



MANUAL DE

Capacitación Android Básico en Kotlin



2018

CONTENIDO

LAB0: “HOLA MUNDO EN ANDROID STUDIO”	3
LAB01: NAVEGACIÓN Y GUI	12
LAB02: PERSISTENCIA DE DATOS	26
LAB03: USO DE SERVICIOS WEB	39
LAB04: USO DE CÁMARA NATIVA	47
LAB05: USO DE GOOGLE MAPS Y GPS	56
LABORATORIO DE EVALUACIÓN 1	72
LABORATORIO DE EVALUACIÓN 2	74



LAB0: “HOLA MUNDO EN ANDROID STUDIO”

Objetivos

- Identificar el proceso de creación de un nuevo proyecto de Android en Android Studio.
- Identificar cómo está conformado el árbol de carpetas de un proyecto de Android en Android Studio.
- Identificar las herramientas y componentes base para el desarrollo y ejecución de la App.
- Crear y ejecutar una App que diga “Hola Mundo” en el Emulador de Android o en un dispositivo físico:
 - Imprimiendo en el LogCat.
 - Asignando texto a un TextView.
 - Generando un mensaje de tipo Toast.

Resumen

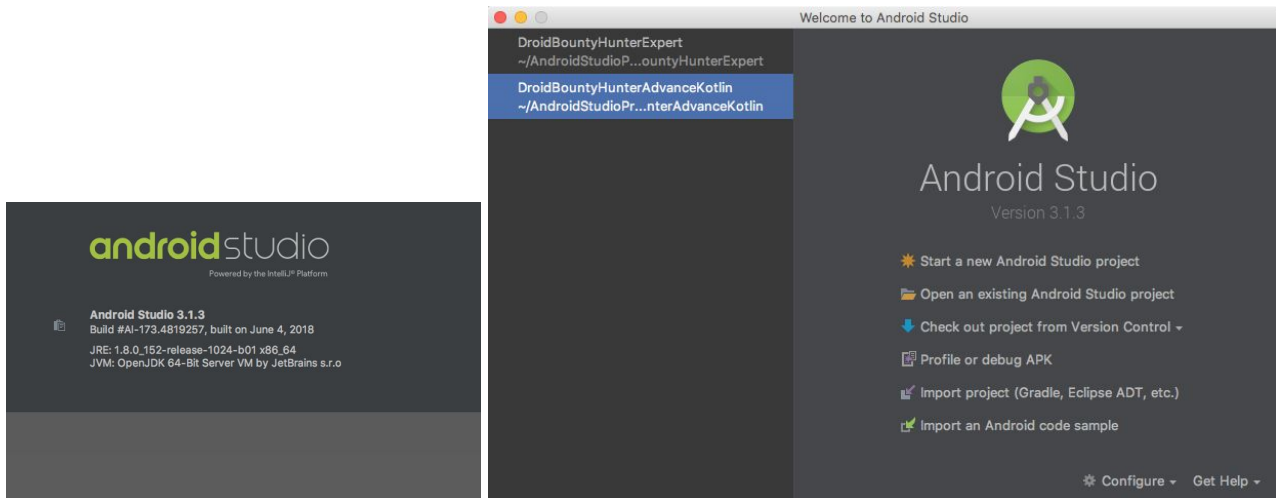
En este laboratorio se identificará el proceso de creación de un nuevo proyecto de Android en Android Studio con los elementos de configuración iniciales utilizando un Activity base con una etiqueta que contenga la leyenda “Hola Mundo!!!”, se configurará un Emulador adecuado de Android y se ejecutará el proyecto sobre el Emulador creado. Así mismo, se deberá imprimir en el LogCat el mismo texto y también generar un mensaje tipo Toast.

Punto de Partida

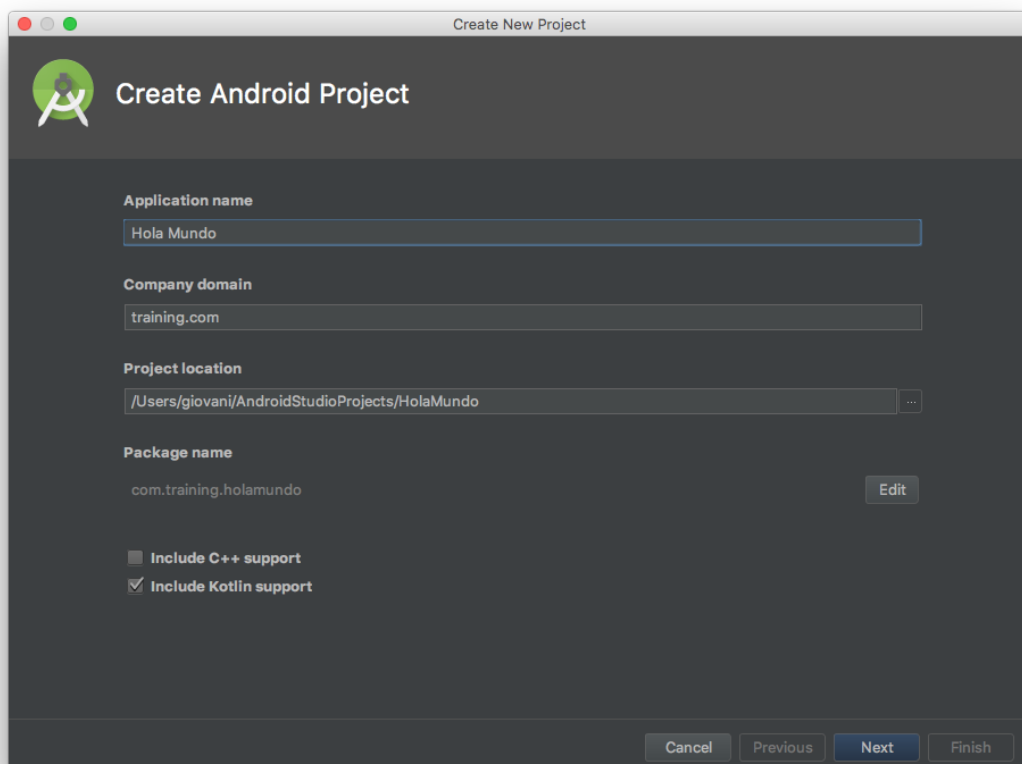
Para este laboratorio solo se necesita el IDE de Android Studio con el SDK de Android actualizado, no hay código de inicio de referencia.

Ejecución del Laboratorio

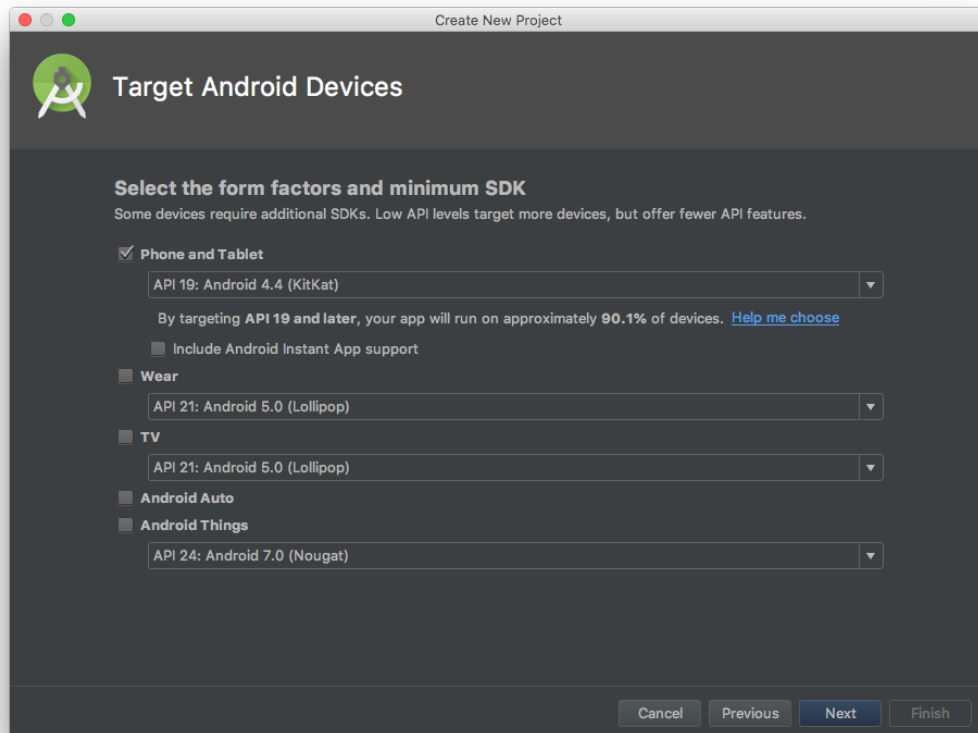
- Ejecutar Android Studio y crear un nuevo workspace para el manejo de los proyectos de Android.



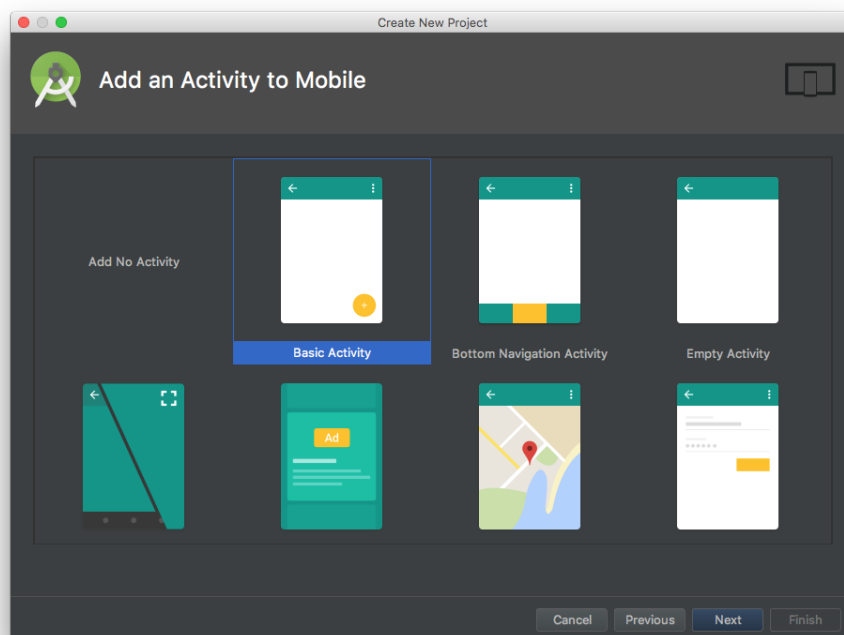
- En la ventana “Welcome to Android Studio” se seleccionará “Start a new Android Studio Project”.
- Configuraremos el proyecto de la siguiente manera, editando solamente el “Application name” en “Hola Mundo” y dando click al botón “Next”, asegurate que la opción “Include Kotlin support” esté seleccionada:



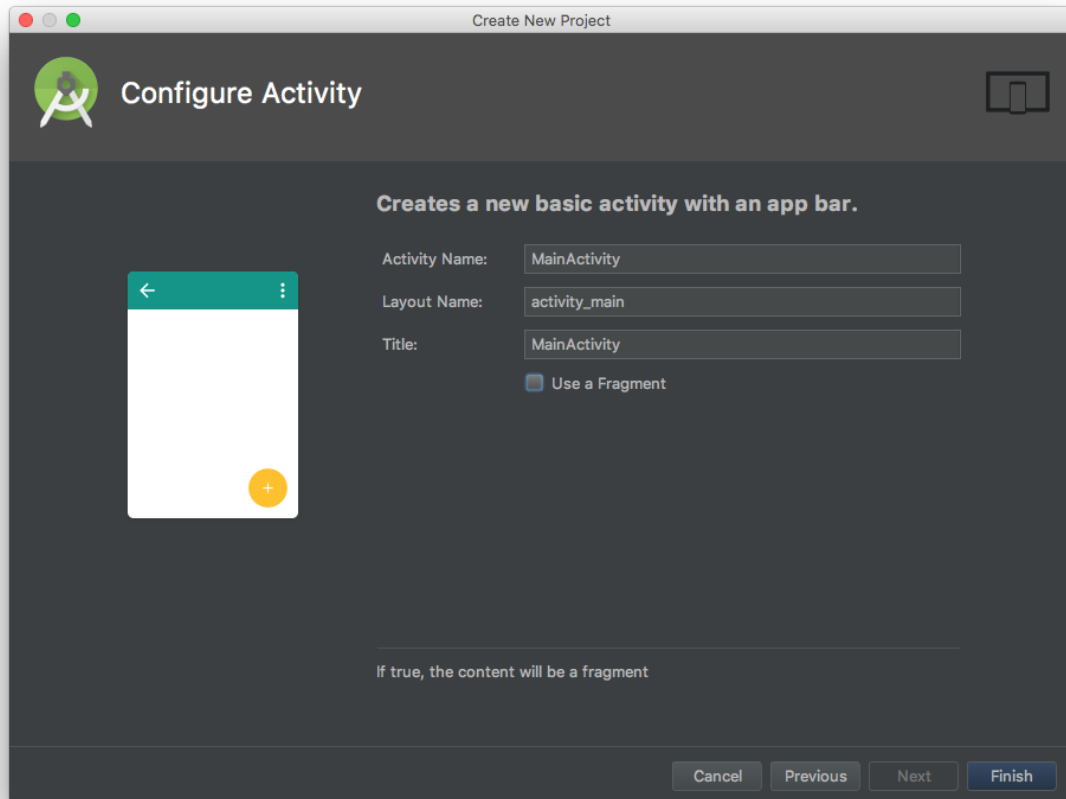
- Dejaremos en “checked” <el apartado de “Phone and Tablet” con un mínimo SDK nivel 16 correspondiente a la versión 4.1 de Android:



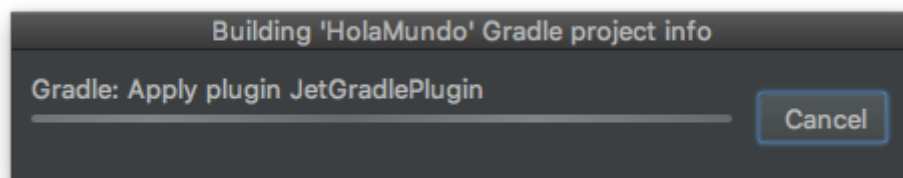
- El siguiente paso será seleccionar el tipo de “Template” a utilizar para nuestra aplicación, en este caso seleccionaremos “Blank Activity”, en otras versiones de Android Studio se mostrará “Basic Activity”.



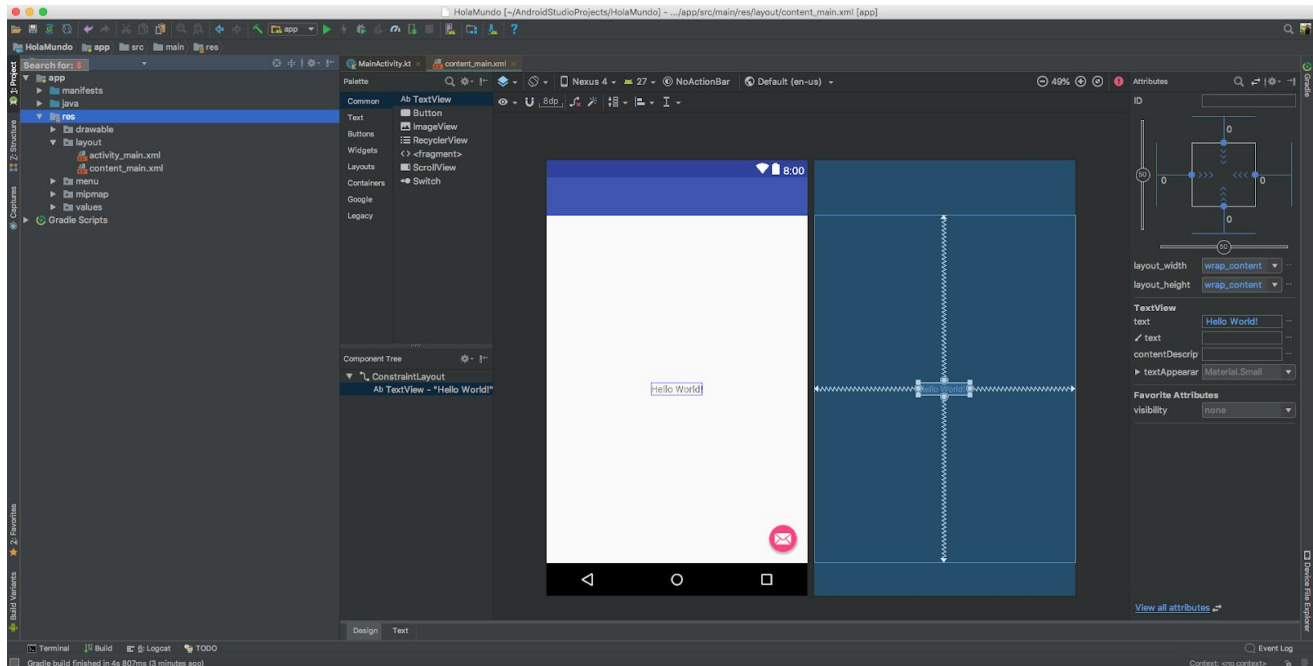
- Por último tendremos la ventana para colocar el Nombre del Activity, nombre del Layout, el título de la ventana y el nombre del layout del menú. Colocaremos los siguientes valores y damos click a “Finish”.



- Comenzará el trabajo de creación del proyecto y de los archivos necesarios para la compilación y ejecución del mismo.



- El IDE creará el proyecto y mostrará el Layout inicial de la App, con una Etiqueta por default con el texto “Hello World!” el cual se debe actualizar seleccionandolo y cambiándolo en su propiedad Text del lado izquierdo de la pantalla, de modo que indique “Hola Mundo!!!”:




- En el archivo MainActivity.kt, modificar el método onCreate para que contenga la siguiente información:

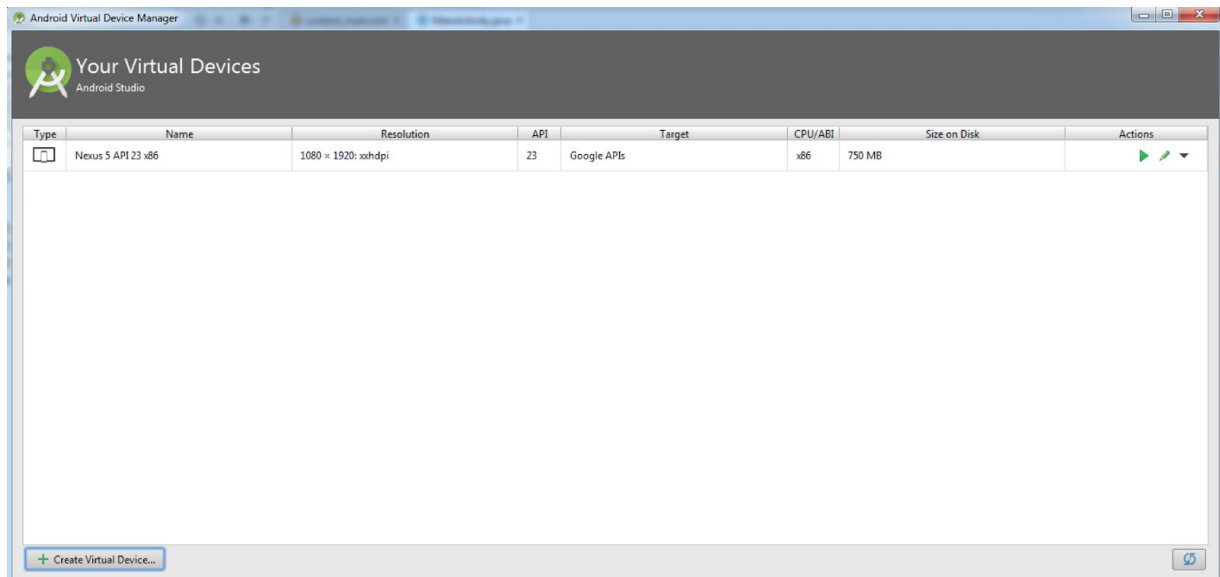
```

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    setSupportActionBar(toolbar)

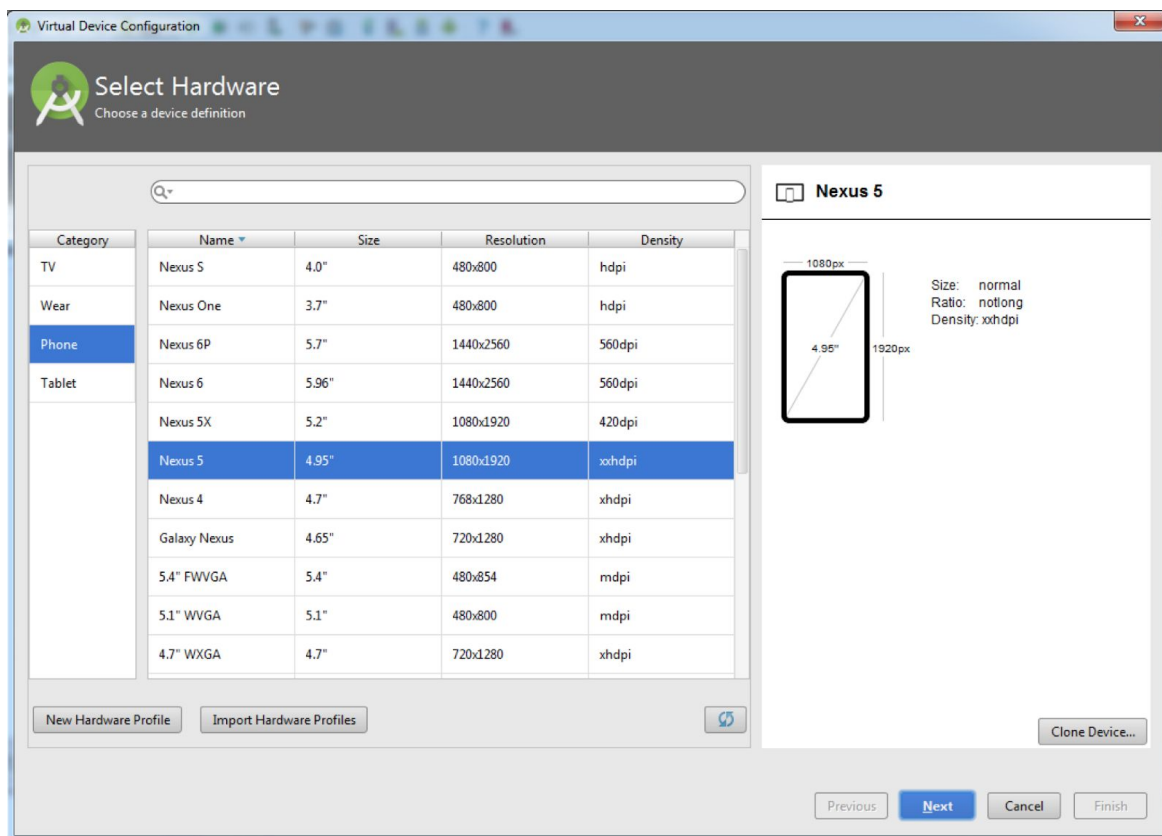
    fab.setOnClickListener { view ->
        Snackbar.make(view, "Replace with your own action", Snackbar.LENGTH_LONG
            .setAction("Action", null)).show()
    }
    Log.d("Main Activity", "Hola Mundo!! - Logger view")
    Toast.makeText(this, "Hola Mundo!!", Toast.LENGTH_LONG).show()
}

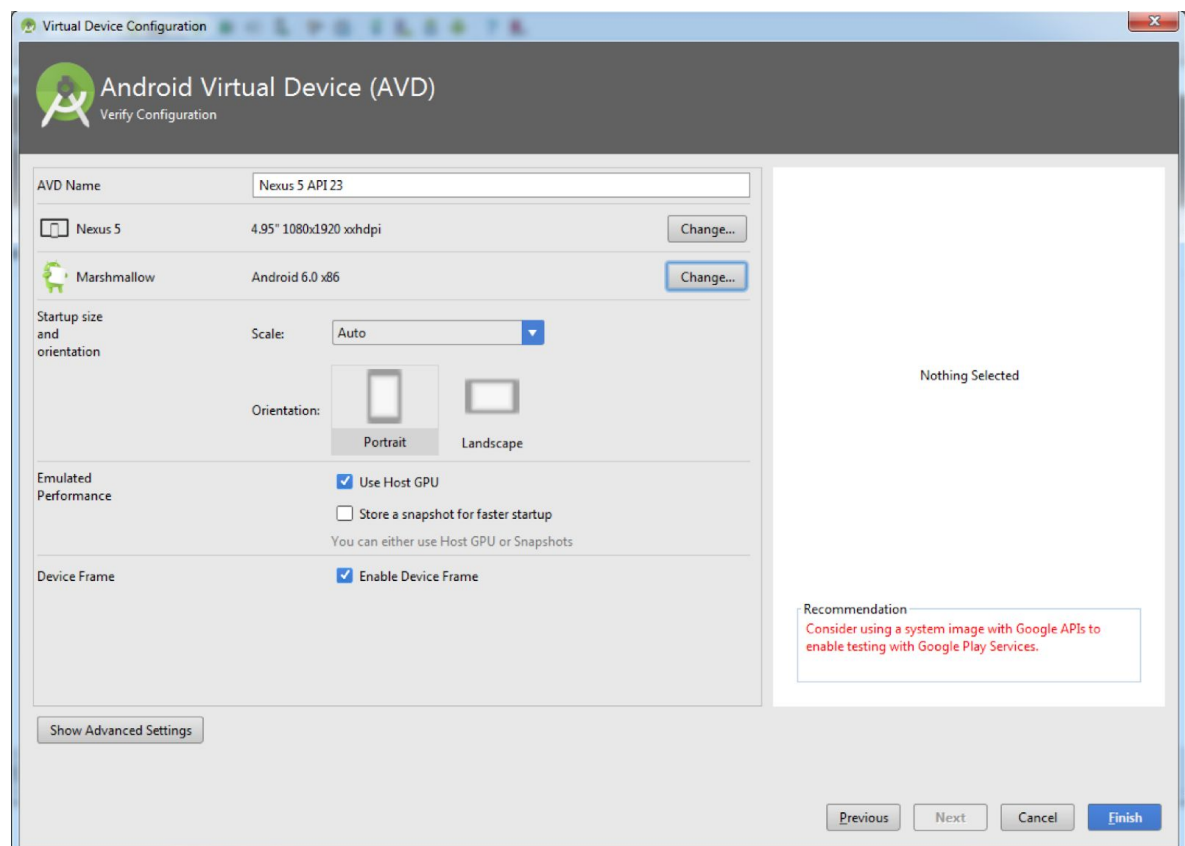
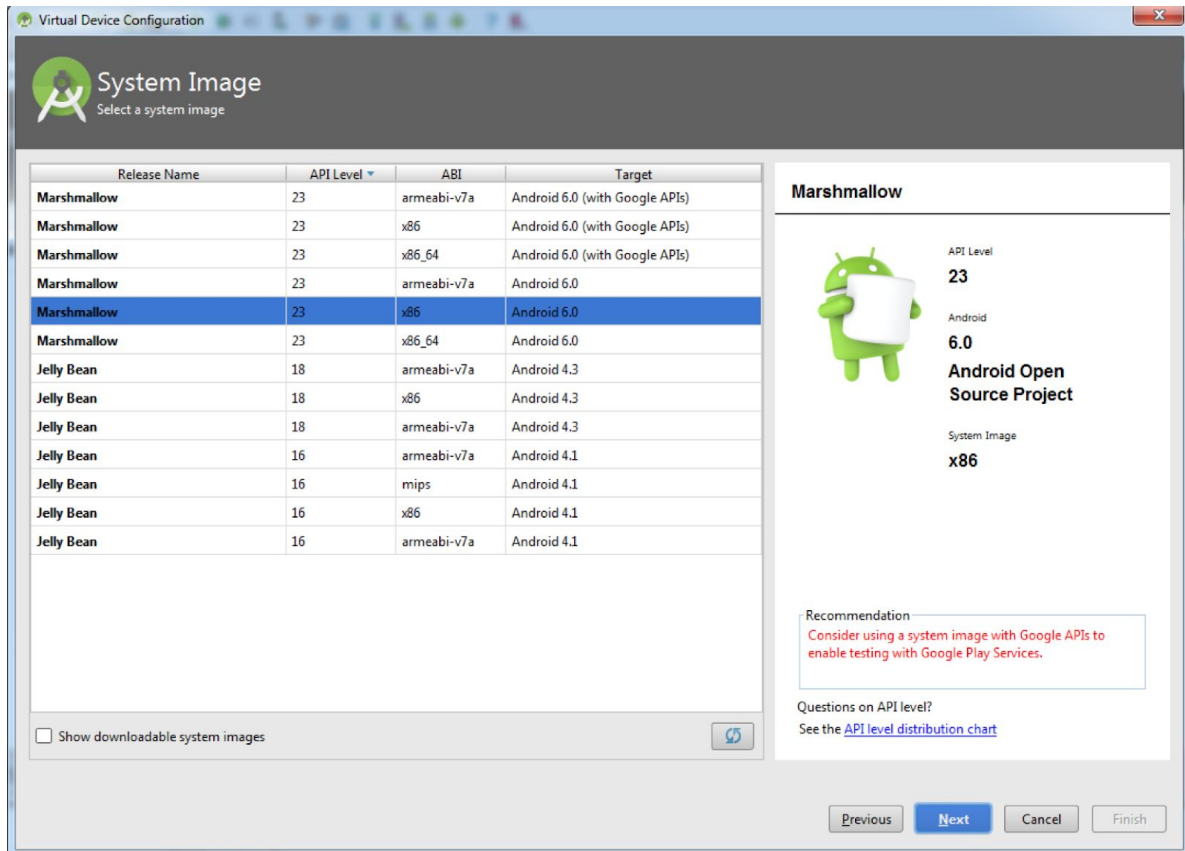
```

- Se deberá crear un nuevo Emulador de Android para la ejecución de la App, presionar el botón de “Android Virtual Device Manager”  en la barra de herramientas. Se iniciará el AVD Manager del SDK de Android:

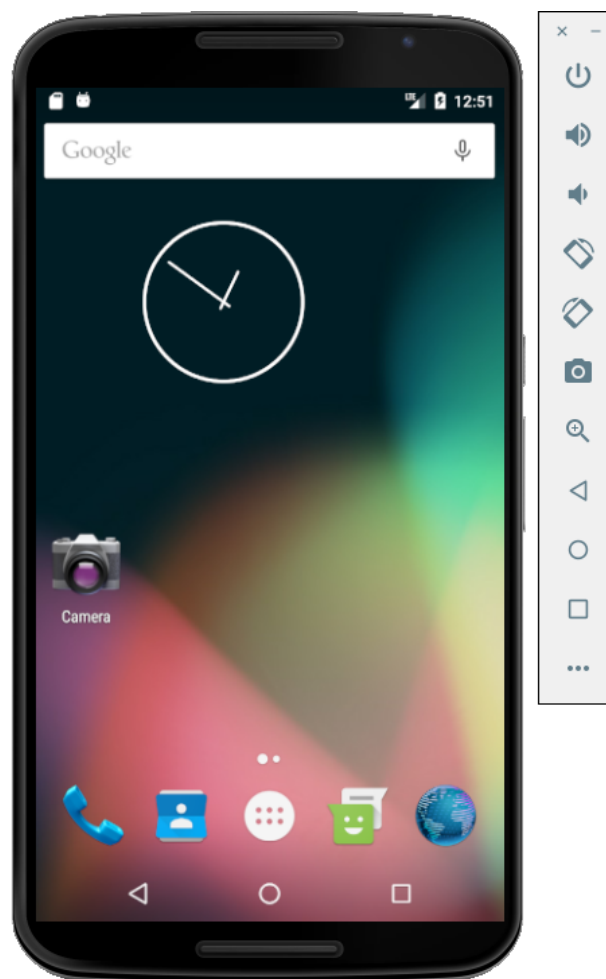
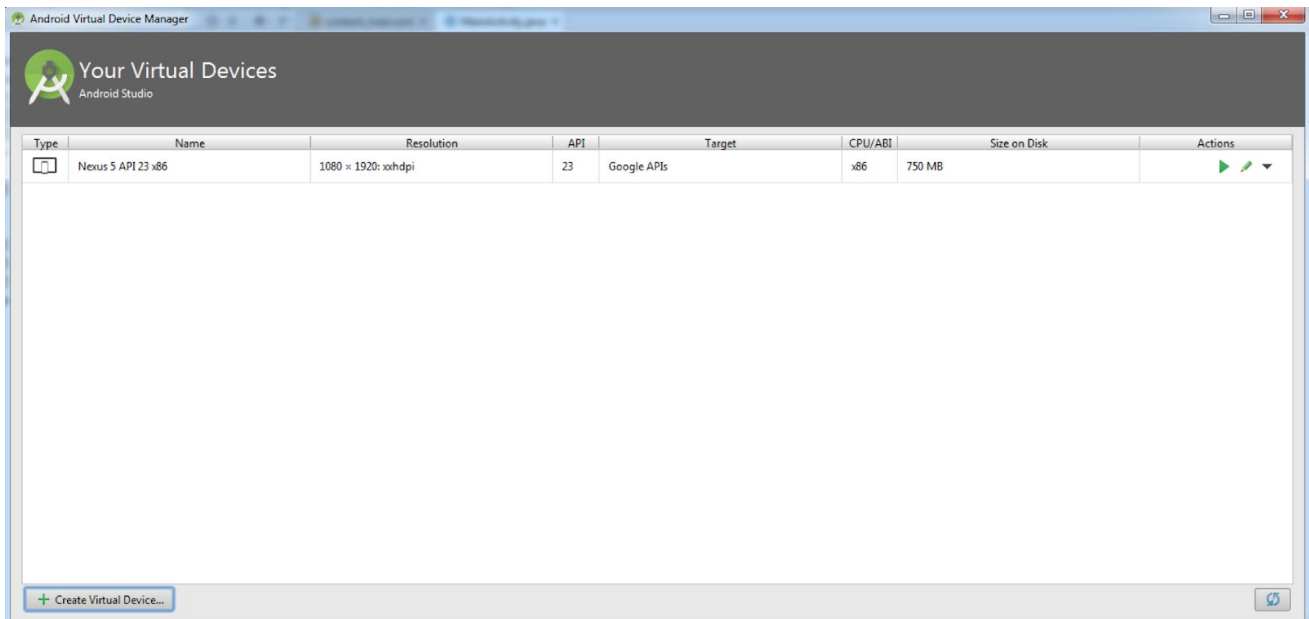



- Crear en “Create Virtual Device” un nuevo Emulador con la siguiente configuración, hasta la última pantalla presionar “Finish”:

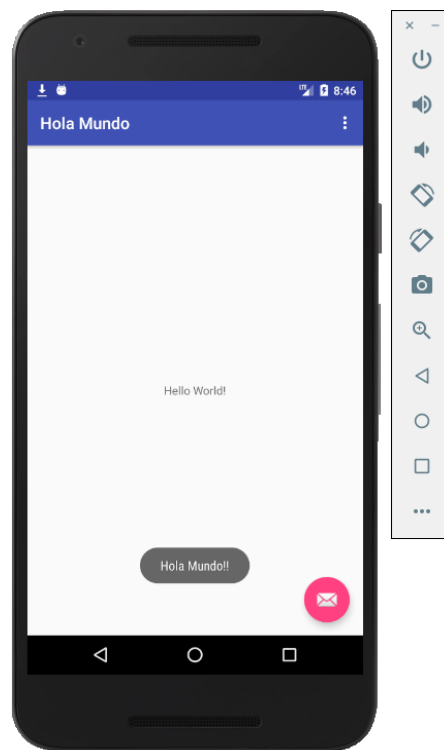
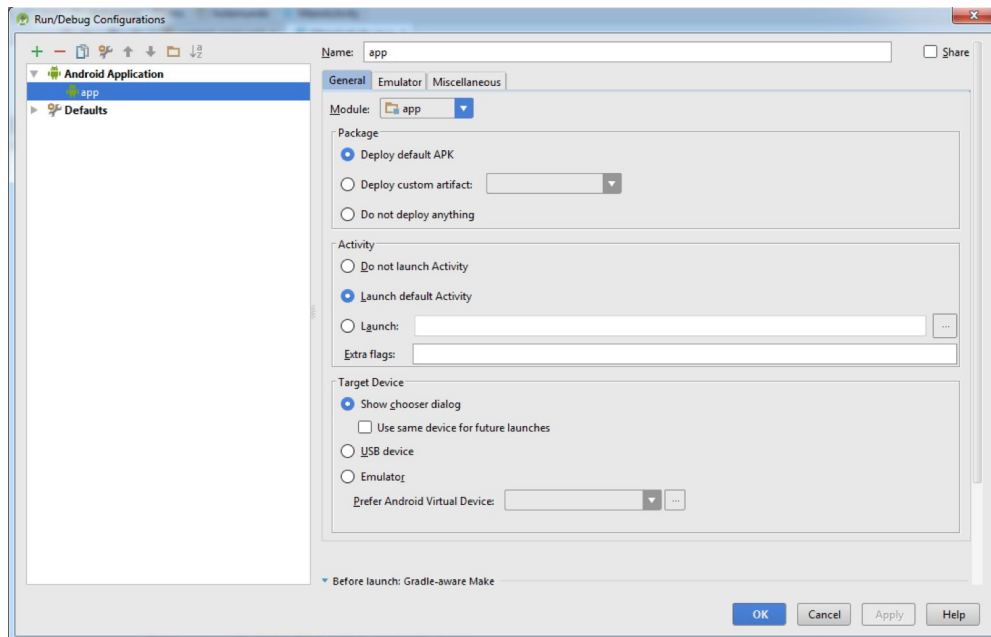




- Seleccionar el Emulador creado en el AVD Manager y presionar “Start”, esto iniciará la ejecución del Emulador, habrá que esperar a que se inicie en la pantalla de Bloqueo y desbloquearlo para tener la pantalla principal:



- Cerrar el AVD Manager y ejecutar el Proyecto presionando el botón  de la barra de herramientas, la App se ejecutará en el Emulador:





LAB01: NAVEGACIÓN Y GUI

Objetivos

- Manejo de Fragments en TabGroups.
- Generación de Layouts, Clases e Intents para manejo de Activities.
- Configurar el menú de opciones y su acción.

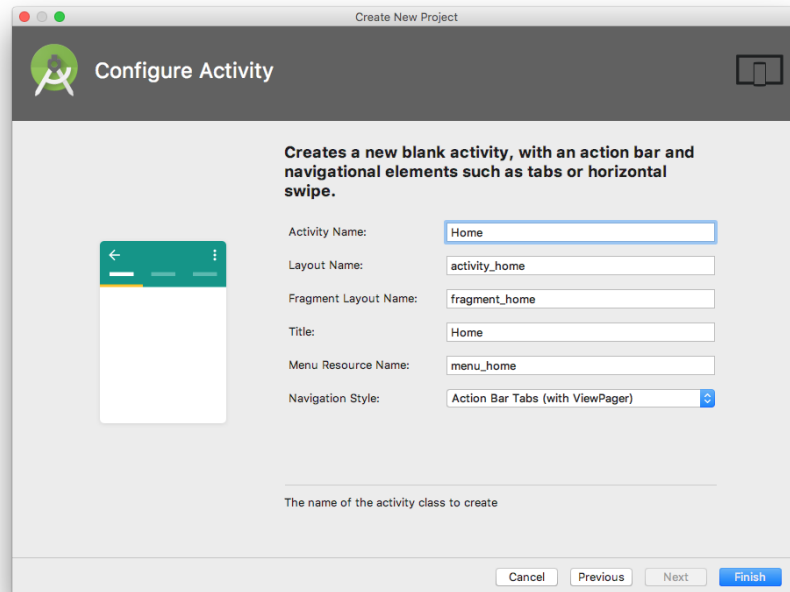
Resumen

Para este laboratorio se requiere crear una App que contenga 3 TabsGroups, el primero mostrará una lista de los fugitivos que no han sido atrapados, el segundo mostrará una lista de los que ya se han atrapado y el tercero será un “Acerca de” la App con el nombre del desarrollador, el botón Menú del teléfono desplegará una opción para agregar un nuevo fugitivo a la lista abriendo un formulario de captura con el que se indica el nombre de éste, al hacer clic sobre un fugitivo o capturado de la lista, se deberá mostrar el detalle del fugitivo y la opción para capturar, si aún anda suelto o para eliminarlo de la lista. En este primer laboratorio solo el despliegue de los Activity y la navegación entre ellos es requerido, la lista de fugitivos puede ser configurada en HardCode y las acciones para Agregar, Eliminar o Capturar no contendrán funcionalidad aún.

Punto de Partida

Generar un nuevo proyecto con los siguientes pasos:

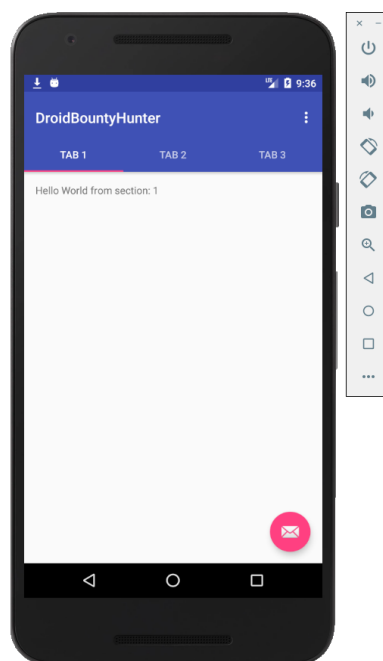
- Crear un nuevo proyecto con el nombre “DroidBountyHunter” con versión mínima 19 (Seleccionar solamente para la aplicación para “Phone and Tablet”).
- Indicar el package como “**training.com**”.
- Asegurate que la casilla “Include Kotlin support” esté seleccionada.
- Selecciona un Tabbed Activity.
- Indicar el Activity Name “Home”.
- Indicar el título como “droidBountyHunter”.
- Indicar el Navigation Style como: ActionBar Tabs (with ViewPager)



Ejecución del Laboratorio

Como primer paso se deberá asegurar que se tiene un Emulador apropiado para ejecutar el Laboratorio y que se tiene el proyecto “droidBountyHunter” con el package “edu.training.droidbountyhunter”, es importante la correspondencia de package pues los archivos en el set de Laboratorios electrónico están sobre ese Package, en caso de tener un Package diferente, deberá actualizarse en cada archivo que se incluya.

Para asegurar que estamos listos para iniciar el laboratorio debemos iniciar el emulador (tardará algunos minutos) y ejecutar el proyecto sobre el emulador esperando un resultado similar al siguiente:



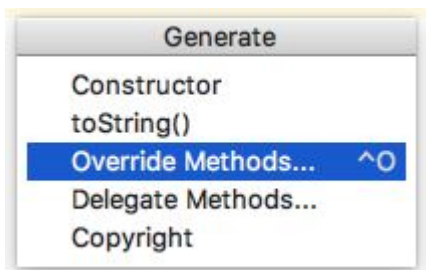
Se deberá editar el archivo de “string.xml” en la carpeta “res/values/” para incluir los strings necesarios para los botones y títulos, el XML resultante deberá lucir así: (Solo colocar lo que se encuentran resaltados).

```
<resources>
    <string name="app_name">DroidBountyHunter</string>
    <string name="tab_text_1">Tab 1</string>
    <string name="tab_text_2">Tab 2</string>
    <string name="tab_text_3">Tab 3</string>
    <string name="action_settings">Settings</string>
    <string name="section_format">Hello World from section: %1$d</string>
    <string name="titulo_capturados">Capturados</string>
    <string name="titulo_fugitivos">Fugitivos</string>
    <string name="titulo_acerca_de">Acerca de...</string>
    <string name="menu_agregar">Agregar</string>
    <string name="boton_guardar">Guardar</string>
    <string name="boton_eliminar">Eliminar</string>
    <string name="boton_capturar">Capturar</string>
    <string name="campo_nombre">Nombre del Fugitivo</string>
    <string name="etiqueta_desarrollo">Desarrollado por:</string>
    <string name="nombre_desarrollador">Giovani González</string>
    <string name="etiqueta_año">2018</string>
    <string name="etiqueta_empresa">D.W. Training S.C.</string>
    <string name="etiqueta_mensaje">Mensaje...</string>
</resources>
```

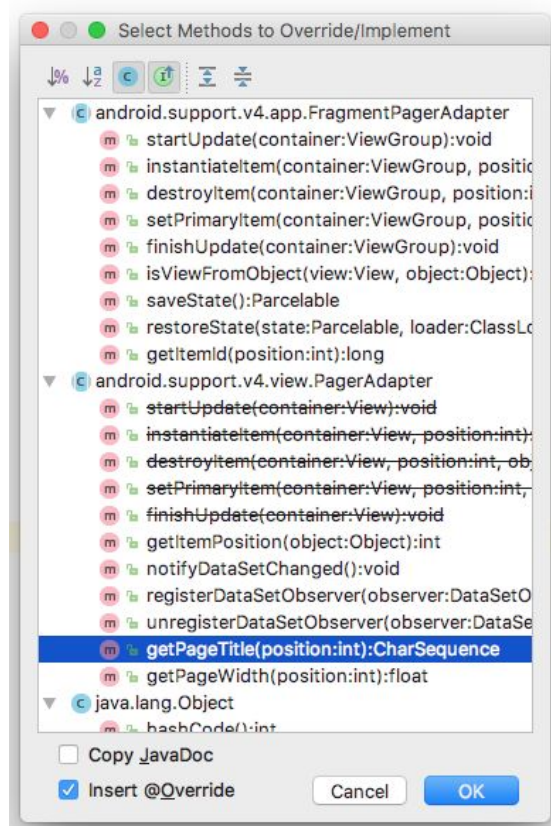
Al modificar los string, se deberán ajustar los Títulos correspondientes a las referencias de los Tabs que sean necesarios, lo anterior en la clase “Home.kt” dentro de la clase interna “SectionsPagerAdapter” en el método “getPageTitle()”.

Si no se encuentra el método “getPageTitle” dentro de la clase “SectionsPagerAdapter” entonces habrá que sobre escribirla manualmente, para esto necesitamos:

1. Presionar (Alt + Insertar [solo en Windows]) o (Alt + N [para Mac]) el cursor debe estar dentro de la clase “SectionsPagerAdapter”.
2. Seleccionar la opción de “Override methods” del menú desplegable.



3. De la lista seleccione el método “getPageTitle(int position)”



Una vez que tenemos localizado el método o que lo hayamos agregado, entonces pasamos a sobreescribirlo para que nos regrese los títulos de acuerdo a la posición en la que estemos, por lo que queda de la siguiente manera:

```
override fun getPageTitle(position: Int) = when (position) {
    0 -> getString(R.string.titulo_fugitivos).toUpperCase()
    1 -> getString(R.string.titulo_capturados).toUpperCase()
    else -> getString(R.string.titulo_acerca_de).toUpperCase()
}
```

Necesitaremos crear los layouts de los Fragments que utilizaremos para representar las listas de los fugitivos, así como el layout para representar cada elemento de la lista y el apartado de "acerca de" de nuestra aplicación, los layouts a crear serán los siguientes:

- fragment_acerca_de.xml
- fragment_list.xml
- item_fugitivo_list.xml

Se creará un layout para representar cada elemento de la lista (item_fugitivo_list.xml)

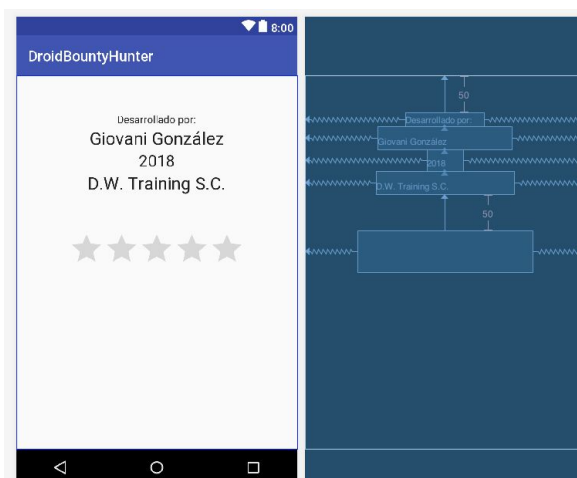
```
<?xml version="1.0" encoding="utf-8"?>
<TextView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/textoRenglon"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:padding="8dp"/>
```

El layout del Fragment para manejar los listados de Fugitivos capturados y libres contendrá el siguiente XML (layout_fragment_list.xml):

```
<?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">

    <ListView
        android:id="@+id/listaFugitivosCapturados"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        tools:listitem="@layout/item_fugitivo_list"/>
</android.support.constraint.ConstraintLayout>
```

El Fragment para mostrar el “Acerca de...” (fragment_acerca_de.xml) Puede contener diferentes elementos UI, lo sugerido en el fragment_acerca_de.xml contiene 4 labels (TextView) y un Rating View que más adelante servirá para el manejo de Properties de Android:



Los textos en las labels son fijos, el XML sería:

```
<?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">

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="50dp"
        android:text="@string/etiqueta_desarrollo"
        android:textAppearance="@style/TextAppearance.AppCompat.Body1"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/nombre_desarrollador"
        android:textAppearance="@style/TextAppearance.AppCompat.Headline"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/textView" />

    <TextView
        android:id="@+id/textView3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/etiqueta_año"
        android:textAppearance="@style/TextAppearance.AppCompat.Large"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/textView2" />

    <TextView
        android:id="@+id/textView4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/etiqueta_empresa"
        android:textAppearance="@style/TextAppearance.AppCompat.Headline"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/textView3" />
```

```

<RatingBar
    android:id="@+id/ratingBar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="50dp"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/textView4" />
</android.support.constraint.ConstraintLayout>

```

El siguiente paso será crear los layouts para los Activity para agregar y para ver el detalle de los fugitivos; XML del Layout para Agregar (activity_agregar.xml):

```

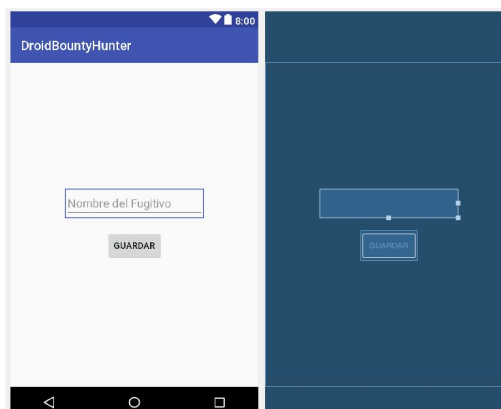
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center">

    <EditText
        android:id="@+id/nombreFugitivoTextView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:ems="10"
        android:hint="@string/campo_nombre"
        android:inputType="textPersonName"
        android:maxLength="200"
        android:maxLength="200"
        android:maxLength="200"
        android:maxLength="200"
        android:maxLength="200" />

    <Button
        android:id="@+id/botonGuardar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dp"
        android:text="@string/boton_guardar" />

</LinearLayout>

```



XML del layout para ver el detalle (activity_detalle.xml):

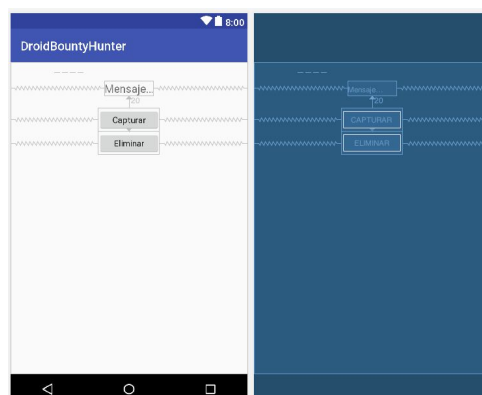
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <TextView
        android:id="@+id/etiquetaMensaje"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/etiqueta_mensaje"
        android:textAppearance="@style/Base.TextAppearance.AppCompat.Medium"
        android:layout_marginTop="30dp"
        android:layout_centerHorizontal="true"/>

    <Button
        android:id="@+id/botonCapturar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:minHeight="30dp"
        android:minWidth="100dp"
        android:text="@string/boton_capturar"
        android:textAppearance="@style/Base.TextAppearance.AppCompat.Body1"
        android:layout_marginTop="20dp"
        android:layout_below="@+id/etiquetaMensaje"
        android:layout_centerHorizontal="true"/>

    <Button
        android:id="@+id/botonEliminar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:minHeight="30dp"
        android:minWidth="100dp"
        android:text="@string/boton_eliminar"
        android:textAppearance="@style/Base.TextAppearance.AppCompat.Body1"
        android:layout_below="@+id/botonCapturar"
        android:layout_centerHorizontal="true"/>

</RelativeLayout>
```



Es importante haber configurado adecuadamente el archivo de Strings.xml para que las referencias a los “text” de los botones y etiquetas no generen errores.

Se debe generar el XML de opciones de menú con el nombre “menu_home.xml” en la carpeta “res/menu” con el siguiente XML (NOTA: se podrá editar el archivo existente de menú):

```
<menu 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"
    tools:context="com.training.droidbountyhunter.Home">
    <item
        android:id="@+id/menu_agregar"
        android:orderInCategory="100"
        android:title="@string/menu_agregar"
        app:showAsAction="never" />
</menu>
```

Una vez que ya tenemos los layouts de la app se deberán crear los Activities, utilizando una clase (archivo independiente) por cada uno, el Activity principal ya tiene su archivo “Home.kt” habrá que crear las clases “AgregarActivity.kt” y “DetalleActivity.kt”.

→ AgregarActivity.kt

```
package com.training.droidbountyhunter

import android.os.Bundle
import android.support.v7.app.AppCompatActivity

class AgregarActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_agregar)
    }
}
```

→ DetalleActivity.kt

```
package com.training.droidbountyhunter

import android.os.Bundle
import android.support.v7.app.AppCompatActivity
import kotlinx.android.synthetic.main.activity_detalle.*
```

Nota: Kotlinx - synthetic se encarga de las referencias de nuestra vista en este caso “activity_detalle.xml” y hace que tengamos acceso a todas las vistas de este layout.

```

class DetalleActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_detalle)
        // Se obtiene el nombre del fugitivo del intent y se usa como título
        title = intent.extras["titulo"] as CharSequence?
        // Se identifica si es Fugitivo o capturado para el mensaje...
        if (intent.extras["modo"] == 0) {
            etiquetaMensaje.text = "El fugitivo sigue suelto..."
        } else {
            etiquetaMensaje.text = "Atrapado!!!"
        }
    }
}

```

Paso seguido se deberá editar el archivo “strings.xml” para agregar los títulos de las actividades que declaramos en nuestro manifiesto:

```

<string name="titulo_activity_agregar">Nuevo fugitivo</string>
<string name="titulo_activity_detalle">Detalle Fugitivo</string>

```

Se deben registrar en el AndroidManifest.xml los Activities a utilizar en la App (dentro del tag <application>):

```

<activity android:name=".AgregarActivity"
    android:label="@string/titulo_activity_agregar"/>
<activity android:name=".DetalleActivity"
    android:label="@string/titulo_activity_detalle"/>

```

El siguiente paso será crear una clase que extiende de Fragment llamado “ListFragment.kt” (es recomendable crear un nuevo paquete llamado “fragments”). En “Home.kt” ya contiene un Fragment definido como “DummySectionFragment” o “PlaceholderFragment”, se removerá por completo y se hará referencia a la clase “ListFragment.kt”:

```

package com.training.fragments

import android.content.Intent
import android.os.Bundle
import android.support.v4.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.ArrayAdapter
import android.widget.TextView
import com.training.droidbountyhunter.DetalleActivity
import com.training.droidbountyhunter.R
import kotlinx.android.synthetic.main.fragment_list.*

```

```

const val SECTION_NUMBER : String = "section_number"

class ListFragment : Fragment() {
    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,
savedInstanceState: Bundle?): View? {
        // Se hace referencia al Fragment generado por XML en los Layouts y
        // se instancia en una View...
        return inflater.inflate(R.layout.fragment_list,container,false)
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        val modo = arguments!![SECTION_NUMBER] as Int
        // Datos dummy para la lista
        val dummyData = listOf(
            "Armando Olmos",
            "Guillermo Ortega",
            "Carlos Martinez",
            "Israel Ramirez",
            "Karen Muñoz",
            "Alejandro Rincon")
        val adaptador = ArrayAdapter<String>(context,R.layout.item_fugitivo_list,dummyData)
        listaFugitivosCapturados.adapter = adaptador
        listaFugitivosCapturados.setOnItemClickListener { adapterView, view,
            position, id ->
            val intent = Intent(context, DetalleActividad::class.java)
            intent.putExtra("titulo", (view as TextView).text)
            intent.putExtra("modo", modo)
            startActivity(intent)
        }
    }
}

```

Luego crearemos el fragmento que ocuparemos para el “Acerca de” llamado “AcercaDeFragment.kt”:

```

package com.training.fragments

import android.os.Bundle
import android.support.v4.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import com.training.droidbountyhunter.R
import kotlinx.android.synthetic.main.fragment_acerca_de.*

class AcercaDeFragment : Fragment(){
    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,
savedInstanceState: Bundle?): View? {
        // Se hace referencia al Fragment generado por XML en los Layouts y
        // se instancia en una View...
        return inflater.inflate(R.layout.fragment_acerca_de,container,false)
    }
}

```

```

override fun onCreateView(view: View, savedInstanceState: Bundle?) {
    var rating = "0.0" // Variable para lectura del rating guardado en Properties
    if (System.getProperty("rating") != null) {
        rating = System.getProperty("rating")
    }
    if (rating.isEmpty()) {
        rating = "0.0"
    }
    ratingBar.rating = rating.toFloat()
    ratingBar.setOnRatingBarChangeListener { ratingBar, rating, fromUser ->
        System.setProperty("rating", rating.toString())
        ratingBar.rating = rating
    }
}
}

```

Se deberá actualizar la referencia el Menú de opciones, utilizando el XML que se importó, según se nombró “menu_home.xml” o “home.xml”, asegurarse que el nombre del archivo esté bien en el método llamado “onCreateOptionsMenu” de “Home.kt”:

```

override fun onCreateOptionsMenu(menu: Menu): Boolean {
    // Inflate the menu; this adds items to the action bar if it is present.
    menuInflater.inflate(R.menu.menu_home, menu)
    return true
}

```

Se deberá codificar el Listener del Menú para abrir el Activity correspondiente al presionar la opción:

```

override fun onOptionsItemSelected(item: MenuItem): Boolean {
    if (item.itemId == R.id.menu_agregar) {
        val intent = Intent(this, AgregarActivity::class.java)
        startActivityForResult(intent, 0)
    }

    return super.onOptionsItemSelected(item)
}

```

Agregar al Activity (propiedad de la clase Home.kt) una variable que almacene los Fragments a utilizar:

```

private var fragments: ArrayList<Fragment> = ArrayList()

```

Se deberá adecuar el SectionPagerAdapter en el método getItem() para abrir los Fragments correctos evaluando la posición del Tab que se está seleccionando:

```

override fun getItem(position: Int): Fragment {
    if (fragments.size < 3){ // Si no contiene los 3 fragments los agregará
        if (position < 2){
            fragments.add(position, ListFragment())
            val arguments = Bundle()
            arguments.putInt(SECTION_NUMBER, position)
            fragments[position].arguments = arguments
        }else{
            fragments.add(position, AcercaDeFragment())
        }
    }
    return fragments[position]
}

```

El siguiente paso será colocarnos en el método onCreate() de la clase "Home" para colocar el Intent hacia el Activity "AgregarActivity" sobre el botón flotante.

```

fab.setOnClickListener { view ->
    val intent = Intent(this, AgregarActivity::class.java)
    startActivityForResult(intent, 0)
}

```

Asegurate que las tabs o TabLayout esté ligada con el ViewPager, para eso se ocupa el método ".setupWithViewPager(viewPager)", si no encuentras esta llamada en el onCreate de "Home.kt" entonces agregala:

```

tabs.setupWithViewPager(container)

```

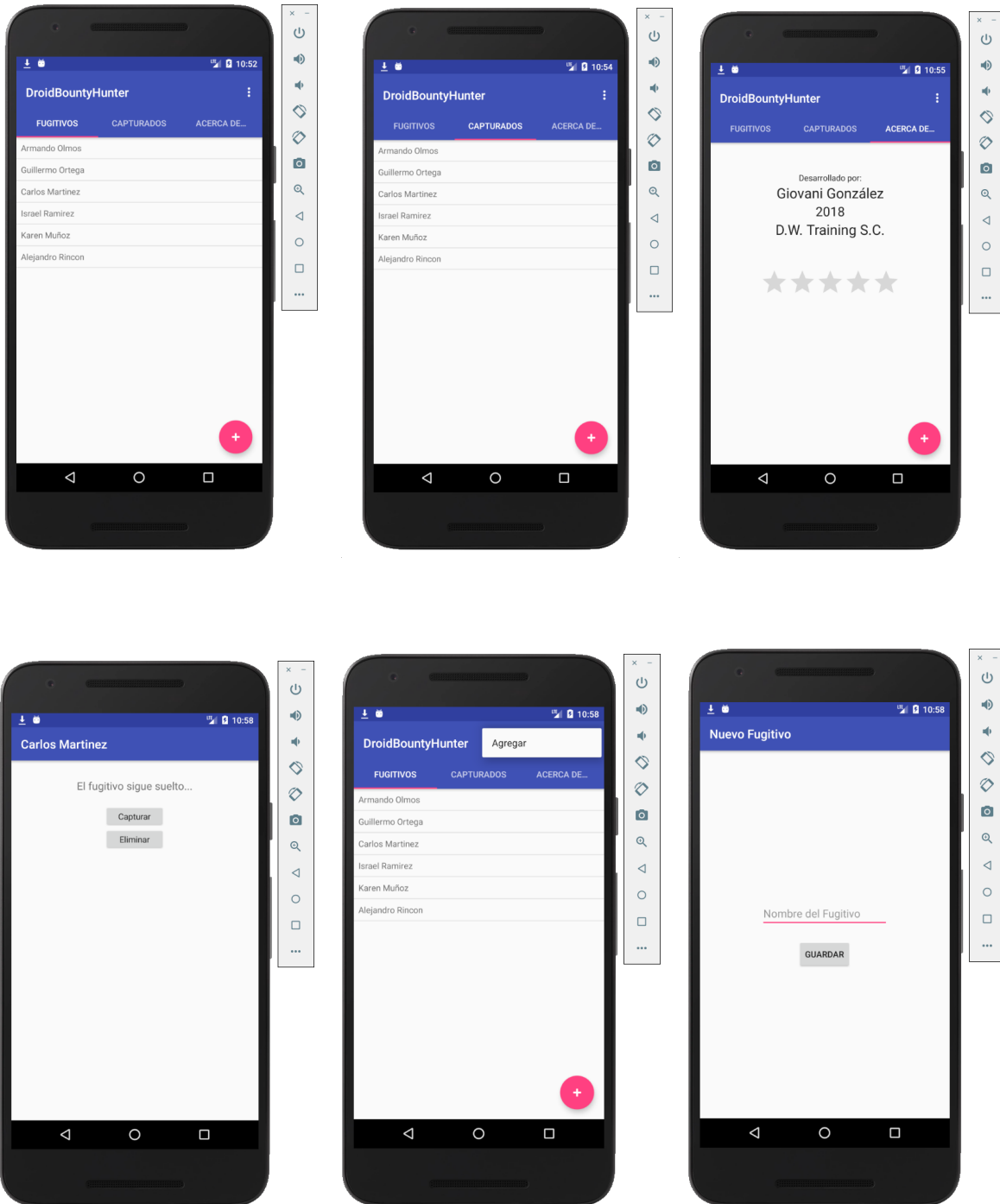
Posteriormente nos colocaremos en el archivo de layout "activity_home.xml" dentro del cual cambiaremos la propiedad "src" o "srcCompat" del componente "FloatingActionButton", para que el ícono tenga el simbolo "+" se toma de los drawables del sistema Android "ic_input_add".

```

<android.support.design.widget.FloatingActionButton
    android:id="@+id/fab"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="end|bottom"
    android:layout_margin="@dimen/fab_margin"
    app:srcCompat="@android:drawable/ic_input_add"
    android:tint="@android:color/white"/>

```


Finalmente ejecutar la aplicación para navegar en los Activities y Fragments (el diseño podría variar según la versión del android studio, debido al manejo de las plantillas):





LAB02: PERSISTENCIA DE DATOS

Objetivos

- Construir una Clase para el manejo de conexiones al SQLite.
- Actualizar la App para el manejo de fugitivos en el SQLite (Agregar, Eliminar, Capturar y Consultar).

Resumen

En este laboratorio se utilizarán las interfaces gráficas del laboratorio anterior para almacenar información en SQLite, consumirla para cargar las listas, agregar, eliminar y actualizar para capturar.

Punto de Partida

Se utilizará el proyecto existente para complementarse con los archivos que se generará en el laboratorio número 2.

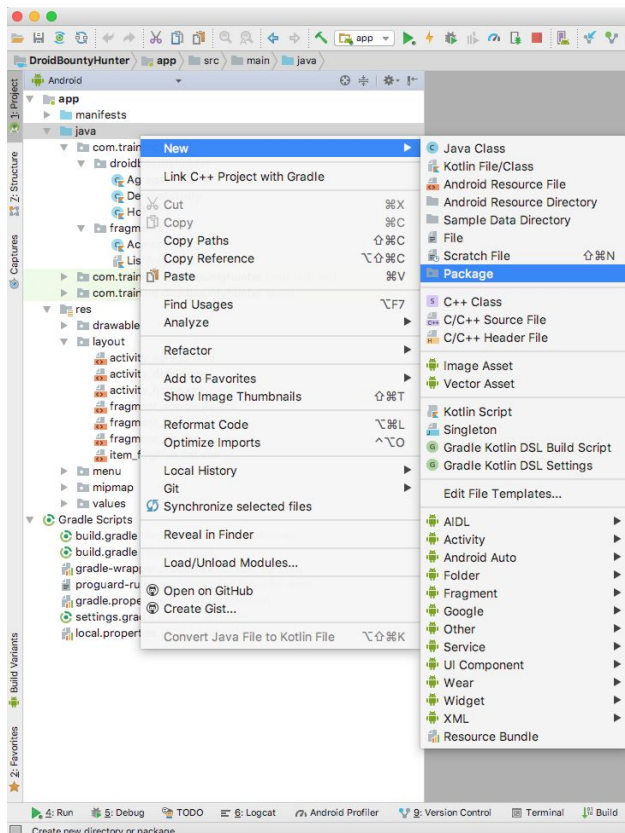
Ejecución del Laboratorio

Para comenzar crearemos una nueva clase que se llamará “DatabaseBountyHunter” para el manejo de conexiones, consultas, inserciones, etc. en la base de datos.

Paso seguido crearemos una clase **estática interna** llamada “DBhelper” que extenderá de SQLiteOpenHelper sobrescribir los métodos onCreate, onUpgrade, mismos que nos servirán para la manipulación de la estructura de nuestra base de datos y sentencias SQL. Dentro de la clase interna DBhelper declararemos las siguientes propiedades para el manejo de la base de datos en la clase.

Antes de poder iniciar con la implementación de la base de datos deberemos crear los modelos de datos que usaremos para las acciones de inserción, actualización y borrado.

1. Primero debemos de seleccionar el folder “java” presionar clic derecho y en el menú desplegable y seleccionar la opción “New > Package”.



2. Crea un paquete nuevo llamado “com.training.models”.
3. Dentro de ese paquete crearemos una clase más llamada “Modelos.kt” que será la clase que utilizaremos para el parseo y manejo de información.

```
data class Fugitivo (
    val id: Int = 0,
    val name: String = "",
    val status: Int = 0)
```

Ahora pasamos a crear las clases que nos ayudarán a la implementación de SQLite en Android, para ello debemos:

1. Crear un paquete más llamado “data”.
2. Crear dentro del paquete “data” una clase nueva llamada “DatabaseBountyHunter”
3. Dentro de esa clase definiremos una variable que llamaremos TAG para poder hacer tracking de los movimientos de la base de datos
4. Declaración del nombre de la base de datos
5. Declaración de la versión
6. Declaración de los nombres de los campos
7. Creación de la tabla “fugitivos” (sentencia SQL).

```

package com.training.data

import android.content.Context
import android.database.sqlite.SQLiteDatabase
import android.database.sqlite.SQLiteOpenHelper
import android.util.Log

/** ----- Nombre de Base de Datos ----- */
const val DATABASE_NAME = "DroidBountyHunterDatabase"
/** ----- Versión de Base de Datos ----- */
const val VERSION = 1
/** ----- Tablas y Campos ----- */
const val TABLE_NAME_FUGITIVOS = "fugitivos"
const val COLUMN_NAME_ID = "id"
const val COLUMN_NAME_NAME = "name"
const val COLUMN_NAME_STATUS = "status"

class DatabaseBountyHunter {
    private val TAG: String = DatabaseBountyHunter::class.java.simpleName
    /** ----- Declaración de Tablas ----- */
    private val TFugitivos = "CREATE TABLE " + TABLE_NAME_FUGITIVOS + " (" +
        COLUMN_NAME_ID + " INTEGER PRIMARY KEY NOT NULL, " +
        COLUMN_NAME_NAME + " TEXT NOT NULL, " +
        COLUMN_NAME_STATUS + " INTEGER, " +
        "UNIQUE (" + COLUMN_NAME_NAME + ") ON CONFLICT REPLACE);"
}

```

Declaramos la clase interna llamada DBHelper que extiende de SQLiteOpenHelper e implementamos los métodos de onCreate y onUpgrade.

```

inner class DBHelper(context: Context) : SQLiteOpenHelper(context, DATABASE_NAME,
    null, VERSION) {
    override fun onCreate(db: SQLiteDatabase?) {}

    override fun onUpgrade(db: SQLiteDatabase?, oldVersion: Int, newVersion: Int)
    {}
}

```

Declaramos las variables que ocuparemos para la comunicación con nuestra interfaz de base de datos dentro de la clase DatabaseBountyHunter.

```

/** ----- Variables y Helpers ----- */
private var helper: DBHelper? = null
private var database: SQLiteDatabase? = null

```

Sobreescribimos el método onCreate de la clase DBHelper en el que realizaremos la creación de la base de la tabla llamada “fugitivos”.

```
override fun onCreate(db: SQLiteDatabase?) {
    Log.d(TAG, "Creación de la base de datos")
    db!!.execSQL(TFugitivos)
}
```

Así mismo sobreescribimos el método onUpgrade, el cual se ejecutará cada que se detecte un cambio de versión en la base de datos, mismo que servirá para realizar los ajustes necesarios a la base de datos cuando cambie de versión y se requiera modificar su estructura. Por el momento en este método se eliminará la tabla “fugitivos” para llamar al método onCreate y crear la nueva estructura de la tabla.

```
override fun onUpgrade(db: SQLiteDatabase?, oldVersion: Int, newVersion: Int) {
    Log.w(TAG, "Actualización de la BDD de la versión " + oldVersion + "a la " +
        newVersion + ", de la que se destruirá la información anterior")
    // Destruir BDD anterior y crearla nuevamente las tablas actualizadas
    db!!.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME_FUGITIVOS)
    // Re-creando nuevamente la BDD actualizada
    onCreate(db)
}
```

Una vez finalizada la creación y definición de la clase interna DBHelper crearemos los métodos necesarios para la manipulación de la información de nuestra base de datos en la clase DatabaseBountyHunter.

Ahora crearemos el constructor de nuestra clase DatabaseBountyHunter:

```
class DatabaseBountyHunter(val context: Context) {
```

El paso siguiente será crear dos métodos para la apertura y cierre de la conexión a la base de datos:

```
fun open(): DatabaseBountyHunter {
    helper = DBHelper(context)
    database = helper!!.writableDatabase
    return this
}

fun close() {
    helper!!.close()
    database!!.close()
}
```

Posteriormente crearemos el método que abrirá la base de datos en modo escritura para sentencias update e insert así como queries.

```
fun querySQL(sql: String, selectionArgs: Array<String>): Cursor{
    open()
    val regreso = database!!.rawQuery(sql,selectionArgs)
    return regreso
}
```

El siguiente paso será crear un método para la eliminación de los fugitivos de la base de datos en el cual se recibirá el id del fugitivo a borrar.

```
fun borrarFugitivo(fugitivo: Fugitivo){
    open()
    database!!.delete(TABLE_NAME_FUGITIVOS, COLUMN_NAME_ID + "=?",
        arrayOf(fugitivo.id.toString()))
    close()
}
```

Crearemos un método para actualización del fugitivo, en el cual recibirá como parámetros un objeto fugitivo.

```
fun actualizarFugitivo(fugitivo: Fugitivo){
    open()
    val values = ContentValues()
    values.put(COLUMN_NAME_NAME, fugitivo.name)
    values.put(COLUMN_NAME_STATUS, fugitivo.status)
    database!!.update(TABLE_NAME_FUGITIVOS,values, COLUMN_NAME_ID + "=?",
        arrayOf(fugitivo.id.toString()))
    close()
}
```

Ahora realizaremos un método para la inserción fugitivos.

```
fun insertarFugitivo(fugitivo: Fugitivo){
    val values = ContentValues()
    values.put(COLUMN_NAME_NAME, fugitivo.name)
    values.put(COLUMN_NAME_STATUS, fugitivo.status)
    open()
    database!!.insert(TABLE_NAME_FUGITIVOS, null,values)
    close()
}
```

El siguiente método que crearemos regresará los fugitivos según el estatus recibido en la función, retornando un ArrayList de objetos Fugitivo con la tabla de fugitivos contenida en el cursor que regresa la lista de la base de datos.

```

fun obtenerFugitivos(status: Int) : Array<Fugitivo> {
    var fugitivos: Array<Fugitivo> = arrayOf()
    val dataCursor = querySQL("SELECT * FROM " + TABLE_NAME_FUGITIVOS +
        " WHERE " + COLUMN_NAME_STATUS + "= ? ORDER BY " + COLUMN_NAME_NAME,
        arrayOf(status.toString()))
    if (dataCursor.count > 0) {
        fugitivos = generateSequence {
            if (dataCursor.moveToNext()) dataCursor else null
        }.map {
            val name = it.getString(it.getColumnIndex(COLUMN_NAME_NAME))
            val statusFugitivo = it.getInt(it.getColumnIndex(COLUMN_NAME_STATUS))
            val id = it.getInt(it.getColumnIndex(COLUMN_NAME_ID))
            return@map Fugitivo(id, name, statusFugitivo)
        }.toList().toTypedArray()
    }
    return fugitivos
}

```

En la clase “ListFragment” se agregará también un método para actualización de los listados, creando una instancia del DatabaseBountyHunter llamando el método obtenerFugitivos de la base de datos:

```

private fun actualizarDatos(listView: ListView?, modo: Int) {
    val database = DatabaseBountyHunter(context!!)
    val fugitivos = database.obtenerFugitivos(modo)
    if (fugitivos.isNotEmpty()) {
        val values = ArrayList<String>()
        fugitivos.mapTo(values) { it.name }
        val adaptador = ArrayAdapter<String>(context,
            R.layout.item_fugitivo_list, values)
        listView!!.adapter = adaptador
        listView.tag = fugitivos
    }
}

```

Se localiza la sección del Fragment (onViewCreated) donde se llena la Lista y se reemplaza por la llamada al método de actualizarDatos:

```

override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    val modo = arguments!![SECTION_NUMBER] as Int
    actualizarDatos(listaFugitivosCapturados, modo)
    listaFugitivosCapturados.setOnItemClickListener { adapterView, view,
        position, id ->
        val intent = Intent(context, DetalleActivity::class.java)
        intent.putExtra("titulo", (view as TextView).text)
        intent.putExtra("modo", modo)
        startActivity(intent)
    }
}

```

Para poder compartir un objeto completo dentro de un bundle ocuparemos hacer el objeto “Parcelable” en este case el objeto “Fugitivo”, por lo cual ya modificado, la clase Fugitivo queda:

```
data class Fugitivo (
    val id: Int = 0,
    var name: String = "",
    var status: Int = 0) : Parcelable {
    constructor(parcel: Parcel) : this(
        parcel.readInt(),
        parcel.readString(),
        parcel.readInt())

    override fun writeToParcel(parcel: Parcel, flags: Int) {
        parcel.writeInt(id)
        parcel.writeString(name)
        parcel.writeInt(status)
    }

    override fun describeContents() = 0

    companion object CREATOR : Parcelable.Creator<Fugitivo> {
        override fun createFromParcel(parcel: Parcel): Fugitivo {
            return Fugitivo(parcel)
        }

        override fun newArray(size: Int): Array<Fugitivo?> {
            return arrayOfNulls(size)
        }
    }
}
```

Actualizar el Listener del click sobre elementos de la lista para enviar al Activity Detalle el fugitivo, también se debe cambiar el startActivity por startActivityForResult:

```
override fun onCreateView(view: View, savedInstanceState: Bundle?) {
    val modo = arguments!![SECTION_NUMBER] as Int
    actualizarDatos(listaFugitivosCapturados, modo)
    listaFugitivosCapturados.setOnItemClickListener { adapterView, view,
        position, id ->
        val intent = Intent(context, DetalleActivity::class.java)
        val fugitivos = listaFugitivosCapturados.tag as Array<Fugitivo>
        intent.putExtra("fugitivo", fugitivos[position])
        startActivityForResult(intent, 0)
    }
}
```


Por lo tanto habrá que actualizar DetalleActivity también para poder leer el fugitivo.

```
class DetalleActivity : AppCompatActivity() {

    var fugitivo: Fugitivo? = null

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_detalle)
        fugitivo = intent.extras["fugitivo"] as Fugitivo
        // Se obtiene el nombre del fugitivo del objeto fugitivo
        // y se usa como título
        title = fugitivo!!.name
        // Se identifica si es Fugitivo o capturado para el mensaje...
        if (fugitivo!!.status == 0) {
            etiquetaMensaje.text = "El fugitivo sigue suelto..."
        } else {
            etiquetaMensaje.text = "Atrapado!!!"
        }
    }
}
```

Agregar al Activity Home un método para actualizar los listados de los Fragments, donde container es el viewPager:

```
fun actualizarListas(index: Int) {
    container.adapter = mSectionsPagerAdapter
    container.currentItem = index
}
```

Se deberá sobrescribir el método para escuchar el resultado de un Activity y actualizar los listados:

```
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)
    actualizarListas(resultCode)
}
```

Identificar en el Layout para el Activity de Agregar el XML del botón para agregar el evento al momento de hacer click sobre él, indicando el método guardarFugitivoPresionado:

```
<Button
    android:id="@+id/botonGuardar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="20dp"
    android:text="@string/boton_guardar"
    android:onClick="guardarFugitivoPresionado"/>
```

Y se agrega el método en la clase del Activity de AgregarActivity.kt para utilizar el DatabaseBountyHunter y su método insertarFugitivo:

```
fun guardarFugitivoPresionado(view: View) {
    val nombre = nombreFugitivoTextView.text.toString()
    if (nombre.isNotEmpty()) {
        val database = DatabaseBountyHunter(this)
        database.insertarFugitivo(Fugitivo(0,nombre,0))
        setResult(0)
        finish()
    } else {
        AlertDialog.Builder(this)
            .setTitle("Alerta")
            .setMessage("Favor de capturar el nombre del fugitivo.")
            .show()
    }
}
```

Identificar el Layout del Activity de Detalle y agregar a los botones las acciones correspondientes a “capturarFugitivoPresionado” y “eliminarFugitivoPresionado”:

```
<Button
    android:id="@+id/botonCapturar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:minHeight="30dp"
    android:minWidth="100dp"
    android:text="@string/boton_capturar"
    android:textAppearance="@style/Base.TextAppearance.AppCompat.Body1"
    android:layout_marginTop="20dp"
    android:layout_below="@+id/etiquetaMensaje"
    android:layout_centerHorizontal="true"
    android:onClick="capturarFugitivoPresionado"/>
```

```
<Button
    android:id="@+id/botonEliminar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:minHeight="30dp"
    android:minWidth="100dp"
    android:text="@string/boton_eliminar"
    android:textAppearance="@style/Base.TextAppearance.AppCompat.Body1"
    android:layout_below="@+id/botonCapturar"
    android:layout_centerHorizontal="true"
    android:onClick="eliminarFugitivoPresionado"/>
```

Agregar los métodos en la clase del Activity DetalleActivity.java utilizando el DatabaseBountyHunter y sus respectivos métodos para actualizar el estatus o eliminar el registro:

```
fun capturarFugitivoPresionado(view: View) {
    database = DatabaseBountyHunter(this)
    fugitivo!!.status = 1
    database!!.actualizarFugitivo(fugitivo!!)
    setResult(0)
    finish()
}
```

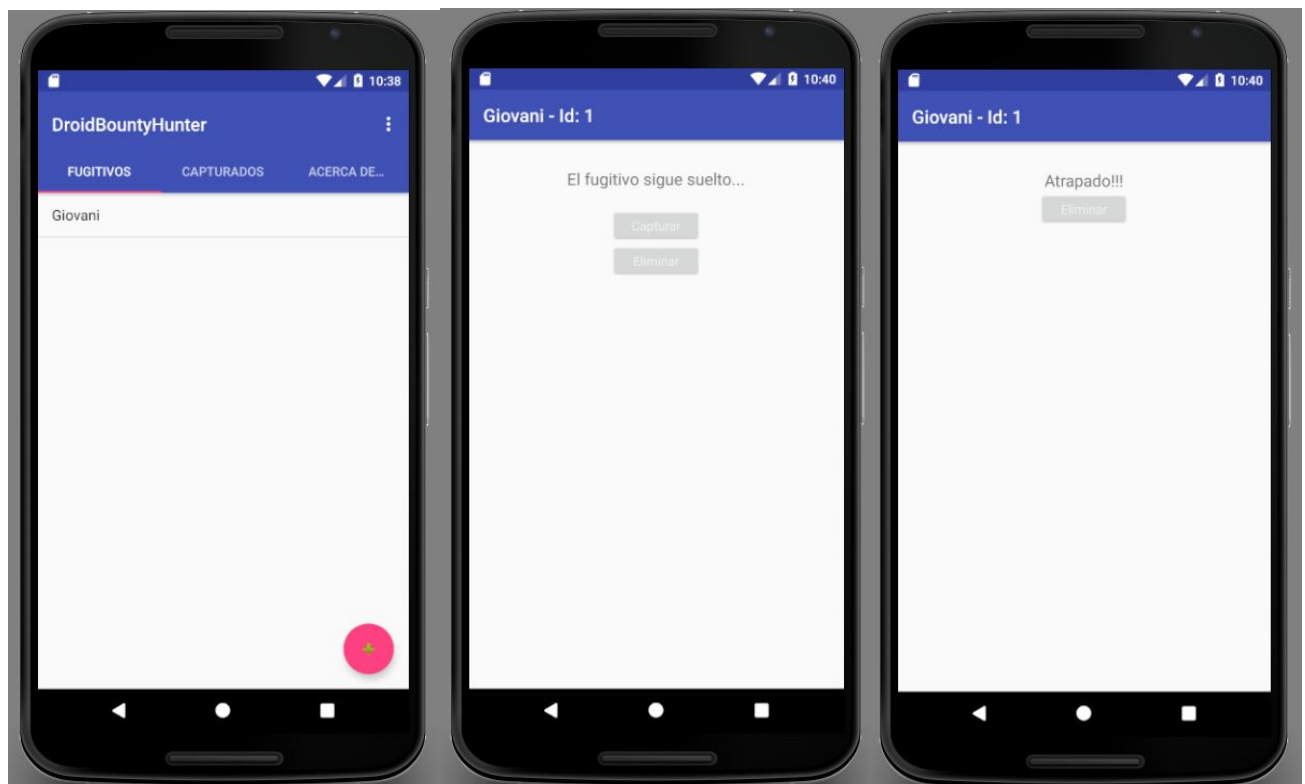
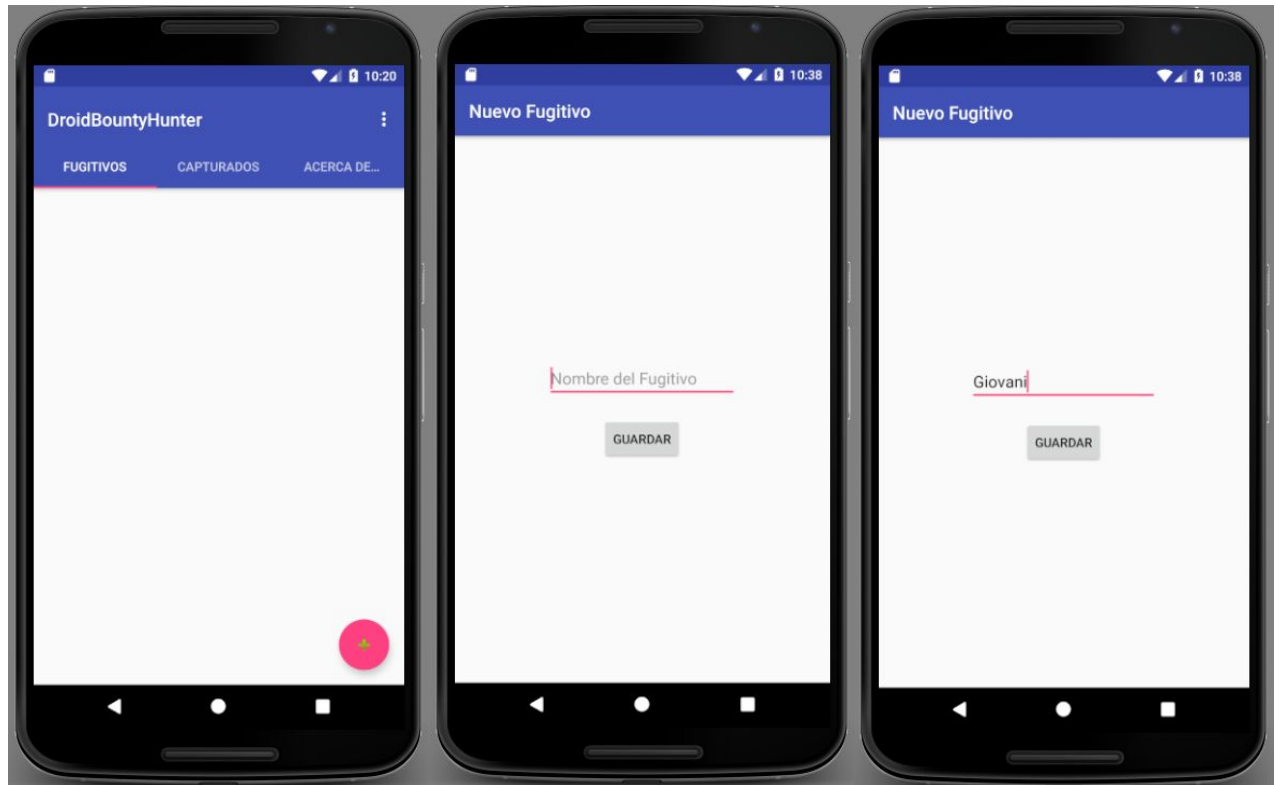
```
fun eliminarFugitivoPresionado(view: View) {
    database = DatabaseBountyHunter(this)
    database!!.borrarFugitivo(fugitivo!!)
    setResult(0)
    finish()
}
```

Se actualiza el título de “DetalleActivity” agregando el id del fugitivo, luego si el fugitivo ha sido capturado no debe poder tener el botón de “Capturar” por lo que lo ocultaremos.

```
var fugitivo: Fugitivo? = null
var database: DatabaseBountyHunter? = null

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_detalle)
    fugitivo = intent.extras["fugitivo"] as Fugitivo
    // Se obtiene el nombre del fugitivo del objeto fugitivo y se usa como título
    title = fugitivo!!.name + " - " + fugitivo!!.id
    // Se identifica si es Fugitivo o capturado para el mensaje...
    if (fugitivo!!.status == 0) {
        etiquetaMensaje.text = "El fugitivo sigue suelto..."
    } else {
        etiquetaMensaje.text = "Atrapado!!!"
        botonCapturar.visibility = GONE
    }
}
```

Finalmente se ejecuta la app mostrando la funcionalidad esperada con las listas vacías al inicio hasta que se utilice la funcionalidad de Agregar, Capturar y Eliminar (el diseño podría variar según la versión del android studio, debido al manejo de las plantillas).





LAB03: USO DE SERVICIOS WEB

Objetivos

- Utilizar un Servicio Web para pre-cargar la lista de Fugitivos a través del método GET.
- Utilizar un Servicio Web para indicar que el dispositivo ha atrapado a un fugitivo a través del método POST.

Resumen

En este laboratorio se utilizará el `URLConnection` con método GET para leer un listado de fugitivos en JSON y guardarlo en la base de datos. Cuando se realice la captura de un fugitivo, se informará al Servicio Web que el dispositivo ya ha capturado un nuevo fugitivo y el servicio regresará un mensaje que indicará el número de éstos que ya se hayan capturado, se deberá mostrar el mensaje al usuario. En caso de que la conexión con el Servicio Web no funcione adecuadamente se deberá mostrar un error.

Punto de Partida

Se utilizará el proyecto existente para complementarse con los archivos que se generarán siguiendo la ejecución del Laboratorio.

Ejecución del Laboratorio

Agregar al `AndroidManifest` el permiso para utilizar Internet en la App:

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

Se creará un nuevo paquete dentro de “com.training” llamado “network”, en el cual crearemos un archivo kotlin llamado “NetworkServices”, dentro escribiremos la interfaz “OnTaskListener”.

```
interface OnTaskListener{  
    fun tareaCompletada(respuesta: String)  
    fun tareaConError(codigo: Int, mensaje: String, error: String)  
}
```

El paso siguiente será crear una clase llamada `NetworkServices` que extenderá de `AsyncTask` para la implementación de una tarea asíncrona para los `WebServices`, de esta manera lograremos que la comunicación remota no interfiera con la interacción del usuario con la aplicación.

La clase recibirá como parámetros `<String, Void, Boolean>`, quedando la declaración de la clase de la siguiente manera:

```
class NetworkServices : AsyncTask<String, Void, Boolean>() {
    override fun doInBackground(vararg params: String?): Boolean {
        return true
    }
}
```

Ahora crearemos las propiedades necesarias de la clase para la comunicación de la aplicación con los `WebServices`:

```
private val TAG = NetworkServices::class.java.simpleName

private val endpoint_fugitivos =
"http://201.168.207.210/services/droidBHServices.svc/fugitivos"
private val endpoint_atrapados =
"http://201.168.207.210/services/droidBHServices.svc/atrapados"

private var JSONStr: String = ""
private var tipo: SERVICE_TYPE = SERVICE_TYPE.FUGITIVOS
private var codigo: Int = 0
private var mensaje: String = ""
private var error: String = ""
```

Creemos el constructor de la clase en el que pasaremos como parámetro el listener “`onTaskListener`” (se modifica la clase).

```
class NetworkServices (val listener: onTaskListener) : AsyncTask<String, Void, Boolean>() {
```

Se creará un enumerador más para el manejo de los tipos de llamadas que se realizarán, en este caso “Fugitivos” y “Atrapados”.

```
enum class SERVICE_TYPE{
    FUGITIVOS, ATRAPADOS
}
```

El siguiente paso será sobrescribir el método `doInBackground` en el que ocuparemos el `HttpURLConnection` para hacer la conexión, pero con el `InputStreamReader`, `BufferedReader` y `StringBuilder` podremos leer los datos que recibimos de esa conexión, así sea en caso de error

```

override fun doInBackground(vararg params: String?): Boolean {
    val esFugitivo = params[0]!!.equals("Fugitivos", true)
    tipo = if (esFugitivo) SERVICE_TYPE.FUGITIVOS else SERVICE_TYPE.ATRAPADOS
    var urlConnection: HttpURLConnection? = null
    try {
        urlConnection = getStructuredRequest(tipo,
            if (esFugitivo) endpoint_fugitivos else endpoint_atrapados,
            if (params.size > 1) params[1]!! else "")
        val inputStream = urlConnection?.inputStream ?: return false
        val reader = BufferedReader(InputStreamReader(inputStream))
        val buffer = StringBuffer()
        do {
            val line: String? = reader.readLine()
            if (line != null) buffer.append(line).append("\n")
        } while (line != null)
        if (buffer.isEmpty()) return false
        JSONStr = buffer.toString()
        Log.d(TAG, "Respuesta del Servidor: $JSONStr")
        return true
    } catch (e: FileNotFoundException) {
        manageError(urlConnection)
        return false
    } catch (e: IOException) {
        manageError(urlConnection)
        return false
    } catch (e: Exception) {
        manageError(urlConnection)
        return false
    } finally {
        urlConnection?.disconnect()
    }
}

```

Habrán algunos métodos que no se encontraron y es por que aun no los hemos creado, el método que crearemos primero será “`getStructuredRequest`” que nos ayudará a poder configurar nuestro objeto `HttpURLConnection` para las diferentes llamadas que necesitaremos hacer (GET, POST) para los endpoints de Fugitivos y atrapados.

```

@Throws(IOException::class, JSONException::class)
private fun getStructuredRequest(type: SERVICE_TYPE, endpoint: String, id:
String): HttpURLConnection {
    val TIME_OUT = 500
    val urlConnection: HttpURLConnection

```

```

val url: URL?
if (type === SERVICE_TYPE.FUGITIVOS) { //----- GET Fugitivos-----
    url = URL(endpoint)
    urlConnection = url.openConnection() as HttpURLConnection
    urlConnection.setReadTimeout(TIME_OUT)
    urlConnection.setRequestMethod("GET")
    urlConnection.setRequestProperty("Content-Type", "application/json")
    urlConnection.connect()
} else { //----- POST Atrapados-----
    url = URL(endpoint)
    urlConnection = url.openConnection() as HttpURLConnection
    urlConnection.setRequestMethod("POST")
    urlConnection.setReadTimeout(TIME_OUT)
    urlConnection.setRequestProperty("Content-Type", "application/json")
    urlConnection.setDoInput(true)
    urlConnection.setDoOutput(true)
    urlConnection.connect()
    val `object` = JSONObject()
    `object`.put("UDIDString", id)
    val dataOutputStream = DataOutputStream(urlConnection.getOutputStream())
    dataOutputStream.write(`object`.toString().toByteArray())
    dataOutputStream.flush()
    dataOutputStream.close()
}
Log.d(TAG, url.toString())
return urlConnection
}

```

Lo siguiente que hay que crear es el manejo de los errores que se hace desde el método “manageError” el cual lee desde el objeto HttpURLConnection el error.

```

private fun manageError(urlConnection: HttpURLConnection?) {
    if (urlConnection != null) {
        try {
            codigo = urlConnection.responseCode
            if (urlConnection.errorStream != null) {
                val inputStream = urlConnection.inputStream
                val reader = BufferedReader(InputStreamReader(inputStream))
                val buffer = StringBuffer()
                do {
                    val line: String? = reader.readLine()
                    if (line != null) buffer.append(line).append("\n")
                } while (line != null)
                error = buffer.toString()
            } else {
                mensaje = urlConnection.responseMessage
            }
            error = urlConnection.errorStream.toString()
            Log.e(TAG, "Error: $error, code: $codigo")
        } catch (e1: IOException) {
            e1.printStackTrace()
        }
    }
}

```



```

        Log.e(TAG, "Error")
    }
} else {
    codigo = 105
    mensaje = "Error: No internet connection"
    Log.e(TAG, "code: $codigo, $mensaje")
}
}
}

```

Por último sobreescribimos el método `onPostExecute` que es el que con ayuda de la interfaz retorna el resultado de nuestro hilo.

```

override fun onPostExecute(result: Boolean?) {
    if (result!!){
        listener.tareaCompletada(JSONStr)
    }else{
        listener.tareaConError(codigo, mensaje, error)
    }
}
}

```

El parseo del Json lo haremos con otra clase para continuar con la modularidad del proyecto. Leído esto deberá de crearse una clase en el paquete “network” en el que haremos el parseo e incluirá cada Fugitivo en la base de datos. La clase la llamaremos “JSONUtils”

```

class JSONUtils {

    companion object {
        fun parsearFugitivos(respuesta: String, context: Context): Boolean {
            val database = DatabaseBountyHunter(context)
            try {
                val array = JSONArray(respuesta)
                for (i in 0 until array.length()) {
                    val objeto = array.getJSONObject(i)
                    val nombreFugitivo = objeto.optString("name", "")
                    database.insertarFugitivo(Fugitivo(0, nombreFugitivo, 0))
                }
            } catch (e: JSONException) {
                e.printStackTrace()
                return false
            } finally {
                return true
            }
        }
    }
}

```

En el Activity “ListFragment.kt” se utilizará el método “actualizarDatos” para hacer la llamada al servicio web en caso de que no haya ningún fugitivo en la base de datos.

```
private fun actualizarDatos(listView: ListView?, modo: Int) {
    val database = DatabaseBountyHunter(context!!)
    val fugitivos = database.obtenerFugitivos(modo)
    if (fugitivos.isNotEmpty()) {
        val values = ArrayList<String>()
        fugitivos.mapTo(values) { it.name }
        val adaptador = ArrayAdapter<String>(context,
            R.layout.item_fugitivo_list, values)
        listView!!.adapter = adaptador
        listView.tag = fugitivos
    } else { // La base de datos se encuentra vacía
        if (modo == 0) {
            val services = NetworkServices(object: onTaskListener {
                override fun tareaCompletada(respuesta: String) {
                    JSONUtils.parsearFugitivos(respuesta, context!!)
                    actualizarDatos(listView, modo)
                }

                override fun tareaConError(codigo: Int, mensaje: String,
                    error: String) {
                    Toast.makeText(
                        context,
                        "Ocurrió un problema con el Webservice!!! --- Código
de error: $codigo \nMensaje: $mensaje",
                        Toast.LENGTH_LONG).show()
                    }
            })
            services.execute("Fugitivos")
        }
    }
}
```

En el DetalleActivity.kt se agregará una variable estática para contener el ID del dispositivo, la carga del ID se puede hacer en el onCreate del Activity:

```
private var UDID: String? = ""
```

Dentro de onCreate se inicializara UDID:

```
@SuppressWarnings("HardwareIds")
UDID = Settings.Secure.getString(contentResolver, Settings.Secure.ANDROID_ID)
```

En el mismo DetalleActivity.kt agregar un método para el mensaje de respuesta del Servicio Web al atrapar un fugitivo:

```

fun mensajeDeCerrado(mensaje: String) {
    val builder = AlertDialog.Builder(this)
    builder.create()
    builder.setTitle("Alerta!!!")
        .setMessage(mensaje)
        .setOnDismissListener {
            setResult(fugitivo!!.status)
            finish()
        }.show()
}

```

Ubicar el método de captura al fugitivo para utilizar el Servicio Web e informar que se ha capturado un nuevo fugitivo:

```

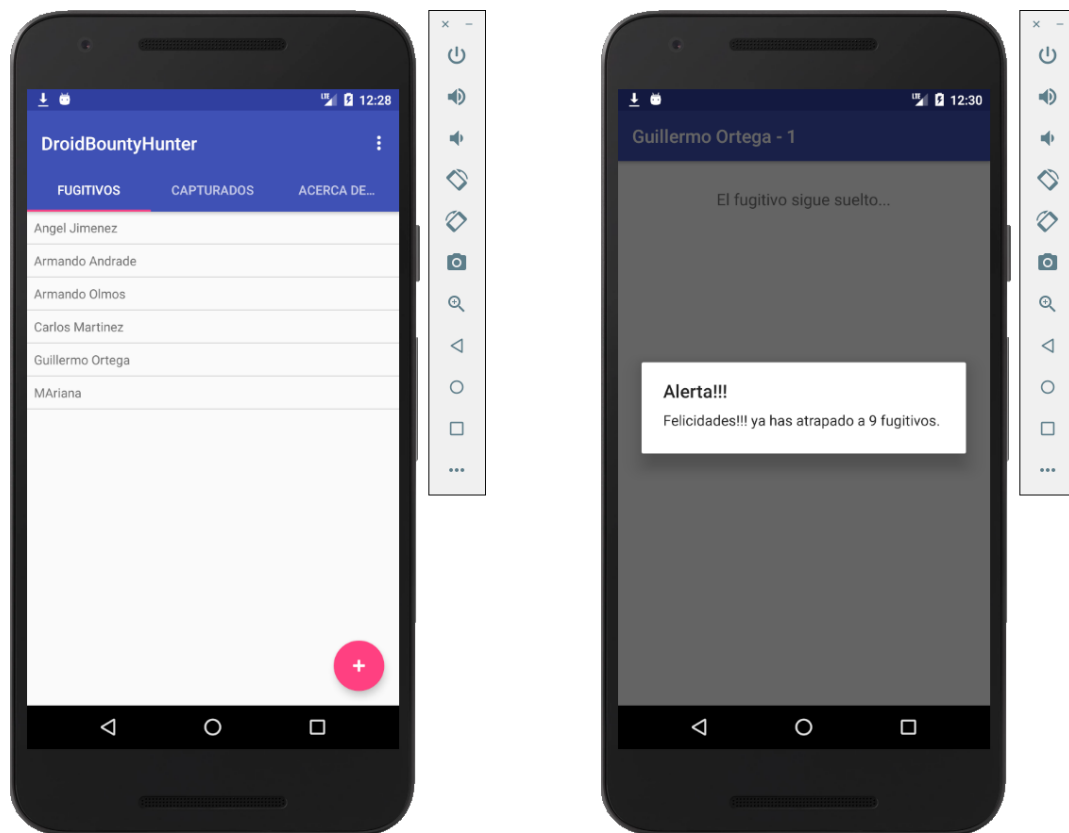
fun capturarFugitivoPresionado(view: View) {
    database = DatabaseBountyHunter(this)
    fugitivo!!.status = 1
    database!!.actualizarFugitivo(fugitivo!!)
    val services = NetworkServices(object: onTaskListener{
        override fun tareaCompletada(respuesta: String) {
            val obj = JSONObject(respuesta)
            val mensaje = obj.optString("mensaje", "")
            mensajeDeCerrado(mensaje)
        }

        override fun tareaConError(codigo: Int, mensaje: String, error: String) {
            Toast.makeText(applicationContext,
                "Ocurrió un problema en la comunicación con el Webservice!!!",
                Toast.LENGTH_LONG).show()
        }
    })
    services.execute("Atrapar", UDID)
    botonCapturar.visibility = GONE
    botonEliminar.visibility = GONE
    setResult(0)
}

```

En caso que el Servicio Web se complete correctamente, se mandará llamar el método de mensaje final generado; en caso de error se mostrará el mensaje, se omite el “finish()” del método para permitir mostrar el mensaje sin truncar el Servicio Web asíncrono, al final se ocultan los botones para evitar acciones mientras se procesa la información en el Servicio Web. El “botonCapturar” y “botonEliminar” se toman de syntethics, por lo que no se necesita tener una instancia de ese objeto declarado en nuestro activity.

Finalmente ejecutar la app y eliminar los registros para que se detecte que es necesario cargarlos del Servicio Web (el diseño podría variar según la versión del android studio, debido al manejo de las plantillas):





LAB04: USO DE CÁMARA NATIVA

Objetivos

- Utilizar la cámara para tomar una foto al fugitivo y almacenar la imagen en el dispositivo.
- Mostrar la imagen capturada del fugitivo al mostrar el detalle de los capturados.

Resumen

En este laboratorio se utilizará la cámara para tomar la foto del fugitivo que se está capturando, almacenarla como imagen en la memoria externa y recuperarla al ver el detalle del fugitivo capturado.

Punto de Partida

Se utilizará el proyecto existente para complementarse con los archivos del “Lab04” de los laboratorios electrónicos siguiendo la ejecución del Laboratorio.

Ejecución del Laboratorio

Sobre el proyecto del Lab03 se importará en src el archivo del Lab04 “PictureTools.kt”, clase diseñada para la adecuación de imágenes en los controles.

Agregar al AndroidManifest el permiso para utilizar la cámara y para escribir en la memoria externa:

```
<uses-permission android:name="android.permission.CAMERA"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>

<uses-feature android:name="android.hardware.camera"/>
```

Agregar además al Androidmanifest un Provider que nos servirá para acceder a los archivos del dispositivo en las versiones más nuevas de Android.

```
<provider
    android:name="android.support.v4.content.FileProvider"
    android:authorities="${applicationId}.provider"
    android:exported="false"
    android:grantUriPermissions="true">
    <meta-data
        android:name="android.support.FILE_PROVIDER_PATHS"
        android:resource="@xml/provider_paths"/>
</provider>
```

Crear un archivo nuevo llamado “provider_paths.xml” en “res/xml”.

```
<?xml version="1.0" encoding="utf-8"?>
<paths>
    <external-path name="external_files" path="." />
</paths>
```

Agregar a los strings en la carpeta “/res/values/” un nuevo elemento para el nombre del botón para tomar la foto:

```
<string name="boton_foto">Tomar Foto</string>
```

Agregar al XML del Layout del Activity Detalle (activity_detalle.xml) un ImageView y un botón para tomar la foto. A continuación se muestra el código del ImageView donde se mostrará la foto del fugitivo.

```
<ImageView
    android:id="@+id/pictureFugitive"
    android:layout_width="200dp"
    android:layout_height="200dp"
    android:layout_below="@id/etiquetaMensaje"
    android:layout_centerHorizontal="true"
    android:src="@android:drawable/ic_menu_gallery"/>
```

Se cambia la referencia de la vista del “botonCapturar”:

```
<Button
    android:id="@+id/botonCapturar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:minHeight="30dp"
    android:minWidth="100dp"
    android:text="@string/boton_capturar"
    android:textAppearance="@style/Base.TextAppearance.AppCompat.Body1"
    android:layout_marginTop="20dp"
    android:layout_below="@+id/pictureFugitive"
    android:layout_centerHorizontal="true"
    android:onClick="capturarFugitivoPresionado"/>
```

Agregar al nuevo botón la referencia al método a ejecutar “onFotoClick” (código del botón):

```
<Button
    android:id="@+id/botonTomarFoto"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/boton_foto"
    android:layout_below="@+id/botonEliminar"
    android:layout_centerHorizontal="true"
    android:textAppearance="@style/Base.TextAppearance.AppCompat.Body1"
    android:onClick="OnFotoClick"/>
```

Modificar el Activity DetalleActivity.kt para la captura de la Foto agregando variables (propiedades de clase) de configuración y control:

```
private var direccionImagen: Uri? = null
private val REQUEST_CODE_PHOTO_IMAGE = 1787
```

Agregar el método “onFotoClick” para el botón de la UI y métodos para el manejo de Archivos con Uri:

```
fun OnFotoClick(view: View) {
    if (PictureTools.permissionReadMemmory(this)) {
        obtenFotoDeCamara()
    }
}

private fun obtenFotoDeCamara() {
    val intent = Intent(MediaStore.ACTION_IMAGE_CAPTURE)
    direccionImagen = PictureTools.getOutputMediaFileUri(this, MEDIA_TYPE_IMAGE)
    intent.putExtra(MediaStore.EXTRA_OUTPUT, direccionImagen)
    startActivityForResult(intent, REQUEST_CODE_PHOTO_IMAGE)
}
```

Se configura el onActivityResult para identificar la respuesta de la cámara:

```
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)
    if (requestCode == REQUEST_CODE_PHOTO_IMAGE) {
        if (resultCode == Activity.RESULT_OK) {
            fugitivo!!.photo = PictureTools.currentPhotoPath
            val bitmap = PictureTools
                .decodeSampledBitmapFromUri(PictureTools.currentPhotoPath, 200, 200)
            pictureFugitive.setImageBitmap(bitmap)
        }
    }
}
```

La imagen fue almacenada de manera automática en el Uri por el Intent de la cámara al recibir el parámetro de EXTRA_OUTPUT.

Actualizar el evento “onCreate()” para identificar cuando es un fugitivo capturado y desplegar su imagen almacenada en el ImageView:

```
override fun onCreate(savedInstanceState: Bundle?) {

    @SuppressWarnings("HardwareIds")
    UDID = Settings.Secure.getString(contentResolver, Settings.Secure.ANDROID_ID)

    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_detalle)
    fugitivo = intent.extras["fugitivo"] as Fugitivo
    // Se obtiene el nombre del fugitivo del objeto fugitivo y se usa como título
    title = fugitivo!!.name + " - " + fugitivo!!.id
    // Se identifica si es Fugitivo o capturado para el mensaje...
    if (fugitivo!!.status == 0){
        etiquetaMensaje.text = "El fugitivo sigue suelto..."
    }else{
        etiquetaMensaje.text = "Atrapado!!!"
        botonCapturar.visibility = GONE
        if (fugitivo!!.photo.isNotEmpty()){
            val bitmap = PictureTools.decodeSampledBitmapFromUri(fugitivo!!.photo,
                                                                    200, 200)
            pictureFugitivo.setImageBitmap(bitmap)
        }
    }
}
```

Ahora pasaremos a agregar el miembro “photo” de tipo string en la clase “Fugitivo” y creamos un constructor nuevo utilizando la anotación “@JvmOverloads” esto creara por nosotros los constructores necesarios para los dos escenarios de con/sin photo, el path de la foto se guardará en nuestro modelo, por lo que también habrá que actualizar la implementación de Parcelable que tenemos actualmente.

```
data class Fugitivo @JvmOverloads constructor(val id: Int, var name: String, var
status: Int, var photo: String = "") : Parcelable {
    constructor(parcel: Parcel) : this(
        parcel.readInt(),
        parcel.readString(),
        parcel.readInt(),
        parcel.readString())
}
```



```

override fun writeToParcel(parcel: Parcel, flags: Int) {
    parcel.writeInt(id)
    parcel.writeString(name)
    parcel.writeInt(status)
    parcel.writeString(photo)
}

override fun describeContents() = 0

companion object CREATOR : Parcelable.Creator<Fugitivo> {
    override fun createFromParcel(parcel: Parcel): Fugitivo {
        return Fugitivo(parcel)
    }

    override fun newArray(size: Int): Array<Fugitivo?> {
        return arrayOfNulls(size)
    }
}

```

Para mantener la referencia del registro en la base de datos será necesario almacenar el Path de la imagen en Base de Datos, para ello se deberá actualizar el DBProvider.java para incluir la columna y su manejo, comenzando con incrementar la versión de la base de datos y agregando la columna:

```

/** ----- Nombre de Base de Datos ----- */
const val DATABASE_NAME = "DroidBountyHunterDatabase"
/** ----- Versión de Base de Datos ----- */
const val VERSION = 2
/** ----- Tablas y Campos ----- */
const val TABLE_NAME_FUGITIVOS = "fugitivos"
const val COLUMN_NAME_ID = "id"
const val COLUMN_NAME_NAME = "name"
const val COLUMN_NAME_STATUS = "status"
const val COLUMN_NAME_PHOTO = "photo"

class DatabaseBountyHunter(val context: Context) {
    private val TAG: String = DatabaseBountyHunter::class.java.simpleName
    /** ----- Declaración de Tablas ----- */
    private val TFugitivos = "CREATE TABLE " + TABLE_NAME_FUGITIVOS + " (" +
        COLUMN_NAME_ID + " INTEGER PRIMARY KEY NOT NULL, " +
        COLUMN_NAME_NAME + " TEXT NOT NULL, " +
        COLUMN_NAME_STATUS + " INTEGER, " +
        COLUMN_NAME_PHOTO + " TEXT, " +
        "UNIQUE (" + COLUMN_NAME_NAME + ") ON CONFLICT REPLACE);"

```

Actualizar el método de actualizarFugitivo para ingresar el dato de la foto, para poder hacer esto solo hay que agregar el dato faltante en el contentValues que se ocupa:

```
fun actualizarFugitivo(fugitivo: Fugitivo) {
    open()
    val values = ContentValues()
    values.put(COLUMN_NAME_NAME, fugitivo.name)
    values.put(COLUMN_NAME_STATUS, fugitivo.status)
    values.put(COLUMN_NAME_PHOTO, fugitivo.photo)
    database!!.update(TABLE_NAME_FUGITIVOS, values, COLUMN_NAME_ID +
        "=?", arrayOf(fugitivo.id.toString()))
    close()
}
```

Actualizar el método de obtenerFugitivos para incluir en el modelo la nueva columna:

```
fun obtenerFugitivos(status: Int) : Array<Fugitivo> {
    var fugitivos: Array<Fugitivo> = arrayOf()
    val dataCursor = querySQL("SELECT * FROM " + TABLE_NAME_FUGITIVOS + " WHERE "
        + COLUMN_NAME_STATUS + "= ? ORDER BY " + COLUMN_NAME_NAME,
        arrayOf(status.toString()))
    if (dataCursor.count > 0) {
        fugitivos = generateSequence {
            if (dataCursor.moveToNext()) dataCursor else null
        }.map {
            val name = it.getString(it.getColumnIndex(COLUMN_NAME_NAME))
            val statusFugitivo = it.getInt(it.getColumnIndex(COLUMN_NAME_STATUS))
            val id = it.getInt(it.getColumnIndex(COLUMN_NAME_ID))
            val photo = it.getString(it.getColumnIndex(COLUMN_NAME_PHOTO))
            return@map Fugitivo(id, name, statusFugitivo, photo)
        }.toList().toTypedArray()
    }
    return fugitivos
}
```

Actualizar el método insertarFugitivo

```
fun insertarFugitivo(fugitivo: Fugitivo) {
    val values = ContentValues()
    values.put(COLUMN_NAME_NAME, fugitivo.name)
    values.put(COLUMN_NAME_STATUS, fugitivo.status)
    values.put(COLUMN_NAME_PHOTO, fugitivo.photo)
    open()
    database!!.insert(TABLE_NAME_FUGITIVOS, null, values)
    close()
}
```

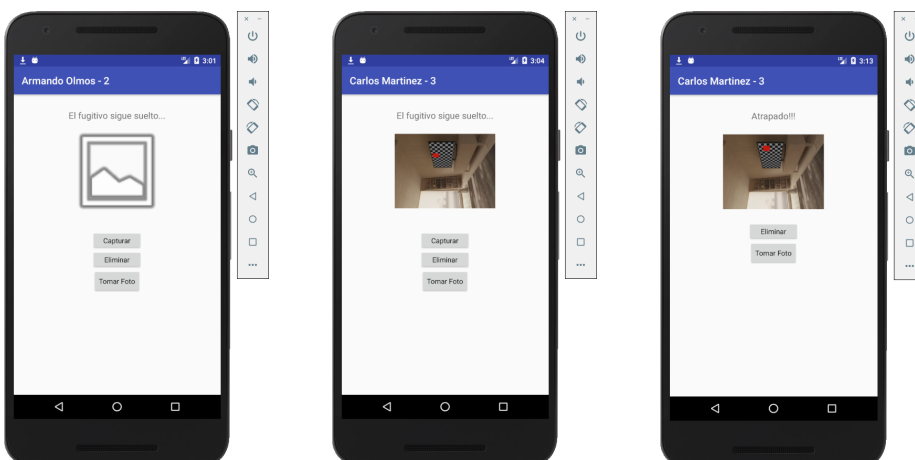
Actualizar finalmente el DetalleActivity.java para enviar el Path de la Foto al Update de Fugitivos, colocando una validación para que no permita capturar el fugitivo si no tiene ya tomada la fotografía:

```

fun capturarFugitivoPresionado(view: View) {
    database = DatabaseBountyHunter(this)
    fugitivo!!.status = 1
    if (fugitivo!!.photo.isEmpty()) {
        Toast.makeText(this,
            "Es necesario tomar la foto antes de capturar al fugitivo",
            Toast.LENGTH_LONG).show()
        return
    }
    database!!.actualizarFugitivo(fugitivo!!)
    val services = NetworkServices(object: onTaskListener{
        override fun tareaCompletada(respuesta: String) {
            val obj = JSONObject(respuesta)
            val mensaje = obj.optString("mensaje", "")
            mensajeDeCerrado(mensaje)
        }

        override fun tareaConError(codigo: Int, mensaje: String, error: String) {
            Toast.makeText(applicationContext,
                "Ocurrio un problema en la comunicación con el Webservice!!!",
                Toast.LENGTH_LONG).show()
        }
    })
    services.execute("Atrapar", UDID)
    botonCapturar.visibility = GONE
    botonEliminar.visibility = GONE
    setResult(0)
}
  
```

Finalmente se ejecuta la App, la actualización de la versión de la base de datos provocará que se elimine la tabla y se construya con la nueva columna, perdiendo los datos y trayéndonos de nuevo del Servicio web (el diseño podría variar según la versión del android studio, debido al manejo de las plantillas).





LAB05: USO DE GOOGLE MAPS Y GPS

Objetivos

- Utilizar el GPS del dispositivo para ubicar y almacenar el lugar de captura del fugitivo.
- Mostrar el lugar de captura de los fugitivos en un mapa de Google.

Resumen

En este laboratorio se utilizará el GPS del dispositivo para geo-localizar a los fugitivos al momento de su captura, los datos de latitud y longitud de captura se almacenarán en la base de datos para su posterior despliegue en un Activity que muestre un Mapa de Google al revisar el detalle de la captura del fugitivo.

Punto de Partida

Se utilizará el proyecto existente para complementarse con los archivos que se generarán en el presente laboratorio.

Ejecución del Laboratorio

Para comenzar con la ejecución del laboratorio procederemos a trasladarnos a la siguiente ruta con un usuario válido de Google para la generación de una API Key para la utilización de mapas.

<https://console.developers.google.com/project>

Lo cual nos arrojará una pantalla como la siguiente (Una vez que nos encontremos logeados) (La imagen podría variar sujeto a disponibilidad de actualización por parte de google):

Google Developers Console

Selecciona un proyecto ▾

Crear proyecto Filter by name, ID, or label Columnas Etiquetas

<input type="checkbox"/>	Nombre del proyecto	ID del proyecto	Solicitudes	Errores	Gastos		
<input type="checkbox"/>	5a19Sep15GDL CAB	scientific-elf-108315	0	0	—		
<input type="checkbox"/>	android project	api-project-152259731019	0	0	—		
<input type="checkbox"/>	API Project	api-project-923040411524	1	0	0,00 \$		
<input type="checkbox"/>	DBHunterNewProject	cogent-range-93722	0	0	—		
<input type="checkbox"/>	droidBHNew	fleet-parsec-91420	0	0	—		
<input type="checkbox"/>	droidBountyHunter	droidbountyhunter-1047	0	0	—		
<input type="checkbox"/>	droidbountyhunter1	droid-bount-app	0	0	—		
<input type="checkbox"/>	followar	formidable-bank-93920	0	0	—		

Daremos click al botón CREATE PROJECT de la pantalla anterior para que nos arroje la siguiente ventana en la que colocaremos el nombre del proyecto y por default nos creará un Project ID.

Nuevo proyecto

Nombre del proyecto ?

droidBountyHunter

El ID del proyecto es droidbountyhunter-1047 ? Edit

[Mostrar las opciones avanzadas...](#)

Crear Cancelar

Una vez realizado lo anterior damos clic en el panel principal sobre la aplicación que recién creamos y nos dirigimos a la pestaña de API's, buscamos la opción "Google Maps Android API" y la activamos quedando de la siguiente manera:

Google Developers Console

Regístrate para la versión de prueba gratuita Carlos

1 Proyecto

droidBHNew

Descripción general

Permisos

Facturación y configuración

APIs y autenticación

APIs

Credenciales

Pantalla de autorización

Puñ

Supervisión

Código fuente

Cálculo

Redes

Almacenamiento

Big Data

Asistencia

¿Necesitas ayuda?

Historia de la API Enlazada APIs (2)

Buscar las más de 100 APIs

API populares

Google Cloud APIs

- Compute Engine API
- BigQuery API
- Cloud Storage API
- Cloud Scheduler API
- Cloud Deployment Manager API
- Cloud DNS API
- Cloud IAM API

Google Maps APIs

- Google Maps Android API
- Google Maps SDK for iOS
- Google Maps JavaScript API
- Google Maps Embed API
- Places API
- Geocoding API
- Maps API

Google Apps APIs

- Drive API
- Drive SDK
- Calendar API
- Google API
- Google Apps Marketplace SDK
- Admin SDK
- Maps API

Mobile APIs

- Cloud Messaging for Android
- Google Play Game Services
- Google Play Developer API

Social APIs

- Google+ API
- Google+ Pages API
- Google+ Domains API

YouTube APIs

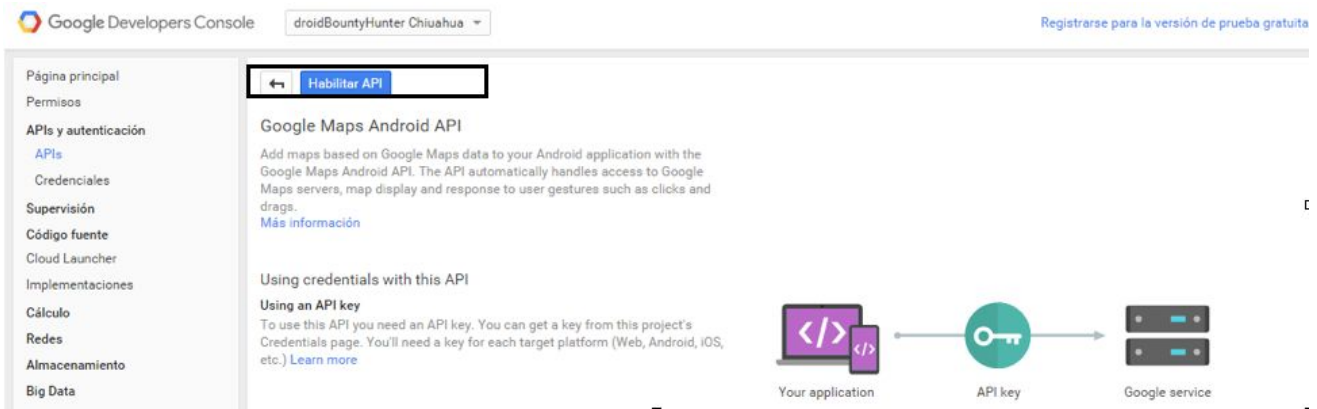
- YouTube Data API
- YouTube Analytics API

Advertising APIs

- AdSense Management API
- DoubleClick Reporting and Trafficking API
- Ad Exchange Seller API
- Ad Exchange Buyer API
- DoubleClick Search API
- Analytics API

Other popular APIs

- Translate API
- Custom Search API
- URL Shortener API
- PageSpeed Insights API
- Form Test API
- Web Font Developer API



El paso siguiente será posicionarnos en la pestaña llamada “Credentials” para dar click sobre el botón “Añadir credenciales” y “Clave de API” lo que nos arrojará el diálogo en el que elegiremos “Android Key”.



En la siguiente ventana procederemos a colocar nuestra huella sha1 del certificado con el que estamos compilando nuestra aplicación “debug.keystore”.



Para esto deberemos acceder a la “Terminal” en Android Studio o en su defecto abrir una consola de sistema para posicionarnos en la carpeta del JDK de java para la utilización de la utilidad “keytool” y extraer del certificado “debug.keystore” la huella SHA1 ejecutando los siguientes comandos:

Para MAC/Linux:

```
keytool -list -v -keystore ~/.android/debug.keystore -alias androiddebugkey
-storepass android -keypass android
```

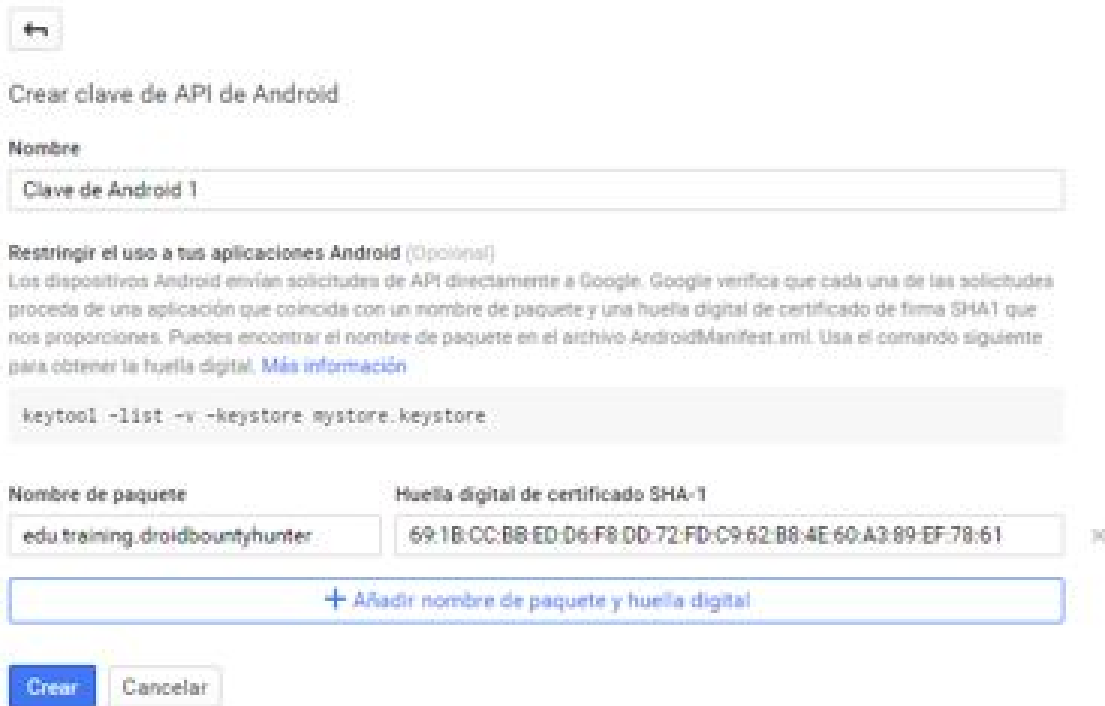
Para Windows:

```
keytool -list -v -keystore c:\users\your_user_name\.android\debug.keystore -alias
androiddebugkey -storepass android -keypass android
```

```
C:\Program Files\Java\jdk1.7.0_40\bin>keytool -list -v -keystore c:\users\Admin\.android\debug.keystore -alias androiddebugkey -storepass android -keypass android
Alias name: androiddebugkey
Creation date: 13/01/2015
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=Android Debug, O=Android, C=US
Issuer: CN=Android Debug, O=Android, C=US
Serial number: 17b0b73a
Valid from: Tue Jan 13 09:02:31 CST 2015 until: Thu Jan 05 09:02:31 CST 2045
Certificate fingerprints:
MD5: 5F:5E:41:9D:1E:94:1F:83:E6:C9:18:B1:0B:B3:90:55
SHA1: 26:FF:7B:3C:01:93:CB:AA:86:07:9C:60:82:98:D9:D5:08:8F:44:AD
SHA256: A1:A6:B9:12:A5:98:F0:13:41:1A:A1:55:58:43:98:E6:B8:E8:2A:32:76:A7:A1:BA:0A:37:DA:2F:F2:53:37:00
Signature algorithm name: SHA256withRSA
Version: 3
```

NOTA: Cabe señalar que el debug.keystore se encuentra localizado en la carpeta “.android” que se encuentra en la carpeta del usuario loggeado.

Mismo que colocaremos en la ventana de nuestra consola de developers de Google en la pestaña “Credentials” que abrimos anteriormente. Colocaremos la huella sha1 y el nombre del package del proyecto.



Crear clave de API de Android

Nombre

Clave de Android 1

Restringir el uso a tus aplicaciones Android (Opcional)

Los dispositivos Android envían solicitudes de API directamente a Google. Google verifica que cada una de las solicitudes proceda de una aplicación que coincida con un nombre de paquete y una huella digital de certificado de firma SHA1 que nos proporciones. Puedes encontrar el nombre de paquete en el archivo AndroidManifest.xml. Usa el comando siguiente para obtener la huella digital. [Más información](#)

```
keytool -list -v -keystore mystore.keystore
```

Nombre de paquete

edu.training.droidbountyhunter

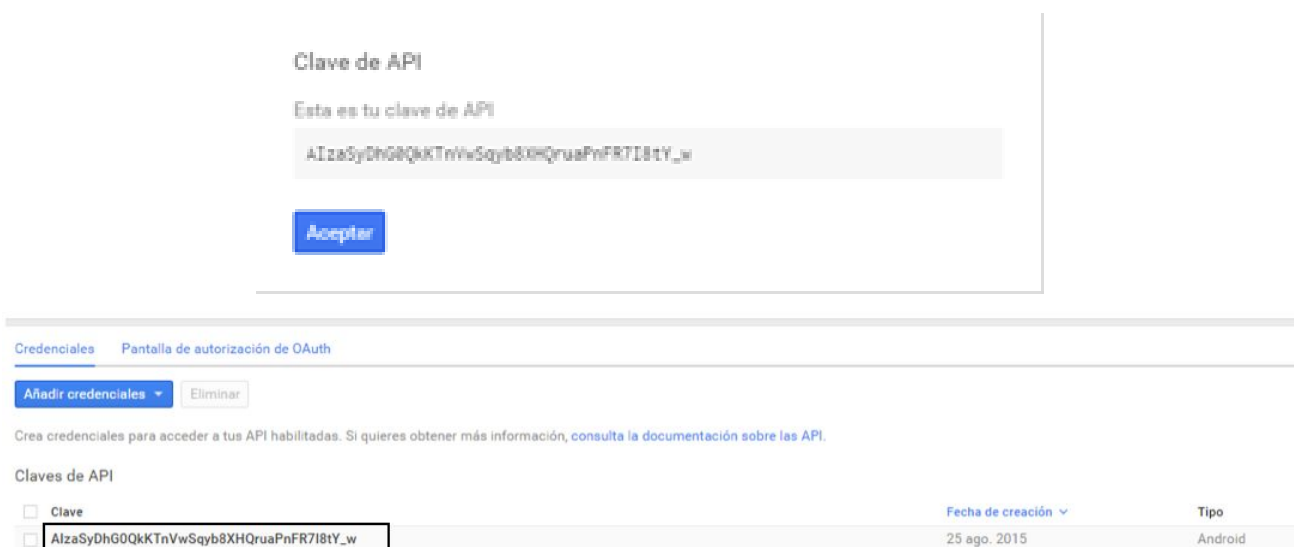
Huella digital de certificado SHA-1

69:1B:CC:BB:ED:D6:F8:DD:72:FD:C9:62:B8:4E:60:A3:89:EF:78:61

+ Añadir nombre de paquete y huella digital

Crear Cancelar

Por último damos click en CREATE.



Clave de API

Esta es tu clave de API

AIzaSyDhG0QkKTnVwSqyb8XHQuaPnFR7I8tY_w

Aceptar

Credenciales Pantalla de autorización de OAuth

Añadir credenciales Eliminar

Crea credenciales para acceder a tus API habilitadas. Si quieres obtener más información, consulta la documentación sobre las API.

Claves de API

<input type="checkbox"/> Clave	Fecha de creación	Tipo
<input type="checkbox"/> AIzaSyDhG0QkKTnVwSqyb8XHQuaPnFR7I8tY_w	25 ago. 2015	Android

Ahora pasaremos a la configuración de nuestro proyecto.

Agregar al AndroidManifest el permiso para utilizar el GPS del dispositivo:

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

Lo siguiente que haremos es cambiar nuevamente el modelo para poder acceder desde el objeto Fugitivo la ubicación apuntada por la latitud y longitud:

```
data class Fugitivo @JvmOverloads constructor(val id: Int, var name: String,
    var status: Int, var photo: String = "", var latitude: Double = 0.0,
    var longitude: Double = 0.0) : Parcelable {
    constructor(parcel: Parcel) : this(
        parcel.readInt(),
        parcel.readString(),
        parcel.readInt(),
        parcel.readString(),
        parcel.readDouble(),
        parcel.readDouble())

    override fun writeToParcel(parcel: Parcel, flags: Int) {
        parcel.writeInt(id)
        parcel.writeString(name)
        parcel.writeInt(status)
        parcel.writeString(photo)
        parcel.writeDouble(latitude)
        parcel.writeDouble(longitude)
    }

    override fun describeContents() = 0

    companion object CREATOR : Parcelable.Creator<Fugitivo> {
        override fun createFromParcel(parcel: Parcel): Fugitivo {
            return Fugitivo(parcel)
        }

        override fun newArray(size: Int): Array<Fugitivo?> {
            return arrayOfNulls(size)
        }
    }
}
```

En el activity “DetalleActivity.kt” se actualizará la funcionalidad para iniciar un proceso de detección de posicionamiento que permitirá tener el dato de Latitud y Longitud al momento de la captura, para ello se agregarán variables globales a la clase del LocationManager, LocationProvider:

```
private val REQUEST_CODE_GPS = 1234
private var locationManager: LocationManager? = null
```

Una vez declarado el LocationManager deberemos hacer que la clase DetalleActivity implemente la interfaz "LocationListener", al hacer esto deberemos sobrescribir los métodos que contiene por defecto esa interfaz.

```
class DetalleActivity : AppCompatActivity(), LocationListener{
```

y en la parte de abajo sobrescribimos los métodos:

```
override fun onLocationChanged(location: Location?) {
    TODO("not implemented") //To change body of created functions use File | Settings | File Templates.
}
```

```
override fun onStatusChanged(provider: String?, status: Int, extras: Bundle?) {
    TODO("not implemented") //To change body of created functions use File | Settings | File Templates.
}
```

```
override fun onProviderEnabled(provider: String?) {
    TODO("not implemented") //To change body of created functions use File | Settings | File Templates.
}
```

```
override fun onProviderDisabled(provider: String?) {
    TODO("not implemented") //To change body of created functions use File | Settings | File Templates.
}
```

El siguiente paso es sobrescribir onLocationChanged y hacer que el modelo de fugitivo este constantemente actualizando la ubicación:

```
override fun onLocationChanged(location: Location?) {
    fugitivo!!.latitude = location!!.latitude
    fugitivo!!.longitude = location.longitude
}
```

Agregar también métodos para Activar y Desactivar la detección del GPS.

```
@SuppressWarnings("MissingPermission")
private fun ActivarGPS() {
    if (isGPSActivated()) {
        locationManager = getSystemService(Context.LOCATION_SERVICE) as
            LocationManager
        locationManager!!.requestLocationUpdates(LocationManager.GPS_PROVIDER,
            2000, 0f, this)
        Toast.makeText(this, "Activando GPS...", Toast.LENGTH_LONG).show()
        val criteria = Criteria()
        criteria.accuracy = Criteria.ACCURACY_FINE
    }
}
```

```

// BestProvider
val provider = locationManager!!.getBestProvider(criteria, true)
// Getting last location available
val location = locationManager!!.getLastKnownLocation(provider)
if (location != null) {
    fugitivo!!.latitude = location.latitude
    fugitivo!!.longitude = location.longitude
}
}
}

private fun apagarGPS() {
    if (locationManager != null) {
        try {
            locationManager!!.removeUpdates(this)
            Toast.makeText(this, "Desactivando GPS...", Toast.LENGTH_LONG)
        } catch (e: SecurityException) {
            Toast.makeText(this, "Error desactivando GPS " + e.toString(),
                Toast.LENGTH_LONG).show()
        }
    }
}
}

```

Para que estos métodos no tengan ningún error deberemos crear otro método llamado `isGPSActivated` que hace el chequeo de permisos en caso de que sea mayor a Android 5.0 para que pueda usar los servicios de ubicación.

```

private fun isGPSActivated(): Boolean {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
        if (ActivityCompat.checkSelfPermission(this,
            Manifest.permission.ACCESS_FINE_LOCATION) !=
            PackageManager.PERMISSION_GRANTED) {

            // Should we show an explanation
            if (ActivityCompat.shouldShowRequestPermissionRationale(this,
                Manifest.permission.ACCESS_FINE_LOCATION)) {
                ActivityCompat.requestPermissions(this,
                    arrayOf(Manifest.permission.ACCESS_FINE_LOCATION),
                    REQUEST_CODE_GPS);
                return false;
            } else {
                //No explanation needed, we can request the permissions.
                ActivityCompat.requestPermissions(this,
                    arrayOf(Manifest.permission.ACCESS_FINE_LOCATION),
                    REQUEST_CODE_GPS);
                return false;
            }
        } else {
    }
}

```

```

        return true;
    }
} else {
    return true;
}
}

```

El siguiente paso será el llamar al método ActivarGPS desde el onCreate solo en caso de que aún no se haya capturado al fugitivo.

```

if (fugitivo!!.status == 0) {
    etiquetaMensaje.text = "El fugitivo sigue suelto..."
    activarGPS()
}

```

Como último paso se sobrescriba el método onDestroy del DetalleActivity para apagar el servicio de localización GPS:

```

override fun onDestroy() {
    apagarGPS()
    pictureFugitive.setImageBitmap(null)
    System.gc()
    super.onDestroy()
}

```

En este método se llama al Garbage Collector para liberar la memoria de la imagen cargada.

Lo único que resta hacer en DetalleActivity es captar con el onActivityResult si para las versiones nuevas de android se han aceptado los permisos de ubicación.

```

override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)
    if (requestCode == REQUEST_CODE_PHOTO_IMAGE) {
        if (resultCode == Activity.RESULT_OK) {
            fugitivo!!.photo = PictureTools.currentPhotoPath
            val bitmap = PictureTools
                .decodeSampledBitmapFromUri(PictureTools.currentPhotoPath,
                    200, 200)
            pictureFugitive.setImageBitmap(bitmap)
        }
    } else if (requestCode == REQUEST_CODE_GPS) {
        activarGPS()
    }
}

```

Será necesario almacenar la Latitud y Longitud en la base de datos, por lo que debemos incluir las nuevas columnas en el DatabaseBountyHunter, actualizar la versión de la base de datos y actualizar los métodos de consulta de fugitivos, inserción y update del fugitivo:

```

/** ----- Nombre de Base de Datos ----- */
const val DATABASE_NAME = "DroidBountyHunterDatabase"
/** ----- Versión de Base de Datos ----- */
const val VERSION = 3
/** ----- Tablas y Campos ----- */
const val TABLE_NAME_FUGITIVOS = "fugitivos"
const val COLUMN_NAME_ID = "id"
const val COLUMN_NAME_NAME = "name"
const val COLUMN_NAME_STATUS = "status"
const val COLUMN_NAME_PHOTO = "photo"
const val COLUMN_NAME_LATITUDE = "latitude"
const val COLUMN_NAME_LONGITUDE = "longitude"
class DatabaseBountyHunter(val context: Context) {
    private val TAG: String = DatabaseBountyHunter::class.java.simpleName
    /** ----- Declaración de Tablas ----- */
    private val TFugitivos = "CREATE TABLE " + TABLE_NAME_FUGITIVOS + " (" +
        COLUMN_NAME_ID + " INTEGER PRIMARY KEY NOT NULL, " +
        COLUMN_NAME_NAME + " TEXT NOT NULL, " +
        COLUMN_NAME_STATUS + " INTEGER, " +
        COLUMN_NAME_PHOTO + " TEXT, " +
        COLUMN_NAME_LATITUDE + " TEXT, " +
        COLUMN_NAME_LONGITUDE + " TEXT, " +
        "UNIQUE (" + COLUMN_NAME_NAME + ") ON CONFLICT REPLACE);"
  
```

En el método obtenerFugitivos se deberá de retornar las dos nuevas columnas:


```

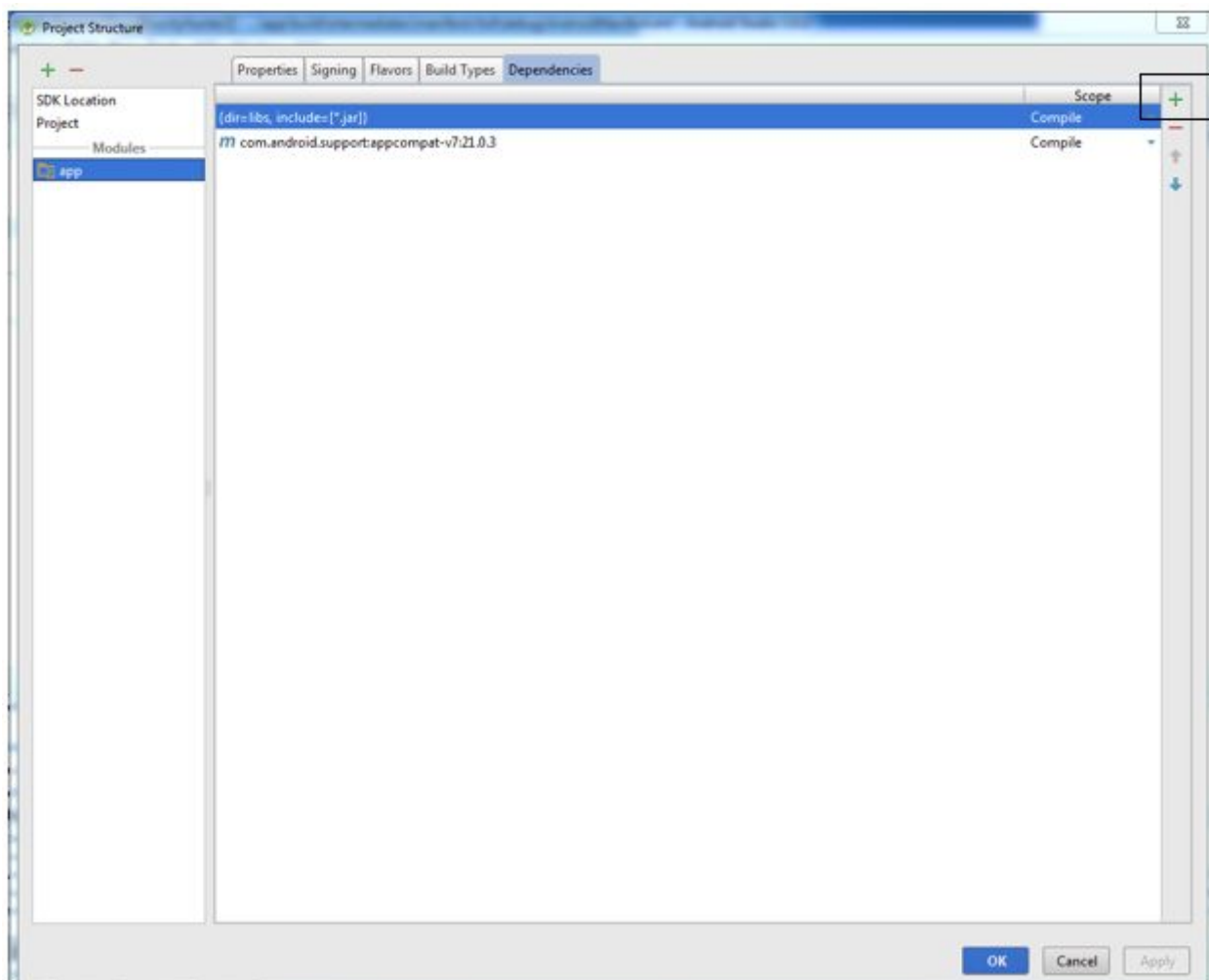
fun obtenerFugitivos(status: Int) : Array<Fugitivo> {
    var fugitivos: Array<Fugitivo> = arrayOf()
    val dataCursor = querySQL("SELECT * FROM " + TABLE_NAME_FUGITIVOS + " WHERE "
        + COLUMN_NAME_STATUS + "= ? ORDER BY " + COLUMN_NAME_NAME,
        arrayOf(status.toString()))
    if (dataCursor.count > 0) {
        fugitivos = generateSequence {
            if (dataCursor.moveToNext()) dataCursor else null
        }.map {
            val name = it.getString(it.getColumnIndex(COLUMN_NAME_NAME))
            val statusFugitivo = it.getInt(it.getColumnIndex(COLUMN_NAME_STATUS))
            val id = it.getInt(it.getColumnIndex(COLUMN_NAME_ID))
            val photo = it.getString(it.getColumnIndex(COLUMN_NAME_PHOTO))
            val latitude = it.getDouble(it.getColumnIndex(COLUMN_NAME_LATITUDE))
            val longitude = it.getDouble(it.getColumnIndex(COLUMN_NAME_LONGITUDE))
            return@map Fugitivo(id, name, statusFugitivo, photo, latitude, longitude)
        }.toList().toTypedArray()
    }
    return fugitivos
}
  
```


El método **actualizarFugitivo** y **insertarFugitivo** se deberá incluir las actualizaciones necesarias para incluir las dos columnas de latitud y longitud en el ContentValues:

```
values.put(COLUMN_NAME_LATITUDE, fugitivo.latitude)
values.put(COLUMN_NAME_LONGITUDE, fugitivo.longitude)
```

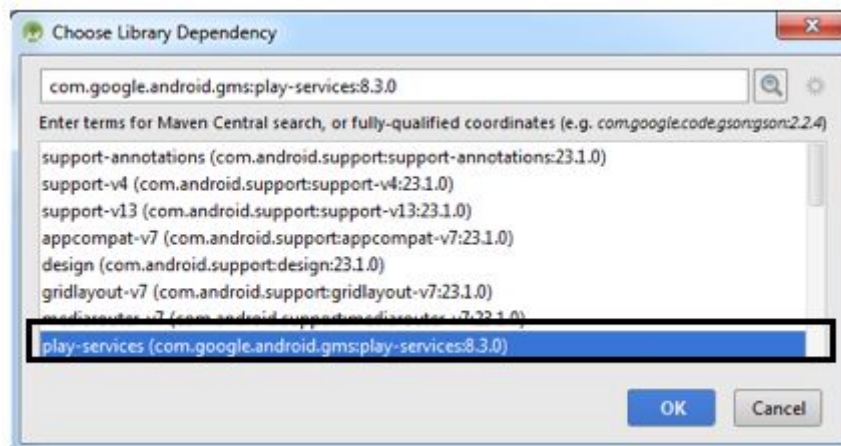
El siguiente paso será referenciar nuestra librería de google-play-services a nuestro proyecto para hacer uso de los mapas, en caso que no cuentes con ella, habrá que bajarla del sdk de android.

Primero damos click en el icono de “Project Structure”  y en la ventana saliente seleccionamos “Dependencies” tal como se muestra en la figura:



Damos click en el botón  de dependencias señalado con un cuadro en la imagen anterior.

En el menú desplegable seleccionamos “Library Dependency” y buscamos en el recuadro que nos muestra la opción “play-services(...)”. Damos click en OK y tendremos referenciada nuestra librería para la utilización de los mapas.



Posteriormente continuaremos con las siguientes indicaciones para concluir de configurar la utilización de los mapas de Google.

Agregar al AndroidManifest los permisos para usar el mapa:

```
<permission android:name="com.training.droidbountyhunter.permission.MAPS_RECEIVE"
    android:protectionLevel="signature"/>
<uses-permission
    android:name="com.training.droidbountyhunter.permission.MAPS_RECEIVE"/>
<uses-permission
    android:name="com.google.android.providers.gsf.permission.READ_GSERVICES"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>

<uses-feature android:name="android.hardware.camera"/>
```

Puesto que la versión 2 de Google Maps Android API requiere OpenGL ES versión 2, se debe agregar el elemento <uses-feature> como un nodo del elemento <manifest>:

```
<uses-feature android:glEsVersion="0x00020000" android:required="true"/>
```

Se creará el layout que contiene el Fragment del mapa (activity_mapa.xml):

```
<fragment xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/map"
    android:name="com.google.android.gms.maps.SupportMapFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

Se creará el activity MapActivity para colocar un marcador en el mapa en la latitud y longitud donde se capturó al fugitivo.

```
class MapActivity : FragmentActivity(), OnMapReadyCallback{

    private var googleMap: GoogleMap? = null
    private var fugitivo: Fugitivo? = null

    override fun onCreate(savedInstanceState: Bundle?, persistentState: PersistableBundle?)
    {
        super.onCreate(savedInstanceState, persistentState)
        setContentView(R.layout.activity_mapa)

        fugitivo = intent.getParcelableExtra("fugitivo")
        // Obtain the SupportMapFragment and get notified when the map is ready to be used.
        val mapFragment = supportFragmentManager
            .findFragmentById(R.id.map) as SupportMapFragment
        mapFragment.getMapAsync(this)
        title = fugitivo!!.name
    }

    override fun onMapReady(map: GoogleMap?) {
        this.googleMap = map

        // Add a marker in Sydney and move the camera
        val position: LatLng
        if (fugitivo!!.latitude == 0.0 && fugitivo!!.longitude == 0.0) {
            position = LatLng(-34.0, 151.0)
        } else {
            position = LatLng(fugitivo!!.latitude, fugitivo!!.longitude)
        }
        googleMap!!.addMarker(MarkerOptions()
            .position(position).title(fugitivo!!.name))
        googleMap!!.animateCamera(CameraUpdateFactory.newLatLngZoom(position, 9f))
    }
}
```

Finalmente agregamos el activity que manejará el mapa, así como la llave que creamos anteriormente, y una meta para la utilización de los mapas v2 (el código será colocado dentro del tag <application>).

NOTA: El valor que se incorporará en el recuadro que tiene por valor "TU_LLAVE" será la llave que se generó en la nube de google developers console.

Agregar el título del nuevo botón para ver el mapa en el archivo de "strings.xml":

```
<string name="boton_mapa">Ver Mapa</string>
```


En el archivo manifest:

```
<meta-data android:name="com.google.android.gms.version"
    android:value="@integer/google_play_services_version"/>
<meta-data
    android:name="com.google.android.geo.API_KEY"
    android:value="@string/google_maps_key" />
<activity android:name=".MapActivity"/>
```

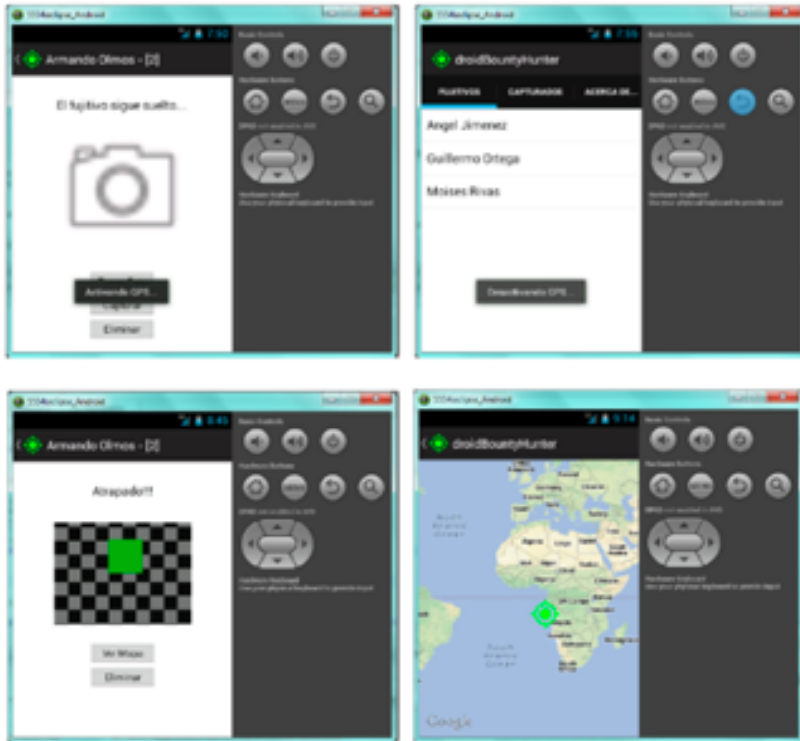
Agregar el layout de Detalle un botón para mostrar el Activity del mapa con las siguientes características:

```
<Button
    android:id="@+id/botonMapa"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/boton_mapa"
    android:layout_below="@+id/botonTomarFoto"
    android:layout_centerHorizontal="true"
    android:textAppearance="@style/Base.TextAppearance.AppCompat.Body1"
    android:onClick="OnMapClick"/>
```

Agregar el método del botón del mapa llamado "onMapClick()" para abrir el nuevo activity, en la clase DetalleActivity:

```
fun OnMapClick(view: View) {
    val intent = Intent(this, MapActivity::class.java)
    intent.putExtra("fugitivo", fugitivo)
    startActivity(intent)
}
```

Finalmente se ejecuta la app, para este laboratorio se recomienda utilizar un dispositivo con GPS (el diseño podría variar según la versión del android studio, debido al manejo de las plantillas).





LABORATORIO DE EVALUACIÓN 1

Objetivos

- Utilizar los conocimientos adquiridos en el curso para crear una lista con múltiples elementos.

Resumen

En este laboratorio se utilizará un layout con 3 elementos para mostrar el listado de los fugitivos, desplegando el nombre del Fugitivo, su fotografía en miniatura y la fecha en que fue capturado. Para poder mostrar la fecha de captura se deberá agregar una nueva columna a la base de datos.

Punto de partida

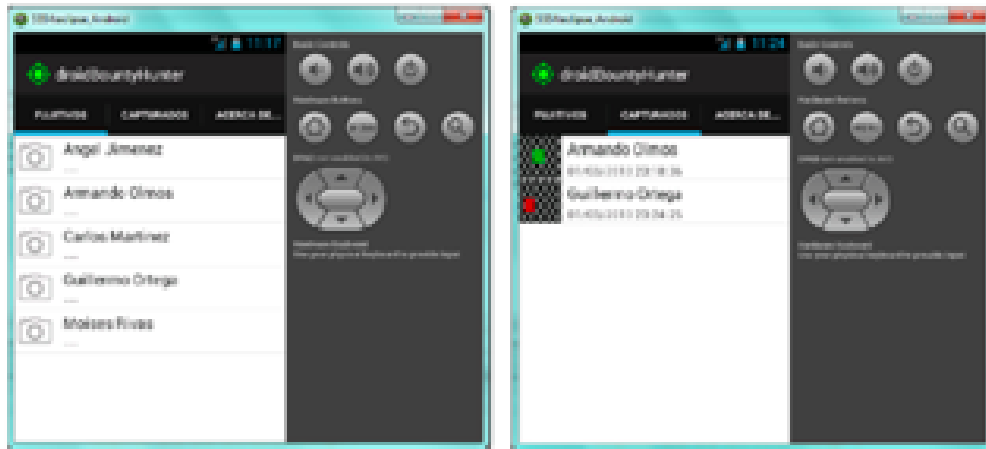
Se utilizará el proyecto existente para ajustar con la funcionalidad requerida en el presente Laboratorio de evaluación para mostrar en un adaptador personalizado las listas de fugitivos.

Ejecución del laboratorio

Los requerimientos a partir de dicho punto serán:

- Crear un adaptador personalizado donde se muestre la lista de fugitivos con la imagen del fugitivo, el nombre y la fecha de captura para los que ya están capturados.
- Agregar al DBProvider un nuevo campo para el manejo de la Fecha (tipo String), recordar actualizar la versión de la base de datos así como los métodos de Update y Consulta.
- Actualizar la carga de la lista de fugitivos y capturados utilizando el adaptador personalizado que previamente se creó.

La pantalla final deberá lucir como sigue (el diseño podría variar según el tipo de plantilla utilizada):





LABORATORIO DE EVALUACIÓN 2

Objetivos

- Hacer uso de la librería de soporte “android-support-v7-appcompat” para implementar un ActionBar en nuestra aplicación.
- Conocer cómo implementar el ActionBar y sus distintas formas para distribuir en nuestra aplicación el acomodo de los elementos de UI.

Resumen

Se mostrarán las funcionalidades de los Activities Home, AgragarActivity y DetalleActivity en un ActionBar para eliminar los botones de la UI y hacer uso de la herramienta para optimizar la distribución de elementos de Interfaz de Usuario.

Punto de partida

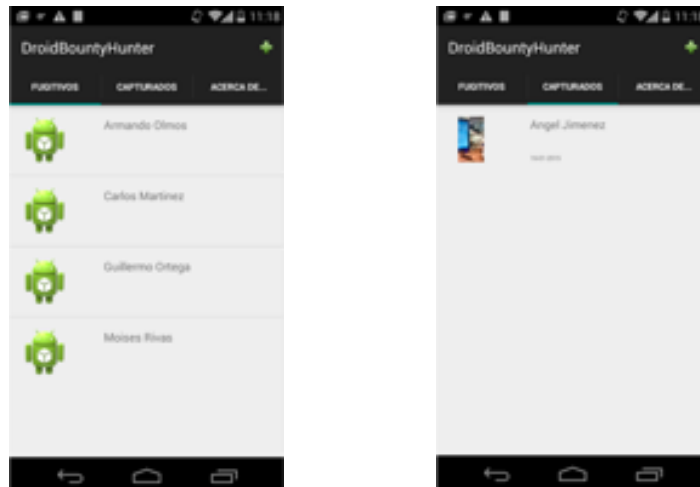
Se utilizará el proyecto existente para incorporar la funcionalidad de un ActionBar a nuestra aplicación de tal manera que las funciones de agregar, capturar, eliminar, tomar foto, mostrar en mapa sean accedidos por medio de un ActionBar.


Ejecución del laboratorio

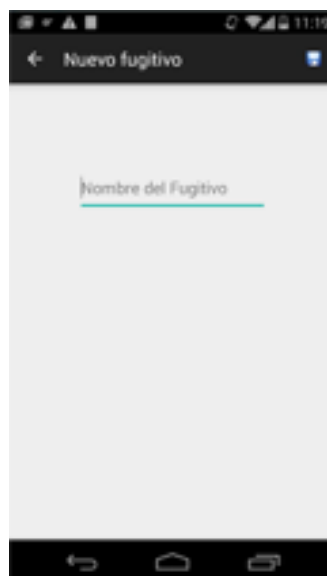
NOTA: La funcionalidad de la aplicación no cambiará, solamente cambiará en apariencia y acomodo de elementos de UI (La interfaz de usuario podría variar en base a la plantilla utilizada).

Los requerimientos a partir de dicho punto serán:

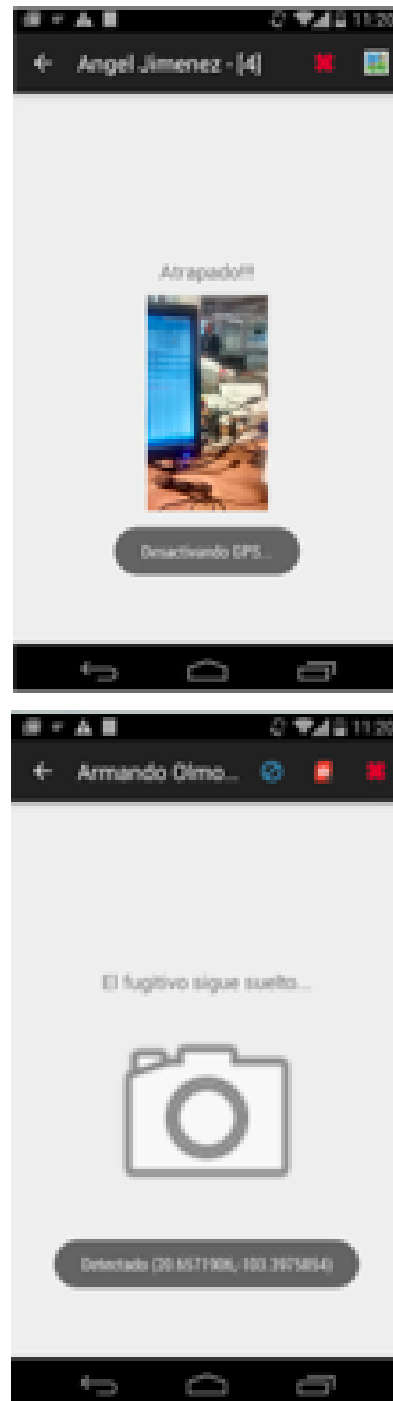
- Colocar un ActionBar en el Home en el que en la parte superior muestre la opción de agregar nuevo fugitivo (Imagen localizada en el folder /Exam02). La pantalla deberá lucir como la siguiente:



- Colocar un ActionBar en el AgregarActivity que tendrá habilitado el back en la barra superior que realizará la misma función que la tecla de Back del dispositivo destruyendo la actividad actual y mostrando la del Home, la imagen a utilizar será  (Imagen localizada en el folder /Exam02) con la funcionalidad existente del guardado del fugitivo. La pantalla deberá lucir como la siguiente:



- Colocar un ActionBar en el DetalleActivity que tendrá habilitado el back en la barra superior que realizará la misma función que la tecla de Back del dispositivo destruyendo la actividad actual y mostrando la del Home, las siguientes serán las imágenes para las acciones de tomar foto , capturar , eliminar , mostrar en mapa (imágenes localizada en el folder /Exam02), manteniendo la funcionalidad de los botones correspondientes y el comportamiento de visibilidad según el estatus del fugitivo. La pantalla deberá lucir como la siguiente:



- NOTA: Todos los botones y menús desplegables de las funcionalidades descritas en los tres puntos anteriores deberán ser ocultos.