSLUCENT\_NAVIGATION***, WindowManager.LayoutParams.***TYPE\_STATUS\_BAR***);  
 setContentView(R.layout.***activity\_main***);  
 }  
}

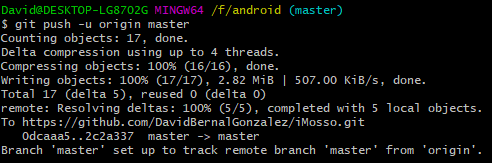
Si ahora finalmente compilamos de nuevo:



Vemos que la parte superior es transparente y que tiene el mismo color que la app.

**Realizando commit + pull**

En este caso queremos compartir en Git Hub nuestra 



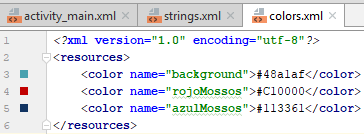
**Definiendo colores en el colors.xml**

Cuando definimos una el valor a una propiedad como puede ser el color, tenemos varias opciones para definir el color.

La primera sería asociar el color directamente a la propiedad (por ejemplo el background de la app):



La cual sería una solución aceptable pero menos mantenible que definir los colores en el xml según a lo que ataque y así poder modificar todo posteriormente de una forma mucho más rápida.



De tal manera que si cambiamos el color en el colors.xml como directamente hemos asociado la variable el color de fondo también cambiará.

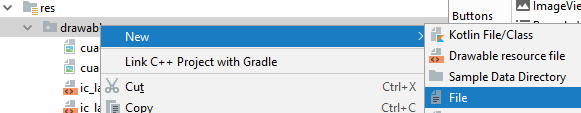


**Redondeando los botones**

Los botones por defecto, tienen una vista rectangular (ver foto de la izquierda) y vamos a ver cómo conseguir que se vean redondeados (como en la foto de la derecha).



Para ello tenemos que crear un archivo dentro de drawable y lo vamos a llamar rounded\_button.xml.





Si pegamos el siguiente código en el archivo xml que acabamos de crear y en la propiedad background del botón asociamos el archivo rounded\_button

<**shape android:shape="rectangle"  
 xmlns:android="http://schemas.android.com/apk/res/android"**>  
 <**solid android:color="@color/button"**/>  
 <**corners android:radius="150dp"**/>  
</**shape**>



Obtendremos como resultado el siguiente:



En este caso, además de tener el botón redondeado le hemos dado una pequeña línea a modo de borde que le dará un detalle bastante estético.

<**shape android:shape="rectangle"  
 xmlns:android="http://schemas.android.com/apk/res/android"**>  
 <**solid android:color="@color/button"**/>  
 <**corners android:radius="150dp"**/>  
 <**stroke  
 android:width="2dp"  
 android:color="#FFFFFF"** ></**stroke**>  
</**shape**>

Obteniendo como resultado el siguiente:



**Código Java que te cambia de vista para los botones**

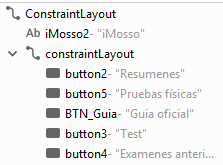
Todos los objetos de Java tienen un id que los identifica de forma única.

Si queremos realiza un cambio de Vista tenemos que definir donde estoy y a donde voy en este caso vamos de MainActivity a MenuActivity.

*//BUTTON CHANGE VIEW ON CLIC*Button BTN\_Acceder=(Button)findViewById(R.id.BTN\_Acceder);  
BTN\_Acceder.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 Intent myIntent = **new** Intent(MainActivity.**this**,  
 MenuActivity.**class**);  
 startActivity(myIntent);  
 }  
});

**Creando el menú**





**Abriendo pdf (local)**

Si pegamos el siguiente código en el archivo xml que

<https://github.com/barteksc/AndroidPdfViewer>

Tenemos que añadir:

compile 'com.github.barteksc:android-pdf-viewer:2.8.2'

Aunque desde hace poco ya no se usa compile sino implementation y por lo tanto quedará de la siguiente manera:

implementation 'com.github.barteksc:android-pdf-viewer:2.8.2'

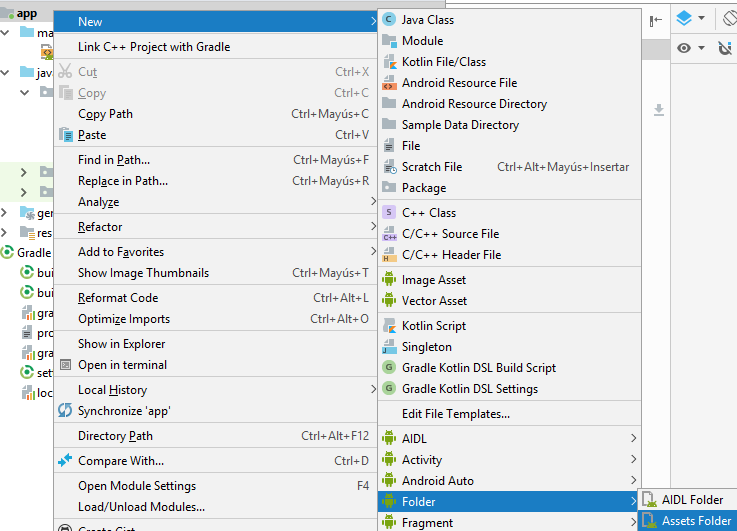


Y una vez implementada la librería, le damos a sincronizar ahora (Sync Now).

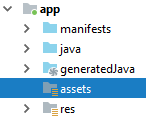
Ahora en el manifest.xml escribimos los permisos para leer archivos externos y el de internet.

*<?***xml version="1.0" encoding="utf-8"***?>*<**manifest xmlns:android="http://schemas.android.com/apk/res/android"  
 package="com.example.david.imosso"**>  
  
 <**uses-permission android:name="android.permission.READ\_EXTERNAL\_STORAGE"** />  
 <**uses-permission android:name="android.permission.INTERNET"** />  
   
 <**application  
 android:allowBackup="true"  
 android:icon="@mipmap/policia\_cuadrado"  
 android:label="@string/app\_name"  
 android:roundIcon="@mipmap/policia"  
 android:supportsRtl="true"  
 android:theme="@style/AppTheme"**>  
 <**activity android:name=".MainActivity"**>  
 <**intent-filter**>  
 <**action android:name="android.intent.action.MAIN"** />  
  
 <**category android:name="android.intent.category.LAUNCHER"** />  
 </**intent-filter**>  
 </**activity**>  
 <**activity android:name=".MenuActivity"** />  
 <**activity android:name=".Pdf\_Guide"**></**activity**>  
 </**application**>  
  
</**manifest**>

Ahora creamos el directorio que por el momento contendrá el pdf:



Y aquí lo tenemos:



Y le añadimos un pdf dentro:



Ahora pegamos:

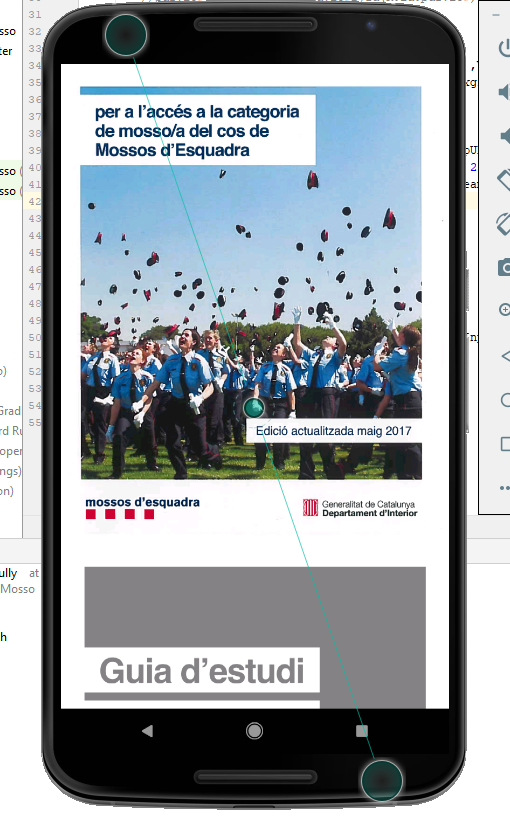
<**com.github.barteksc.pdfviewer.PDFView  
 android:id="@+id/pdfView"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"**/>

En la vista en la que queremos que se muestre lo introducimos de manera que quedé como en el ejemplo siguiente:

*<?***xml version="1.0" encoding="utf-8"***?>*<**android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:background="@color/background"  
 tools:context=".Pdf\_Guide"**>  
 <**com.github.barteksc.pdfviewer.PDFView  
 android:id="@+id/pdfView"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"**/>  
  
</**android.support.constraint.ConstraintLayout**>

Bien ahora finalmente:

**package** com.example.david.imosso;  
  
**import** android.os.AsyncTask;  
**import** android.support.v7.app.AppCompatActivity;  
**import** android.os.Bundle;  
  
**import** com.github.barteksc.pdfviewer.PDFView;  
  
**import** java.io.BufferedInputStream;  
**import** java.io.IOException;  
**import** java.io.InputStream;  
**import** java.lang.String;  
**import** java.net.HttpURLConnection;  
**import** java.net.MalformedURLException;  
**import** java.net.URL;  
  
**public class** Pdf\_Guide **extends** AppCompatActivity {  
 PDFView **pdfView**;  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***pdf\_guide***);  
  
 *//OPEN PDF local  
 // This is method from read PDF from Assets (local)* **pdfView** = (PDFView) findViewById(R.id.***pdfView***);  
 **pdfView**.fromAsset(**"Guia.pdf"**).load();  
 }}

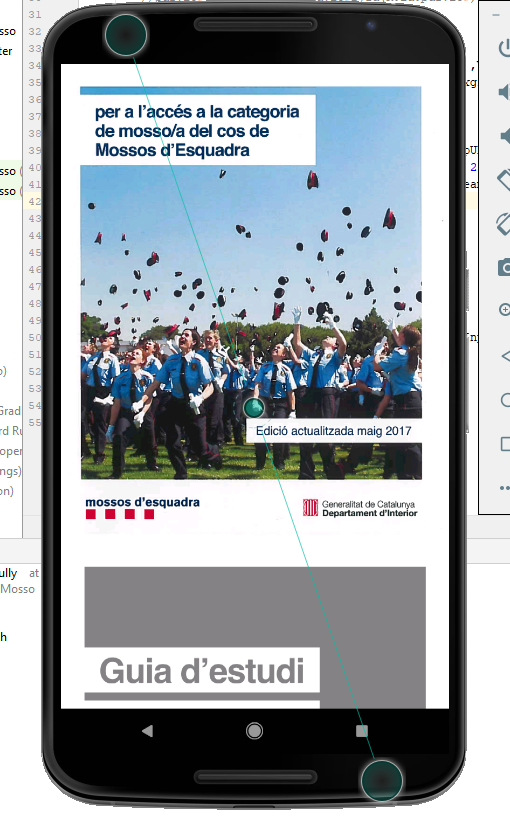


**Añadiendo el pdf (externo)**

Tras realizar la apertura del PDF interno, y comprobar que la guía de Mossos tiene copyright por parte de la Generalitat de Catalunya. No podemos introducir su PDF en nuestro directorio Asset y posteriormente abrirlo con PdfViewPager porque no tenemos autorización por parte del propietario. Pero si podemos hacer que habrá el PDF de la guía ya que tenemos dicha guía en abierto. Que es justamente lo que haremos.

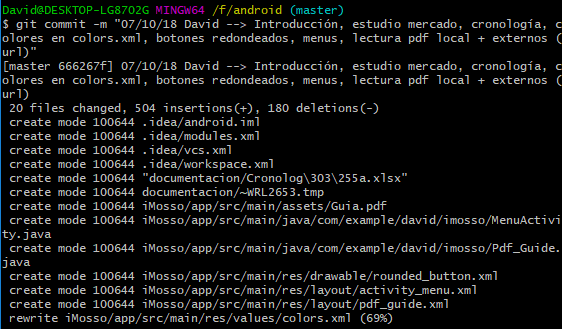
<https://github.com/voghDev/PdfViewPager>

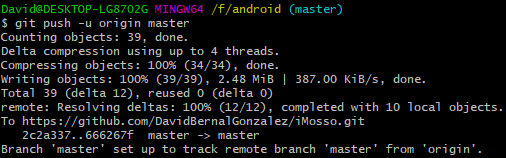
**package** com.example.david.imosso;  
  
**import** android.os.AsyncTask;  
**import** android.support.v7.app.AppCompatActivity;  
**import** android.os.Bundle;  
  
**import** com.github.barteksc.pdfviewer.PDFView;  
  
**import** java.io.BufferedInputStream;  
**import** java.io.IOException;  
**import** java.io.InputStream;  
**import** java.lang.String;  
**import** java.net.HttpURLConnection;  
**import** java.net.MalformedURLException;  
**import** java.net.URL;  
  
**public class** Pdf\_Guide **extends** AppCompatActivity {  
 PDFView **pdfView**;  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***pdf\_guide***);  
  
 *//OPEN PDF local  
 // This is method from read PDF from Assets (local)  
 //pdfView = (PDFView) findViewById(R.id.pdfView);  
 // pdfView.fromAsset("Guia.pdf").load();  
 //OPEN pdf from url* **pdfView** = (PDFView) findViewById(R.id.***pdfView***);  
   
  
 *//String url = "https://mossos.gencat.cat/ca/inici";* String url = **"https://mossos.gencat.cat/web/.content/home/01\_els\_mossos\_desquadra/ingresCos/Escala-bAsica/docs/Guia-definitiva-sencera.pdf"**;  
 **new** DownloadPdf().execute(url);  
 }*//end of on Create() method* **private class** DownloadPdf **extends** AsyncTask<String ,Void,InputStream> {  
 @Override **protected** InputStream doInBackground(String... strings) {  
 InputStream inputStream = **null**;  
 **try** {  
 URL uri = **new** URL(strings[0]);  
 HttpURLConnection urlConnection = (HttpURLConnection) uri.openConnection();  
 **if** (urlConnection.getResponseCode() == 200) {  
 inputStream = **new** BufferedInputStream(urlConnection.getInputStream());  
 }  
 } **catch** (MalformedURLException e) {  
 e.printStackTrace();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 **return** inputStream;  
 }  
 @Override **protected void** onPostExecute(InputStream inputStream) {  
 **pdfView**.fromStream(inputStream).load();  
 }  
 }*//end downloadpdf class}//end of main Activivty*}



**Realizando commit + pull**

Llega la hora de realizar otro commit, en este caso también hemos realizado un pull a GitHub para tener una backup.





**JSON vs BBDD SQL para almacenar los test**

Los motivos para realizar los test con JSON en vez de con una BBDD han sido muy meditados y analizados minuciosamente.

Entre los motivos para realizar los test con JSON destacan:

* Buscamos un proyecto que dependa poco de servidores externos debido tratando así de reducir la inversión inicial superior, los costes de mantenimiento del servidor y una reducción de los beneficios.
* Inconvenientes de realizar los test con BBDD relaciones:
  + En SQL, deberíamos de realizar dos tablas (TEST y PREGUNTAS). Y por lo tanto, suponiendo que un test tiene 30 preguntas deberíamos de realizar 30 INSERTS en la tabla TEST y 1 INSERT en la tabla PREGUNTAS.
  + Si queremos actualizar un test, deberíamos de realizar UPDATES.
  + Consultas completas comparadas con trabajar con un JSON y sin arrays por defecto.
  + Por lo tanto el tiempo de trabajo y el sistema de mantenimiento de los test es mucho más complejo y elaborado.
* Beneficios de trabajar con JSON:
  + No se realizar peticiones al servidor se trabaja desde local. Por lo tanto, no se depende de internet. Obteniendo así los datos con una velocidad superior.
  + Mayor velocidad de creación de los test, mejor estructura de los test y mejor mantenimiento (actualización).

**Leyendo String JSON I**

Bien la primera prueba con archivo JSON proviene de un STRING que le hemos pasado con estructurado en formato JSON de la siguiente manera:

"{\"empleado\":

{

\"nombre\":\"David Bernal\",

\"salario\":65000

}

}"

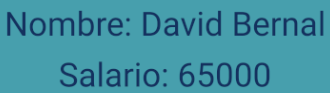
En este caso tenemos un empleado que se llama David Bernal que gana 65000.

Si vamos al Android studio, deberemos definirlo de la siguiente manera:

**package** com.example.david.imosso;  
  
**import** android.support.v7.app.AppCompatActivity;  
**import** android.os.Bundle;  
**import** android.widget.TextView;  
  
**import** org.json.JSONException;  
**import** org.json.JSONObject;  
  
**public class** Test **extends** AppCompatActivity {  
  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_test***);  
  
 String JSON\_STRING = **"{\"empleado\":{\"nombre\":\"David Bernal\",\"salario\":65000}}"**;  
 String name, salary;  
 TextView pregunta = (TextView)findViewById(R.id.***pregunta***);  
 TextView respuesta = (TextView)findViewById(R.id.***respuesta***);  
  
 **try** {  
 *// get JSONObject from JSON file* JSONObject obj = **new** JSONObject(JSON\_STRING);  
 *// fetch JSONObject named employee* JSONObject employee = obj.getJSONObject(**"empleado"**);  
 *// get employee name and salary* name = employee.getString(**"nombre"**);  
 salary = employee.getString(**"salario"**);  
 *// set employee name and salary in TextView's* pregunta.setText(**"Nombre: "**+name);  
 respuesta.setText(**"Salario: "**+salary);  
 } **catch** (JSONException e) {  
 e.printStackTrace();  
 }  
  
 }  
}

Y creamos dos TextView para referenciar la información.



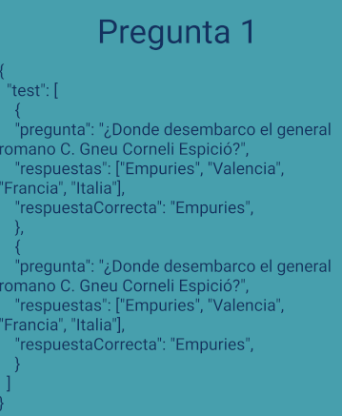


**Leyendo JSON from Assets folder I**

Hemos hecho una función que se llama loadJsonFromAsset que en este caso nos devuelve el JSON entero.

**package** com.example.david.imosso;  
  
**import** android.support.v7.app.AppCompatActivity;  
**import** android.os.Bundle;  
**import** android.widget.TextView;  
  
**import** org.json.JSONException;  
**import** org.json.JSONObject;  
  
**import** java.io.IOException;  
**import** java.io.InputStream;  
  
**public class** Test **extends** AppCompatActivity {  
  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_test***);  
 TextView pregunta = (TextView)findViewById(R.id.***pregunta***);  
 pregunta.setText(**"Pregunta 1"**);  
 TextView respuesta = (TextView)findViewById(R.id.***respuesta***);  
 respuesta.setText(loadJSONFromAsset());  
 }  
 **public** String loadJSONFromAsset() {  
 String json = **null**;  
 **try** {  
 InputStream is = getAssets().open(**"test.json"**);  
 **int** size = is.available();  
 **byte**[] buffer = **new byte**[size];  
 is.read(buffer);  
 is.close();  
 json = **new** String(buffer, **"UTF-8"**);  
 } **catch** (IOException ex) {  
 ex.printStackTrace();  
 **return null**;  
 }  
 **return** json;  
 }  
}

Estructurado de la siguiente forma:



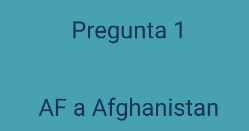
**Leyendo JSON from Assets folder II**

Vamos a seguir avanzando en el xml:

**package** com.example.david.imosso;  
  
**import** android.support.v7.app.AppCompatActivity;  
**import** android.os.Bundle;  
**import** android.widget.TextView;  
  
**import** org.json.JSONArray;  
**import** org.json.JSONException;  
**import** org.json.JSONObject;  
  
**import** java.io.IOException;  
**import** java.io.InputStream;  
**import** java.util.ArrayList;  
  
**public class** Test **extends** AppCompatActivity {  
  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_test***);  
  
 **try** {  
 JSONObject obj = **new** JSONObject(loadJSONFromAsset());  
 JSONArray countries=obj.getJSONArray(**"countries"**);  
 **for** (**int** i=0;i<countries.length();i++){  
 JSONObject jsonObject=countries.getJSONObject(0);*//i* String code=jsonObject.getString(**"code"**);  
 String nameAr=jsonObject.getString(**"nameAr"**);  
 String nameEn=jsonObject.getString(**"nameEn"**);  
 TextView pregunta = (TextView)findViewById(R.id.***pregunta***);  
 pregunta.setText(**"Pregunta 1"**);  
 TextView respuesta = (TextView)findViewById(R.id.***respuesta***);  
 respuesta.setText(code.toString() + **" "** + nameAr.toString() + **" "** + nameEn.toString());  
 }  
  
 } **catch** (JSONException e) {  
 e.printStackTrace();  
 }  
 }  
 **public** String loadJSONFromAsset() {  
 String json = **null**;  
 **try** {  
 InputStream is = getAssets().open(**"userDetail2.json"**);  
 **int** size = is.available();  
 **byte**[] buffer = **new byte**[size];  
 is.read(buffer);  
 is.close();  
 json = **new** String(buffer, **"UTF-8"**);  
 } **catch** (IOException ex) {  
 ex.printStackTrace();  
 **return null**;  
 }  
 **return** json;  
 }  
}

El xml será el siguiente:

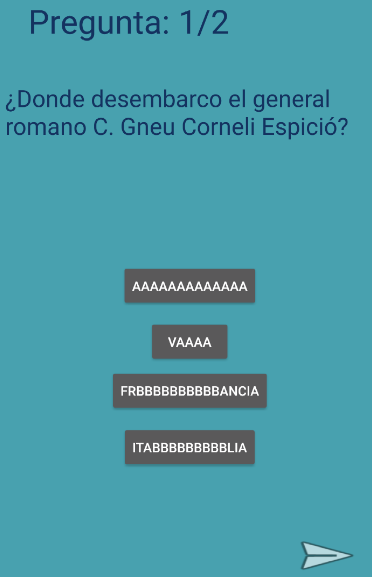
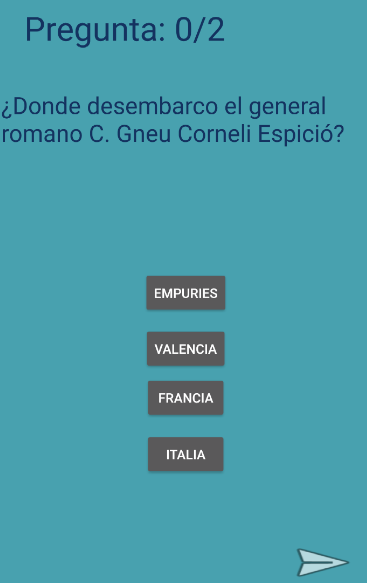
{**"countries"**: [  
 {  
 **"nameEn"**: **"Afghanistan"**,  
 **"code"**: **"AF"**,  
 **"nameAr"**: **"a"** },  
 {  
 **"nameEn"**: **"Åland Islands"**,  
 **"code"**: **"AX"**,  
 **"nameAr"**: **"a"** }]}



**Leyendo JSON from Assets folder III**

A la tercera va a la vencida:

**package** com.example.david.imosso;  
  
**import** android.content.Intent;  
**import** android.support.v7.app.AppCompatActivity;  
**import** android.os.Bundle;  
**import** android.view.View;  
**import** android.widget.Button;  
**import** android.widget.TextView;  
  
**import** org.json.JSONArray;  
**import** org.json.JSONException;  
**import** org.json.JSONObject;  
  
**import** java.io.IOException;  
**import** java.io.InputStream;  
**import** java.util.ArrayList;  
  
**public class** Test **extends** AppCompatActivity {  
 **int increment** = 0;  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_test***);  
 Button BTN\_next=(Button)findViewById(R.id.***BTN\_next***);  
 BTN\_next.setOnClickListener(**new** View.OnClickListener() {  
 @Override  
 **public void** onClick(View v) {  
 showJSONFromAsset(+1);  
 **increment**++;  
 }  
 });  
 showJSONFromAsset(0);  
 }  
 **public** String loadJSONFromAsset() {  
 String json = **null**;  
 **try** {  
 InputStream is = getAssets().open(**"test.json"**);  
 **int** size = is.available();  
 **byte**[] buffer = **new byte**[size];  
 is.read(buffer);  
 is.close();  
 json = **new** String(buffer, **"UTF-8"**);  
 } **catch** (IOException ex) {  
 ex.printStackTrace();  
 **return null**;  
 }  
 **return** json;  
 }  
  
 **public void** showJSONFromAsset(**int** increment){  
 **try** {  
 JSONObject obj = **new** JSONObject(loadJSONFromAsset());  
 JSONArray test = obj.getJSONArray(**"test"** + 1);  
 **for** (**int** i = 0; i < test.length(); i++) {  
 JSONObject jsonObject = test.getJSONObject(increment);*//i* String pregunta = jsonObject.getString(**"pregunta"**);  
 String respuesta1 = jsonObject.getString(**"respuesta1"**);  
 String respuesta2 = jsonObject.getString(**"respuesta2"**);  
 String respuesta3 = jsonObject.getString(**"respuesta3"**);  
 String respuesta4 = jsonObject.getString(**"respuesta4"**);  
 String respuestaCorrecta = jsonObject.getString(**"respuestaCorrecta"**);  
  
 TextView TV\_pregunta = (TextView) findViewById(R.id.***TV\_pregunta***);  
 TV\_pregunta.setText(pregunta);  
 TextView TV\_num = (TextView) findViewById(R.id.***TV\_num***);  
 TV\_num.setText(**"Pregunta: "** + (increment + 1) + **"/"** + test.length());  
 Button BTN\_respuesta1 = (Button) findViewById(R.id.***BTN\_respuesta1***);  
 BTN\_respuesta1.setText(respuesta1.toString());  
 Button BTN\_respuesta2 = (Button) findViewById(R.id.***BTN\_respuesta2***);  
 BTN\_respuesta2.setText(respuesta2.toString());  
 Button BTN\_respuesta3 = (Button) findViewById(R.id.***BTN\_respuesta3***);  
 BTN\_respuesta3.setText(respuesta3.toString());  
 Button BTN\_respuesta4 = (Button) findViewById(R.id.***BTN\_respuesta4***);  
 BTN\_respuesta4.setText(respuesta4.toString());  
 }  
 }**catch**( JSONException e) {  
 e.printStackTrace();  
 }  
 }  
}



Principal material de apoyo:

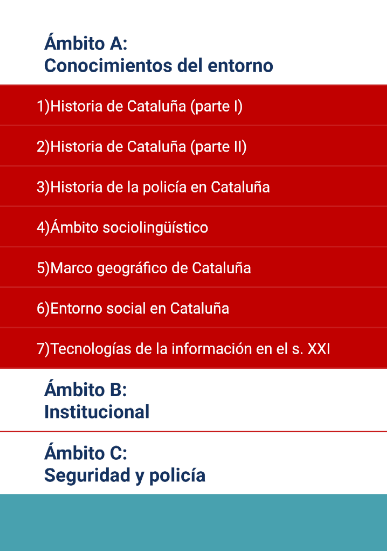
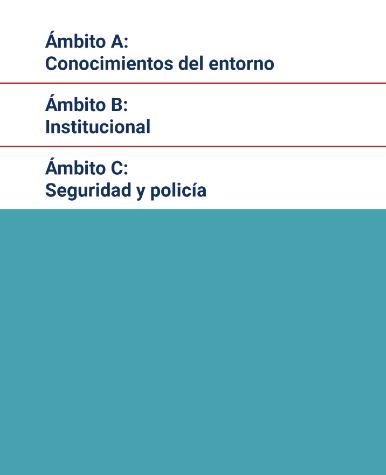
<http://www.androidcoding.in/2017/11/12/android-parsing-local-json-url-fetching-data-local-url/>

<https://abhiandroid.com/programming/json#Sample_JSON_format>

**Creando el Expandable List View**

Bien nuestra intención es desplegar un menú que tenga las diferentes categorías y vaya mostrando los diferentes temas.

Dando como resultado lo siguiente:



Bien pues para ello, vamos a crear una nueva vista. En mi caso, dicha vista se llamará activity\_test\_menu.xml y en ella insertaremos el ExpandableListView que hay introducido justo aquí abajo.

*<?***xml version="1.0" encoding="utf-8"***?>*<**android.support.constraint.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 xmlns:app="http://schemas.android.com/apk/res-auto"  
 xmlns:tools="http://schemas.android.com/tools"  
 android:layout\_width="match\_parent"  
 android:layout\_height="match\_parent"  
 android:background="@color/background"  
 tools:context=".TestMenu"**>  
  
 <**ExpandableListView  
 android:id="@+id/lvExp"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:background="@color/texto2"**></**ExpandableListView**>  
</**android.support.constraint.ConstraintLayout**>

Para continuar, vamos a ir al directorio layout (dentro de res) y vamos a crear una nueva vista llamada list\_group.xml que contendrá un Text View. De la siguiente manera:

*<?***xml version="1.0" encoding="utf-8"***?>*<**LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 android:orientation="vertical" android:layout\_width="match\_parent"  
 android:padding="8dp"  
 android:background="@color/colorPrimary"  
 android:layout\_height="wrap\_content"**>  
  
 <**TextView  
 android:id="@+id/lblListHeader"  
 android:paddingLeft="?android:attr/expandableListPreferredItemPaddingLeft"  
 android:textSize="20dp"  
 android:textColor="@color/texto"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"** />  
  
</**LinearLayout**>

Bien, ahora creamos una vista llamada list\_item.xml y ponemos lo siguiente:

*<?***xml version="1.0" encoding="utf-8"***?>*<**LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
 android:orientation="vertical" android:layout\_width="match\_parent"  
 android:layout\_height="55dip"**>  
  
 <**TextView  
 android:id="@+id/lblListItem"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:paddingLeft="?android:attr/expandableListPreferredChildPaddingLeft"  
 android:paddingTop="10dp"  
 android:paddingBottom="10dp"  
 android:textColor="@color/white"  
 android:textSize="16dip"** />  
  
</**LinearLayout**>

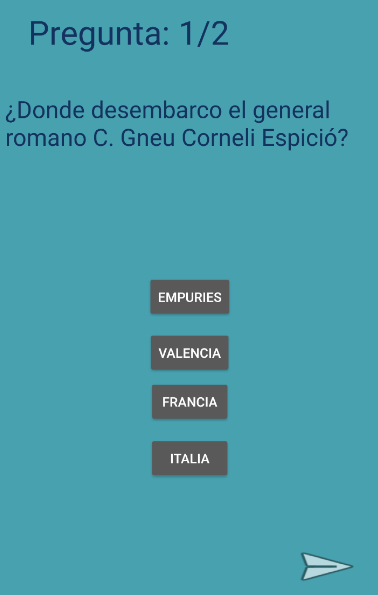
Y por último, creamos una clase en java llamada ExpandableListAdapter.class en la que añadimos el siguiente código:

**package** com.example.david.imosso;  
  
**import** android.content.Context;  
**import** android.graphics.Typeface;  
**import** android.view.LayoutInflater;  
**import** android.view.View;  
**import** android.view.ViewGroup;  
**import** android.widget.BaseExpandableListAdapter;  
**import** android.widget.TextView;  
  
**import** java.util.HashMap;  
**import** java.util.List;  
  
**public class** ExpandableListAdapter **extends** BaseExpandableListAdapter {  
 **private** Context **context**;  
 **private** List<String> **listDataHeader**;  
 **private** HashMap<String,List<String>> **listHashMap**;  
  
 **public** ExpandableListAdapter(Context context, List<String> listDataHeader, HashMap<String, List<String>> listHashMap) {  
 **this**.**context** = context;  
 **this**.**listDataHeader** = listDataHeader;  
 **this**.**listHashMap** = listHashMap;  
 }  
  
 @Override  
 **public int** getGroupCount() {  
 **return listDataHeader**.size();  
 }  
  
 @Override  
 **public int** getChildrenCount(**int** i) {  
 **return listHashMap**.get(**listDataHeader**.get(i)).size();  
 }  
  
 @Override  
 **public** Object getGroup(**int** i) {  
 **return listDataHeader**.get(i);  
 }  
  
 @Override  
 **public** Object getChild(**int** i, **int** i1) {  
 **return listHashMap**.get(**listDataHeader**.get(i)).get(i1); *// i = Group Item , i1 = ChildItem* }  
  
 @Override  
 **public long** getGroupId(**int** i) {  
 **return** i;  
 }  
  
 @Override  
 **public long** getChildId(**int** i, **int** i1) {  
 **return** i1;  
 }  
  
 @Override  
 **public boolean** hasStableIds() {  
 **return false**;  
 }  
  
 @Override  
 **public** View getGroupView(**int** i, **boolean** b, View view, ViewGroup viewGroup) {  
 String headerTitle = (String)getGroup(i);  
 **if**(view == **null**)  
 {  
 LayoutInflater inflater = (LayoutInflater)**this**.**context**.getSystemService(Context.***LAYOUT\_INFLATER\_SERVICE***);  
 view = inflater.inflate(R.layout.***list\_group***,**null**);  
 }  
 TextView lblListHeader = (TextView)view.findViewById(R.id.***lblListHeader***);  
 lblListHeader.setTypeface(**null**, Typeface.***BOLD***);  
 lblListHeader.setText(headerTitle);  
 **return** view;  
 }  
  
 @Override  
 **public** View getChildView(**int** i, **int** i1, **boolean** b, View view, ViewGroup viewGroup) {  
 **final** String childText = (String)getChild(i,i1);  
 **if**(view == **null**)  
 {  
 LayoutInflater inflater = (LayoutInflater)**this**.**context**.getSystemService(Context.***LAYOUT\_INFLATER\_SERVICE***);  
 view = inflater.inflate(R.layout.***list\_item***,**null**);  
 }  
  
 TextView txtListChild = (TextView)view.findViewById(R.id.***lblListItem***);  
 txtListChild.setText(childText);  
 **return** view;  
 }  
  
 @Override  
 **public boolean** isChildSelectable(**int** i, **int** i1) {  
 **return true**;  
 }  
}

Finalmente, vamos a la clase de la vista TestMenu.class en la que queremos que se muestre el menú y pegamos lo siguiente:

**package** com.example.david.imosso;  
  
**import** android.annotation.SuppressLint;  
**import** android.content.Intent;  
**import** android.support.v7.app.AppCompatActivity;  
**import** android.os.Bundle;  
**import** android.view.View;  
**import** android.widget.Button;  
**import** android.widget.ExpandableListView;  
**import** android.widget.ListView;  
**import** android.widget.TextView;  
**import** android.widget.Toast;  
  
**import** java.nio.FloatBuffer;  
**import** java.util.ArrayList;  
**import** java.util.HashMap;  
**import** java.util.List;  
  
**import static** android.widget.Toast.\*;  
  
**public class** TestMenu **extends** AppCompatActivity {  
 String **titulo1**=**""**;  
 ExpandableListView **listView**;  
 ExpandableListAdapter **listAdapter**;  
 List<String> **listDataHeader**;  
 HashMap<String,List<String>> **listHash**;  
 @SuppressLint(**"WrongViewCast"**)  
 @Override  
 **protected void** onCreate(Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 setContentView(R.layout.***activity\_test\_menu***);  
  
 **listView** = (ExpandableListView)findViewById(R.id.***lvExp***);  
 initData();  
 **listAdapter** = **new** ExpandableListAdapter(**this**,**listDataHeader**, **listHash**);  
 **listView**.setAdapter(**listAdapter**);  
 Click\_group();  
 Click\_child();  
 }  
  
 **private void** initData() {  
 **listDataHeader** = **new** ArrayList<>();  
 **listHash** = **new** HashMap<>();  
  
 **listDataHeader**.add(**"Ámbito A: \n"** +  
 **"Conocimientos del entorno"**);  
 **listDataHeader**.add(**"Ámbito B: \n"** +  
 **"Institucional"**);  
 **listDataHeader**.add(**"Ámbito C: \n"** +  
 **"Seguridad y policía"**);  
  
 List<String> categoriaA = **new** ArrayList<>();  
 **titulo1** = **"1)Historia de Cataluña (parte I)"**;  
 categoriaA.add(**titulo1**);  
 categoriaA.add(**"2)Historia de Cataluña (parte II)"**);  
 categoriaA.add(**"3)Historia de la policía en Cataluña"**);  
 categoriaA.add(**"4)Ámbito sociolingüístico"**);  
 categoriaA.add(**"5)Marco geográfico de Cataluña"**);  
 categoriaA.add(**"6)Entorno social en Cataluña"**);  
 categoriaA.add(**"7)Tecnologías de la información en el s. XXI"**);  
  
 List<String> categoriaB = **new** ArrayList<>();  
 categoriaB.add(**"Expandable ListView"**);  
 categoriaB.add(**"Google Map"**);  
 categoriaB.add(**"Chat Application"**);  
 categoriaB.add(**"Firebase "**);  
  
 List<String> categoriaC = **new** ArrayList<>();  
 categoriaC.add(**"Xamarin Expandable ListView"**);  
 categoriaC.add(**"Xamarin Google Map"**);  
 categoriaC.add(**"Xamarin Chat Application"**);  
 categoriaC.add(**"Xamarin Firebase "**);  
  
 List<String> uwp = **new** ArrayList<>();  
 uwp.add(**"UWP Expandable ListView"**);  
 uwp.add(**"UWP Google Map"**);  
 uwp.add(**"UWP Chat Application"**);  
 uwp.add(**"UWP Firebase "**);  
  
 **listHash**.put(**listDataHeader**.get(0),categoriaA);  
 **listHash**.put(**listDataHeader**.get(1),categoriaB);  
 **listHash**.put(**listDataHeader**.get(2),categoriaC);  
  
 }  
  
 **private void** Click\_group(){  
 **listView**.setOnGroupClickListener(**new** ExpandableListView.OnGroupClickListener() {  
 @Override  
 **public boolean** onGroupClick(ExpandableListView expandableListView, View view, **int** groupPosition, **long** l) {  
 *//Toast.makeText(TestMenu.this, listDataHeader.get(groupPosition), Toast.LENGTH\_LONG).show();* **return false**;  
 };  
 });  
 }  
  
 **private void** Click\_child(){  
 **listView**.setOnChildClickListener(**new** ExpandableListView.OnChildClickListener() {  
 @Override  
 **public boolean** onChildClick(ExpandableListView expandableListView, View view, **int** groupPosition, **int** childPosition, **long** l) {  
 *//Toast.makeText(TestMenu.this, listHash.get(listDataHeader.get(groupPosition)).get(childPosition), Toast.LENGTH\_SHORT).show();  
 //Toast.makeText(TestMenu.this, listHash.get(listDataHeader.get(groupPosition)).get(childPosition), Toast.LENGTH\_SHORT).show();  
 //if the position its = titulo 1 go to...* **if**(**listHash**.get(**listDataHeader**.get(groupPosition)).get(childPosition)==**titulo1**){  
 Intent myIntent = **new** Intent(TestMenu.**this**,  
 Test.**class**);  
 startActivity(myIntent);  
 } **else if**(**listHash**.get(**listDataHeader**.get(groupPosition)).get(childPosition)!=**titulo1**){  
 Intent myIntent = **new** Intent(TestMenu.**this**,  
 Test.**class**);  
 startActivity(myIntent);  
 }  
 **return false**;  
 }  
 });  
 }  
}

Para que funcione perfectamente hemos tenido que crear dos funciones una que seleccione el elemento padre que sería la función Click\_group y luego otra con el click\_child que sería para los hijos del grupo. Los Toast, son los mensajes con los que hemos mostrado los que nos muestra por pantalla. En este caso, hemos una vez realizado lo que queríamos hemos deshabilitado ambos. Y hemos realizado un if se hace un click en el grupo y el elemento child vaya a al test correspondiente. Aunque por el momento solo va a un test que tenemos temporalmente hecho para hacer pruebas.



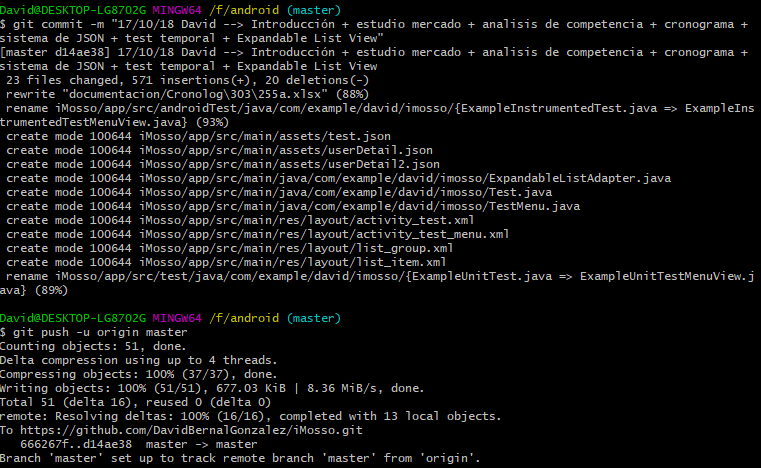
Principal material de apoyo:

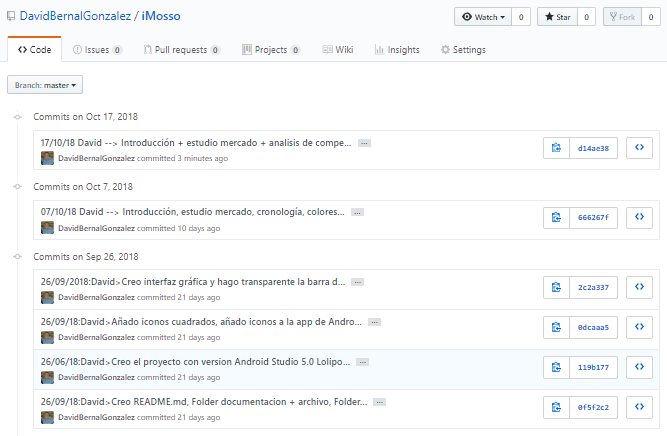
<https://www.youtube.com/watch?v=jZxZIFnJ9jE> Expandable list view

<https://www.youtube.com/watch?v=91k4u5_Cl6s> tutorial de listexpand view click mesage

**Realizando commit + pull**

Llega la hora de realizar otro commit, en este caso también hemos realizado un pull a GitHub para tener una backup.





**21/10/18 David --> Verificando si el test existe y enviando un parámetro de una vista a otra**

Nuestro objetivo en este punto es realizar una función que primeramente nos aseguré de que el archivo existe. En el caso de que el archivo exista, realice un cambio de actividad a Test y que mediante a SharedPreferences, le asignemos el test que tiene que ejecutar de tal manera que con una sola vista, podamos abrir todos los test de una manera sencilla. En el caso de que no mostraremos por pantalla “Test no disponible”.

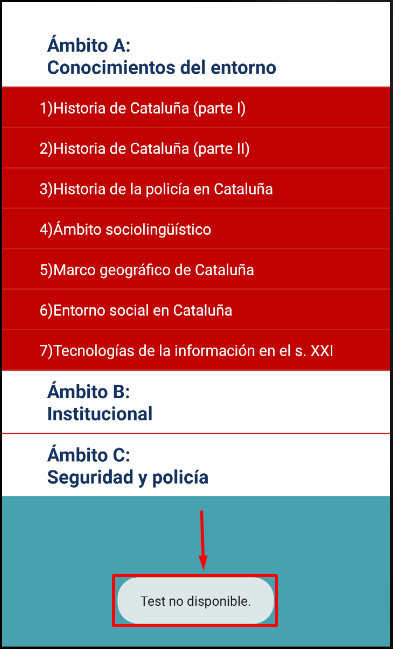
**private void** existeArchivo(String name){  
 AssetManager mg = getResources().getAssets();  
 InputStream is = **null**;  
 **try** {  
 **try** (InputStream inputStream = is = mg.open(name)) {  
 *//File exists so do something with it* SharedPreferences prefs = getSharedPreferences(**"MisPreferencias"**,Context.***MODE\_PRIVATE***);  
 SharedPreferences.Editor editor = prefs.edit();  
 editor.putString(**"test"**, name);  
 editor.commit();  
 *//finish();* Intent myIntent = **new** Intent(TestMenu.**this**,  
 Test.**class**);  
 startActivity(myIntent);  
 Toast.*makeText*(TestMenu.**this**, **"file is already there"**, Toast.***LENGTH\_SHORT***).show();  
 }  
 } **catch** (IOException ex) {  
 *//file does not exist* Toast.*makeText*(TestMenu.**this**, **"Not find file "**, Toast.***LENGTH\_SHORT***).show();  
 } **finally** {  
 **if** (is != **null**) {  
 **try** {  
 is.close();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
}

Lo hemos hecho desde una función pero ¿porque?

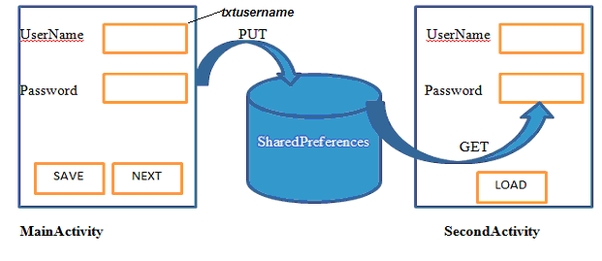
**private void** Click\_child(){  
 **listView**.setOnChildClickListener(**new** ExpandableListView.OnChildClickListener() {  
 @Override  
 **public boolean** onChildClick(ExpandableListView expandableListView, View view, **int** groupPosition, **int** childPosition, **long** l) {**if**(**listHash**.get(**listDataHeader**.get(groupPosition)).get(childPosition)==**titulo1**){  
 existeArchivo(**"test.json"**);  
 } **else if**(**listHash**.get(**listDataHeader**.get(groupPosition)).get(childPosition)==**titulo2**){  
 existeArchivo(**"test1.json"**);  
 } **else if**(**listHash**.get(**listDataHeader**.get(groupPosition)).get(childPosition)==**titulo3**){  
 existeArchivo(**"test2.json"**);  
 }  
 **return false**;  
 }  
 });  
 }

Si nos fijamos de esta manera con poner existeArchivo(**"test.json"**); el código es mucho más mantenible ya que no tiene redundancia y con ello evitamos incrementar las líneas del código y trabajamos.

Si no existe el archivo como en el caso del test2.json y hacemos clic:



Bien que es lo hemos hecho con SharedPreferences es pasar una información de una vista a otra. En este caso el test que tiene que abrir la Actividad.



Mediante al siguiente código guardamos el test en la key test y el value será el nombre del test y realizamos un commit finalmente.

SharedPreferences prefs = getSharedPreferences(**"MisPreferencias"**,Context.***MODE\_PRIVATE***);  
 SharedPreferences.Editor editor = prefs.edit();  
 editor.putString(**"test"**, name);  
 editor.commit();

En la vista que va a recibir el SharedPreferences, tenemos que cargarlo de la siguiente manera.  
SharedPreferences prefs =  
 getSharedPreferences(**"MisPreferencias"**,Context.***MODE\_PRIVATE***);  
String ruta = prefs.getString(**"test"**, **null**);

**Realizando commit + pull**

Llega la hora de realizar otro commit, en este caso también hemos realizado un pull a GitHub para tener una backup.