



Hito de Dispositivos móviles

MARIO VALVERDE
CAMPUS FP

ÍNDICE

Pág1.....	Índice
Pág2.....	Tecnologías usadas
Pág3.....	Explicación del emulador
Pág4.....	Explicación del código

TECNOLOGÍAS USADAS

Como IDE se ha utilizado: Android Studio

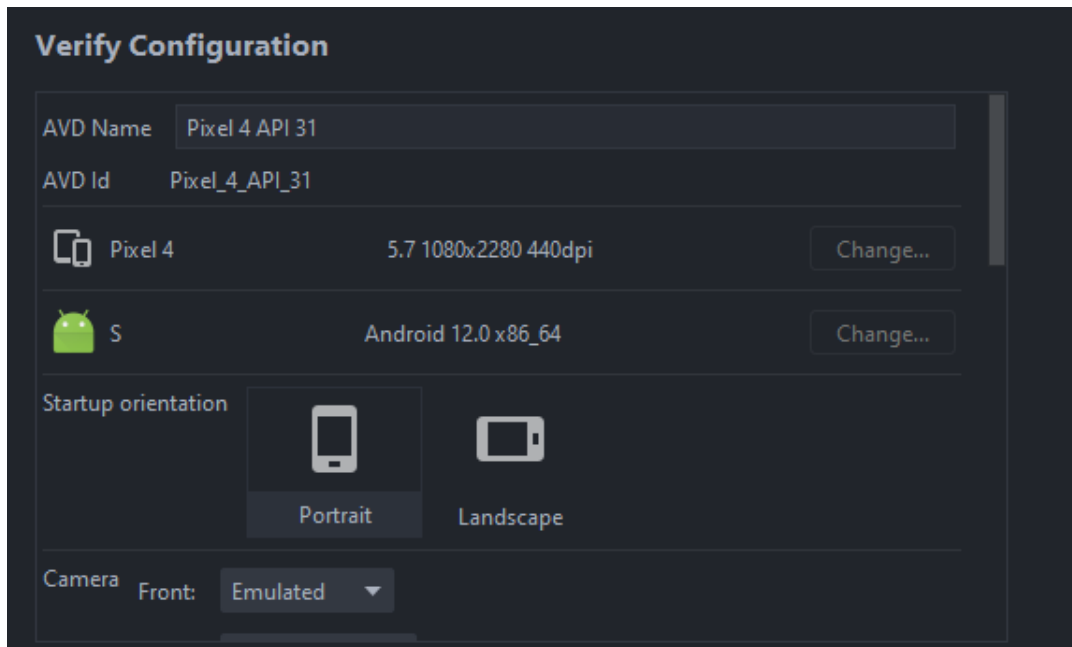


Y el lenguaje que se ha usado ha sido Kotlin:




EXPLICACIÓN DEL EMULADOR

En este primer apartado veremos qué tipo de emulador he elegido y sus características




En primer lugar, he elegido este emulador Pixel 4 debido a que contiene PlayStore por lo que a la hora de arrancar el emulador podemos usar PlayStore dentro del emulador.

Pixel 4		5,7"	1080x2280	440dpi
---------	---	------	-----------	--------

La primera ventana nos indica el tamaño del emulador, la segunda la resolución del emulador y la ultima la densidad.

Lo siguiente que elegimos en el emulador es el System Image, que en este caso he elegido la que se llama "S" y la imagen es de x86_64, y debido a que el Level API es una 31, que esto quiere decir que estamos más cerca de nuevas versiones que otras más antiguas y a su vez es recomendable porque es compatible con Google play.

Memory and Storage

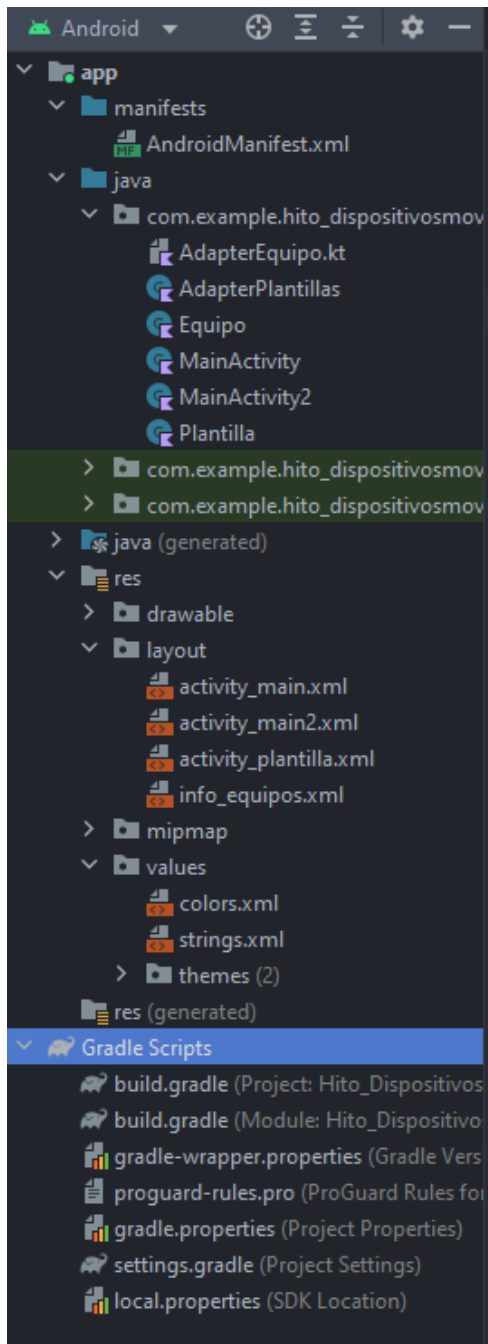
RAM:	1536	MB
VM heap:	228	MB
Internal Storage:	6144	MB
SD card:	<input checked="" type="radio"/> Studio-managed 512	MB
	<input type="radio"/> External file	
	<input type="radio"/> No SDCard	

Y por último veremos que es importante saber que dentro de los emuladores es necesario tener más de 5000MB para que el emulador pueda funcionar correctamente y arranque las apps con la mayor rapidez y facilidad posible.

EXPLICACIÓN DEL CÓDIGO

Ahora pasaremos a la explicación del código desde la primera carpeta usada, hasta las propiedades de Gradle.

Primeramente, adjunto una foto de cómo está estructurado mi proyecto y en las carpetas que se encuentran:



Empezaremos por el manifest:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.example.hito_dispositivosmoviles">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="Hito_DispositivosMoviles"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.Hito_DispositivosMoviles">
        <activity
            android:name=".Plantilla"
            android:exported="false" />
        <activity
            android:name=".MainActivity2"
            android:exported="false" />
        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

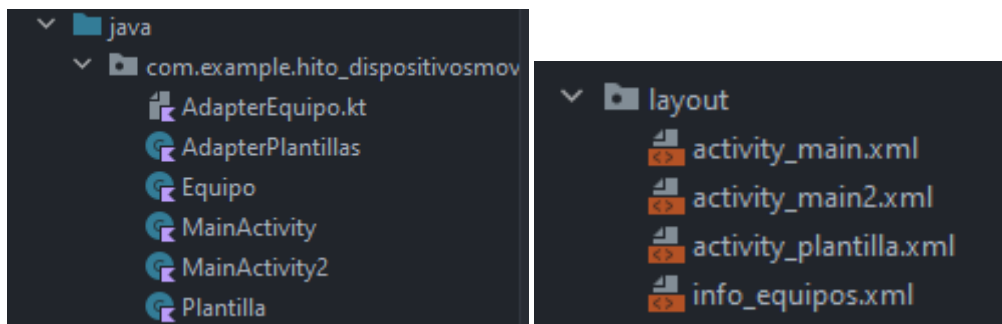
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

En primer lugar, nos damos cuenta que es un archivo con extensión .xml, donde se guarda esta app y también el nombre del paquete en el que se encuentra.

Y dentro de la application en primer lugar vemos las características de esta app, y después en la cual tenemos varios activity, lo que quiere decir que hemos usado el objeto intent.

Ahora entramos de verdad a lo que es el código desarrollado por mí y a la vez de analizar este código también observaremos la carpeta layout para poder entenderlo correctamente:



Antes de nada hay que fijarse que el código de esta app está dentro de la carpeta java y dentro del paquete com.example.hito, que esto es importante fijarse para luego a la hora de añadir alguna clase o archivo.

Empecemos por el primero de todos que es el MainActivity:

```
package com.example.hito_dispositivosmoviles

import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.Button

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        var boton1=findViewById<Button>(R.id.btn_pasar)

        boton1.setOnClickListener { it: View!
            val equipo=Intent( packageContext: this, MainActivity2::class.java)
            startActivity(equipo)
        }
    }
}
```

Esta primera clase no tiene mucho código debido a que lo más relevante es el uso del objeto intent, que esto lo que nos permite es el poder pasar de una actividad a otra.

Como bien nos podemos fijar es que en Kotlin para declarar una variable puede ser con “val” o “var”. Y que para poder usar un objeto intent, la gran mayoría de veces suele ir dentro de la propiedad setOnClickListener. Y dentro de él es donde desarrollamos y vemos que dentro del intent se pasa a la actividad que quieres pasar que en mi caso se llama MainActivity2.

Y para entender más el código de porque la propiedad Clicklistener se le pone a botón1 la vemos en el layout:



El botón acceder tiene un **id** de btn_pasar, y en el código podemos ver que nosotros accedemos a este botón a través del método findViewById.

Como bien he dicho antes el botón nos lleva en este caso al MainActivity2, asique lo analizaremos:

```
package com.example.hito_dispositivosmoviles

import ...

class MainActivity2 : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main2)

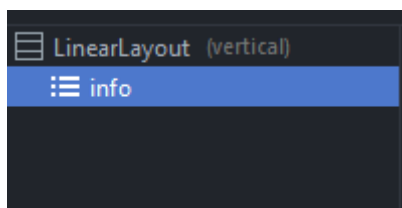
        val info= findViewById<RecyclerView>(R.id.info)
        info.layoutManager = LinearLayoutManager(context=this)

        val listaEquipos = ArrayList<Equipo>()

        listaEquipos.add(Equipo(nombre="Real Madrid", pais="ESPAÑA", mote="Merengues"))
        listaEquipos.add(Equipo(nombre="Barcelona", pais="ESPAÑA", mote="Cules"))
        listaEquipos.add(Equipo(nombre="Atletico de Madrid", pais="ESPAÑA", mote="Colchoneros"))
        listaEquipos.add(Equipo(nombre="Sevilla", pais="ESPAÑA", mote="Palanganas"))
        listaEquipos.add(Equipo(nombre="Manchester City", pais="Inglaterra", mote="Sky Blues"))

        val adaptadorEquipo = AdaptadorEquipo(listaEquipos, context=this)
        info.adapter = adaptadorEquipo
    }
}
```

Aquí podemos observar que tenemos un ArrayList en el cual le pasamos unos equipos. Y estos equipos se le pasa la información de la clase Equipo y para encontrar su información se encuentra dentro un RecyclerView con id info:



Y la clase equipo se compone de lo siguiente:

```
package com.example.hito_dispositivosmoviles

class Equipo {
    var nombre: String? = null
    var pais: String? = null
    var mote: String? = null

    constructor(nombre: String, pais: String, mote: String) {
        this.nombre = nombre
        this.pais = pais
        this.mote = mote
    }
}
```

Ahora pasaremos a ver una clase bastante importante que es la de AdapterEquipos en la cual se muestra bastante información importante.

```
package com.example.hito_dispositivosmoviles

import ...

class AdapterEquipo(val listaEquipos: List<Equipo>, private val context: Context): RecyclerView.Adapter<AdapterEquipo.ViewHolder>() {

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): AdapterEquipo.ViewHolder {
        val vista = LayoutInflater.from(parent.context).inflate(R.layout.info_equipos, parent, attachToRoot: false)
        return ViewHolder(vista)
    }

    override fun onBindViewHolder(holder: AdapterEquipo.ViewHolder, position: Int) {
        val equipo = listaEquipos[position]

        holder.tvNombre.text = equipo.nombre
        holder.tvPais.text = equipo.pais
        holder.tvMote.text = equipo.mote
        holder.cardIf.setOnClickListener() { // it: View!

            Toast.makeText(context, text: "$position", Toast.LENGTH_LONG).show()
            val plantilla=Intent(context,Plantilla::class.java)
            plantilla.putExtra(name: "Plantilla", equipo.nombre)
            context.startActivity(plantilla)

            //val plantilla= context.startActivity(Intent(context, Plantilla::class.java))
        }
    }

    override fun getItemCount(): Int {
        return listaEquipos.size
    }
}
```

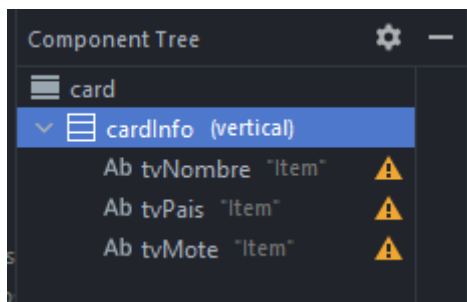
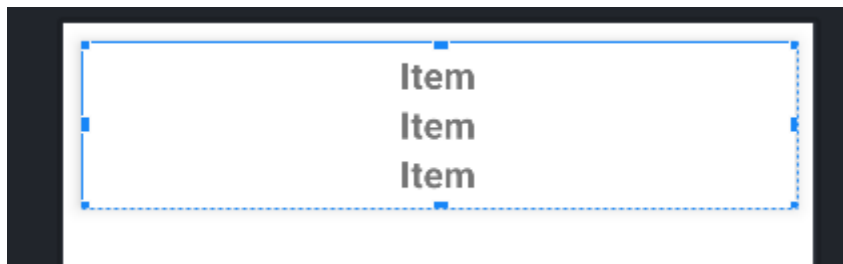
Iremos paso por paso:

Antes que nada, vemos que dentro de la clase le pasamos unos parámetros los cuales les indicamos de donde van a obtener la información y donde la tiene que pintar.

En este primer override observamos lo siguiente:

```
override fun onCreateView(parent: ViewGroup, viewType: Int): AdapterEquipo.ViewHolder {  
    val vista = LayoutInflater.from(parent.context).inflate(R.layout.info_equipos, parent, attachToRoot: false)  
    return ViewHolder(vista)  
}
```

El cual primero le pasamos unos parámetros y extendemos de AdapterEquipo que está dentro de un ViewHolder. Después he hecho una variable vista la cual le paso el contenido del layout de info_equipos y después retornamos toda la información de vista, ahora vemos el layout de info_equipos:



En este caso para que se vea la información de manera distinta lo que he hecho ha sido hacer un card, que lo que provoca es que se vea la información en forma de una tarjeta.

Ahora vemos el siguiente override:

```
override fun onBindViewHolder(holder: AdapterEquipo.ViewHolder, position: Int) {  
    val equipo = listaEquipos[position]  
  
    holder.tvNombre.text = equipo.nombre  
    holder.tvPais.text = equipo.pais  
    holder.tvMote.text = equipo.mote  
    holder.cardIf.setOnClickListener() { it: View!  
  
        Toast.makeText(context, text: "$position", Toast.LENGTH_LONG).show()  
        val plantilla=Intent(context, Plantilla::class.java)  
        plantilla.putExtra(name: "Plantilla", equipo.nombre)  
        context.startActivity(plantilla)  
  
        //val plantilla= context.startActivity(Intent(context, Plantilla::class.java))  
    }  
}
```

En esta función vemos que también extiende del principal de la clase AdapterEquipo, en esta clase es importante fijarse en el parámetro position ya que nos ayudara posteriormente a ordenar la información.

Observamos que a esta función se le pasan los textos con el id nombre, país, mote. Y vemos que dentro del idCardInfo le hemos pasado otro intent que nos redirige a la clase Plantilla. Una cosa a destacar es que aparte de pasar de a una nueva clase, le añadimos a plantilla una propiedad .putExtra, que esto lo que nos permite es visualizar la información que la pasemos dentro de la clase Plantilla. Y por último arrancamos la actividad Plantilla.

Y por último analizamos la clase Plantilla:

```
class Plantilla : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_plantilla)

        val equipo = intent.getStringExtra(name: "Plantilla")
        println(equipo)
        val plantillas = mapOf(
            "Real Madrid" to arrayOf(
                "Courtois",
                "Militao",
                "Alaba",
                "Rudiguer",
                "Mendy",
                "Kroos",
                "Modric",
                "Valverde",
                "Vinicius",
                "Benzema",
                "Rodrygo"
            ),
            "Barcelona" to arrayOf(
                "Ter Stegen",
                "Pique",
                "Kounde",
                "Jordi Alba",
                "Araujo",
                "Gavi",
                "Pedri",
                "De Jong",
                "Depay",
                "Lewa",
                "Dembele"
            ),
        ),
```

En esta clase plantilla lo que he hecho es hacer una variable la cual recoger la información del putExtra y toda la información la recojo dentro de un array y la paso por un mapOf y después la pinto por un arrayOf que es donde le paso la información.

Y por último pasamos a la explicación del Gradle:

```
android {
    compileSdk 32

    defaultConfig {
        applicationId "com.example.hito_dispositivosmoviles"
        minSdk 27
        targetSdk 32
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-
        }
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
    kotlinOptions {
        jvmTarget = '1.8'
    }
    viewBinding{
        enabled true
    }
}
```

Dentro de este módulo es importante fijarnos en la configuración de del sdk, sobre todo si nuestro proyecto va a ser usado por otras personas o cogemos proyectos de otras personas es importante fijarse en esto debido a que no todas las personas tenemos el mismo sdk o versión.

Y dentro de esta captura también hay que fijarse que por defecto, kotlin no trae por defecto el viewBinding por lo que hay que importarlo y ponerlo en “true” para poder usarlo.

```
dependencies {  
  
    implementation 'androidx.core:core-ktx:1.7.0'  
    implementation 'androidx.appcompat:appcompat:1.4.1'  
    implementation 'com.google.android.material:material:1.5.0'  
    implementation 'androidx.constraintlayout:constraintlayout:2.1.3'  
    testImplementation 'junit:junit:4.+'  
    androidTestImplementation 'androidx.test.ext:junit:1.1.3'  
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'  
}
```

Y, por último, también dependiendo el proyecto que tengas en mente hay que fijarse las dependencias que hay que importar o usar.