



Universidad Nacional Autónoma de México

Facultad de Ingeniería

Cómputo Móvil

Tarea 4: Reporte primer app en Android Studio

Montoya Landa Jonás

Grupo 02

Semestre 2021-1

¿Qué es Android Studio?

Android Studio es el IDE (*Integrated Development Environment*) oficial para el desarrollo de aplicaciones móviles que corran sobre el sistema operativo Android.

Entre las ventajas de desarrollar en este ecosistema se encuentra un emulador que es rápido, con los suficientes recursos de la computadora, y lleno de herramientas disponibles para el desarrollador, además de soporte integrado para un desarrollo eficiente.

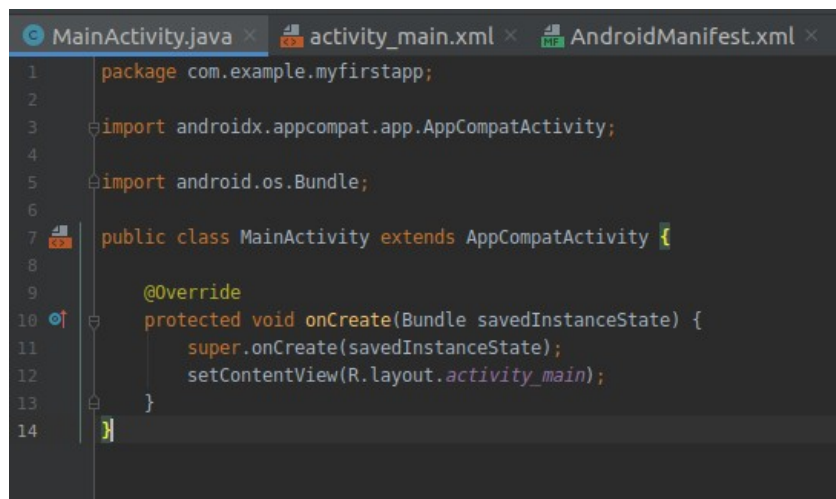
Cabe destacar que cada proyecto en Android Studio contiene uno o más módulos con archivos, entre los cuales se encuentran módulos propios de Android, Google App Engines y bibliotecas. Cabe destacar la presencia de un archivo llamado `AndroidManifest.xml` que contiene todas las dependencias del proyecto.

Desarrollo de la primera app

Llevé a cabo la instalación de Android Studio en un sistema operativo Linux Ubuntu versión 20. Por tanto, la instalación pudo llevarse a cabo de manera fácil con el archivo `.deb` disponible para descarga en la página oficial del IDE.

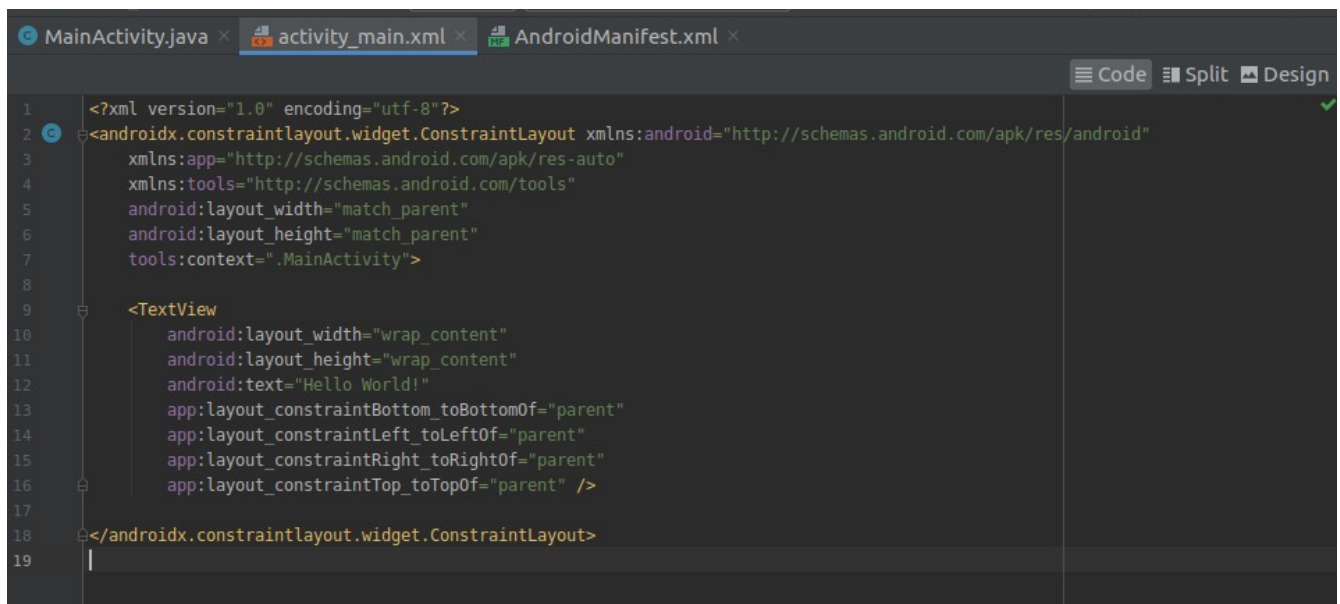
De manera general, una app en Android está compuesta por componentes que pueden ser invocados de manera individual. Dentro del desarrollo se le llama ‘actividad’ a un componente que provee una interfaz de usuario. Una vez dicho lo anterior, la actividad principal de una app comienza a ejecutarse desde el momento en que la app es abierta por el usuario.

Al momento de crear un nuevo proyecto se tiene la opción de usar dos lenguajes: Kotlin o Java. Yo elegí Java. A continuación se muestra la actividad principal que se mencionó, plasmado en una clase `.java`



```
1 package com.example.myfirstapp;
2
3 import androidx.appcompat.app.AppCompatActivity;
4
5 import android.os.Bundle;
6
7 public class MainActivity extends AppCompatActivity {
8
9     @Override
10    protected void onCreate(Bundle savedInstanceState) {
11        super.onCreate(savedInstanceState);
12        setContentView(R.layout.activity_main);
13    }
14 }
```

Además de esta clase, se tiene un archivo `activity_main.xml` que contiene todas las definiciones para la interfaz gráfica de nuestra app, correspondiente a esta actividad. Esto significa que cada actividad está asociada a un archivo xml que define su interfaz gráfica, y por tanto la clase `.java` se encarga de la parte lógica.

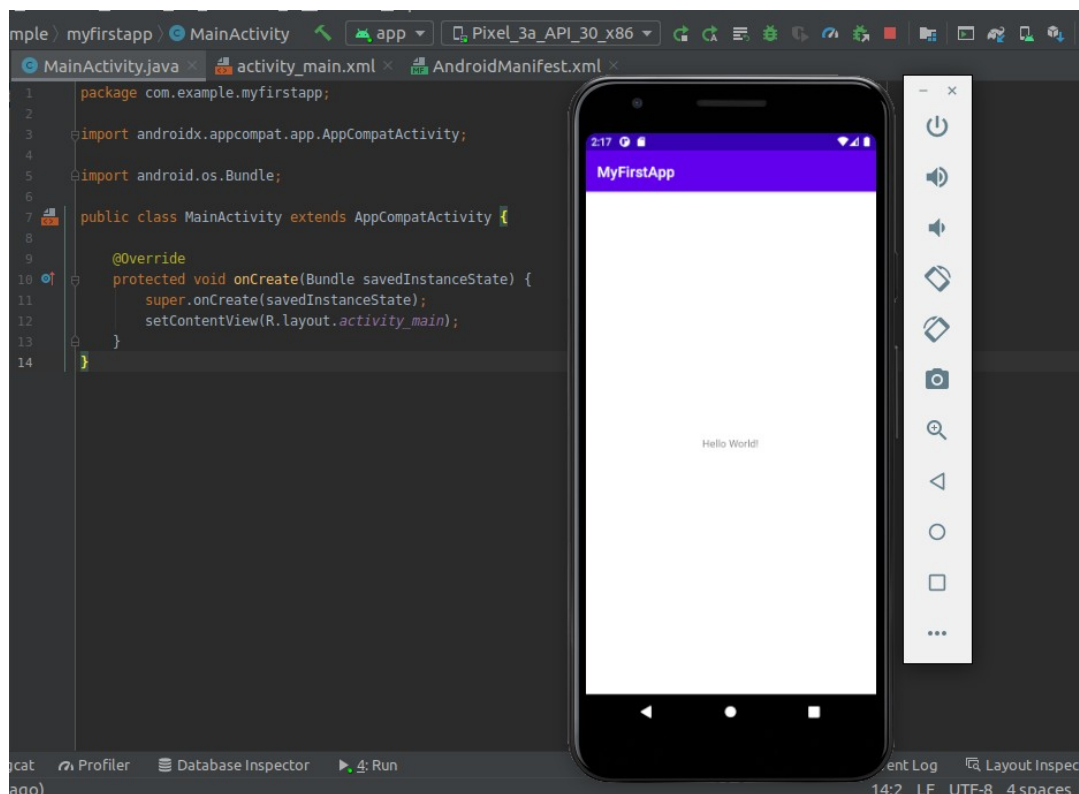


```
1 <?xml version="1.0" encoding="utf-8"?>
2 <androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
3     xmlns:app="http://schemas.android.com/apk/res-auto"
4     xmlns:tools="http://schemas.android.com/tools"
5     android:layout_width="match_parent"
6     android:layout_height="match_parent"
7     tools:context=".MainActivity">
8
9     <TextView
10         android:layout_width="wrap_content"
11         android:layout_height="wrap_content"
12         android:text="Hello World!"
13         app:layout_constraintBottom_toBottomOf="parent"
14         app:layout_constraintLeft_toLeftOf="parent"
15         app:layout_constraintRight_toRightOf="parent"
16         app:layout_constraintTop_toTopOf="parent" />
17
18 </androidx.constraintlayout.widget.ConstraintLayout>
19
```

Notar que se tiene un Hello World como atributo en la etiqueta TextView. Con lo que el comportamiento esperado es que veamos este mensaje al ejecutar nuestra app.

Dentro de las opciones que se tiene en Android Studio para ejecutar la app en desarrollo se tiene el conectar un dispositivo vía USB y probarlo en ‘tiempo real’, o bien emular. Yo elegí emular.

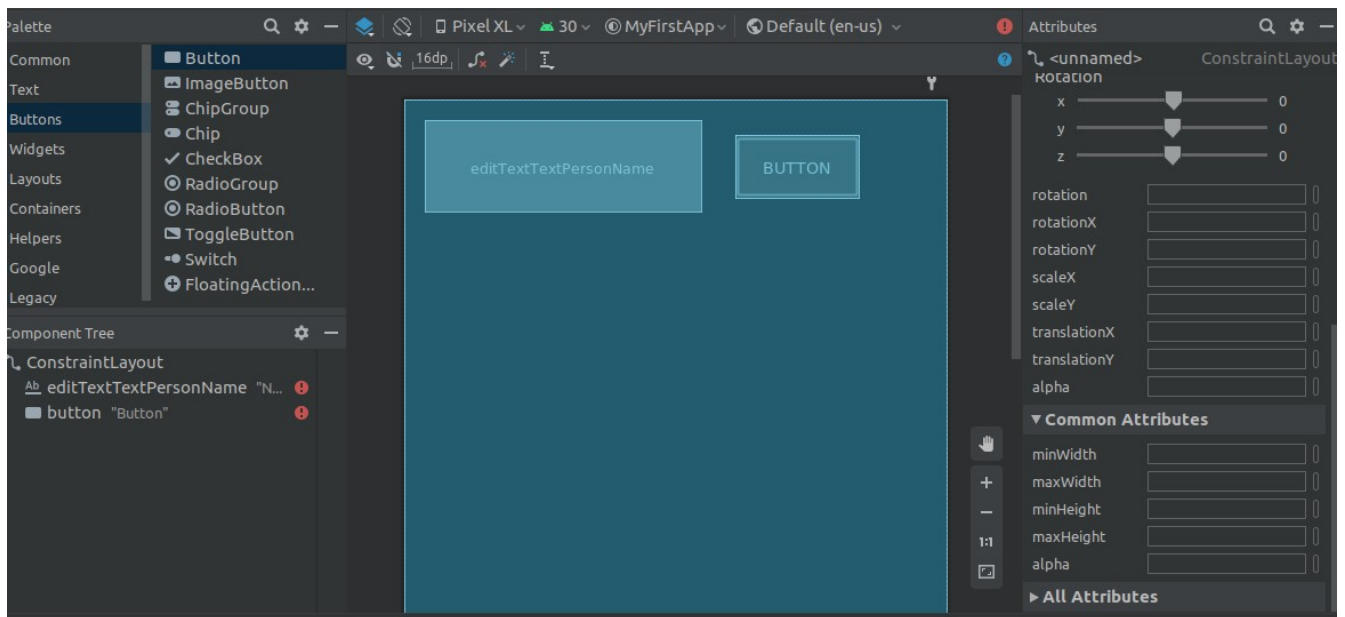
El dispositivo a emular puede elegirse en la barra de debug presente en la parte superior del IDE. Al ejecutar puede verse el mensaje de manera muy pequeña, pero es porque los atributos de dicha cadena no han sido modificados.



El siguiente paso en el tutorial fue desarrollar una entrada de texto que pudiera ser enviada al tocar un botón y mostrarse en pantalla, es decir, enviarlo a otra actividad.

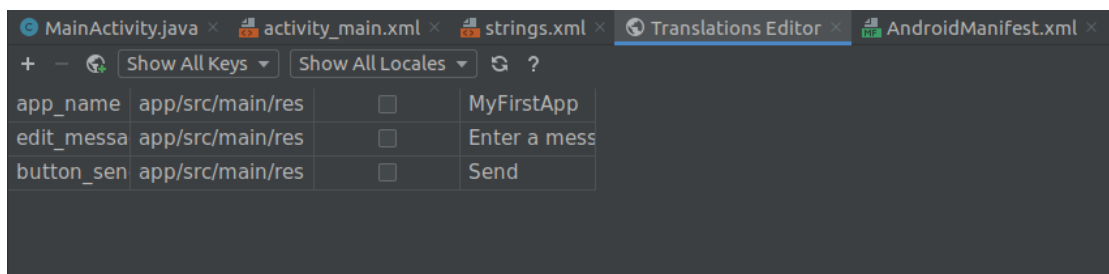
La interfaz gráfica de una app de Android tiene una jerarquía de *layouts* y *widgets*. Los primeros son objetos de tipo *ViewGroup*, mientras que los segundos son de tipo *View* tales como botones y cajas de texto. Lo anterior se puede verificar al mirar el archivo xml que ya se mencionó.

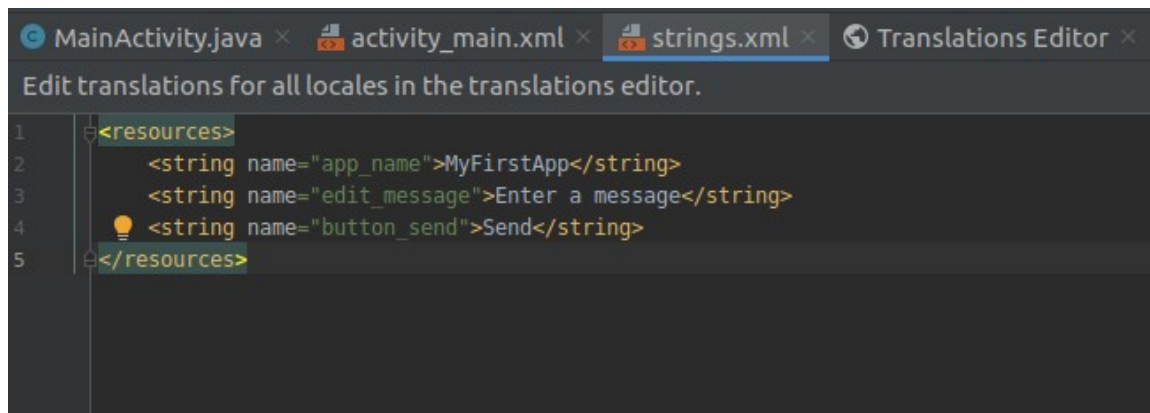
Con base en lo anterior, la interfaz puede ser programada manualmente en el archivo xml, pero el IDE ofrece un desarrollo más dinámico al incluir un diseño *drag and drop* de la interfaz. Con esto el código es automáticamente añadido por Android Studio y el desarrollador puede diseñar la interfaz arrastrando los componentes.



Como puede verse en la imagen se arrastró un botón y una caja de texto en el modo *blueprint*, que nos permite adaptar las dimensiones de los mismos.

Para cambiar las cadenas de los componentes, Android Studio tiene un archivo `strings.xml` donde se puede administrar de manera fácil todas las cadenas del proyecto. De igual manera, se cuenta con editor que hace la experiencia más amigable en lugar de escribir código xml.

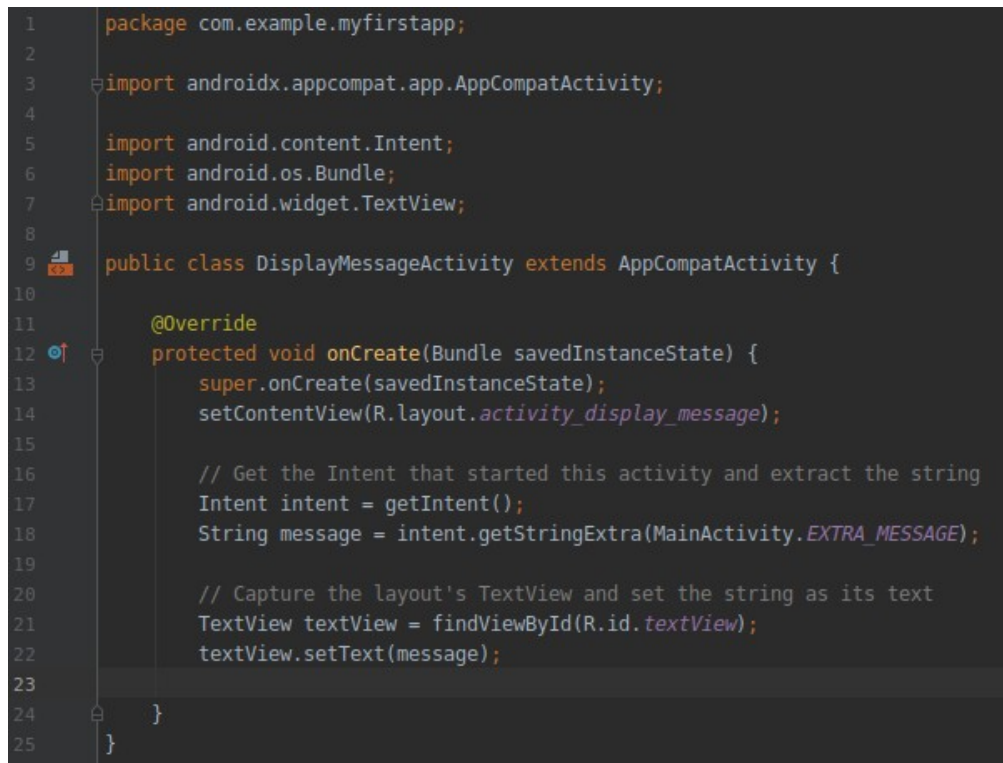




The screenshot shows the 'Translations Editor' window in Android Studio. The title bar includes tabs for 'MainActivity.java', 'activity_main.xml', 'strings.xml', and 'Translations Editor'. The main text area displays the XML content of 'strings.xml' with the following code:

```
1 <resources>
2   <string name="app_name">MyFirstApp</string>
3   <string name="edit_message">Enter a message</string>
4   <string name="button_send">Send</string>
5 </resources>
```

Para poder desplegar el mensaje cuando el usuario presione el botón, es necesario crear una nueva actividad así como su correspondiente interfaz.



The screenshot shows the 'MainActivity.java' file in Android Studio. The code defines a new activity class, 'DisplayMessageActivity', which extends 'AppCompatActivity'. The class includes an '@Override' method 'onCreate' that handles the activity's lifecycle, including setting the content view and displaying the message from the intent.

```
1 package com.example.myfirstapp;
2
3 import androidx.appcompat.app.AppCompatActivity;
4
5 import android.content.Intent;
6 import android.os.Bundle;
7 import android.widget.TextView;
8
9 public class DisplayMessageActivity extends AppCompatActivity {
10
11     @Override
12     protected void onCreate(Bundle savedInstanceState) {
13         super.onCreate(savedInstanceState);
14         setContentView(R.layout.activity_display_message);
15
16         // Get the Intent that started this activity and extract the string
17         Intent intent = getIntent();
18         String message = intent.getStringExtra(MainActivity.EXTRA_MESSAGE);
19
20         // Capture the layout's TextView and set the string as its text
21         TextView textView = findViewById(R.id.textView);
22         textView.setText(message);
23
24     }
25 }
```

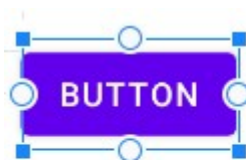
Puede apreciarse que se tiene una instancia de la clase *Intent*, la cual se encarga de conectar las dos actividades que hemos creado. Ahora sólo falta agregar un método en la actividad principal que se ejecute cada vez que el usuario presione el botón de enviar.

```

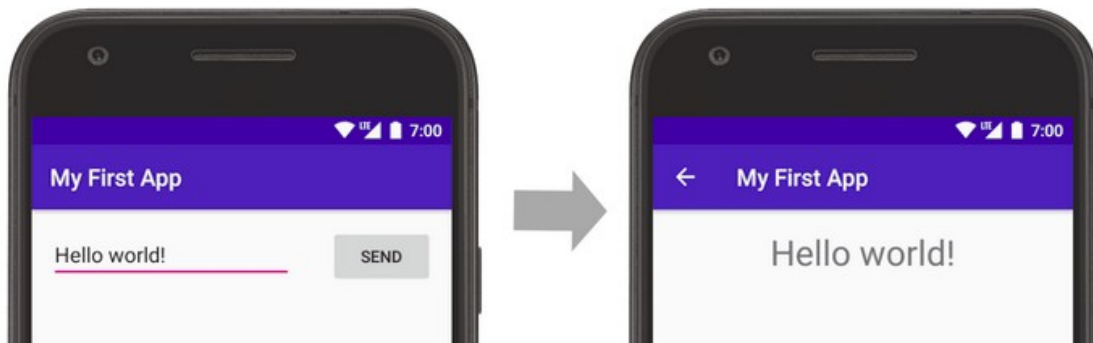
1  package com.example.myfirstapp;
2
3  import androidx.appcompat.app.AppCompatActivity;
4  import android.content.Intent;
5  import android.os.Bundle;
6  import android.view.View;
7  import android.widget.EditText;
8
9  public class MainActivity extends AppCompatActivity {
10
11      public static final String EXTRA_MESSAGE = "com.example.myfirstapp.MESSAGE";
12
13      @Override
14      protected void onCreate(Bundle savedInstanceState) {
15          super.onCreate(savedInstanceState);
16          setContentView(R.layout.activity_main);
17      }
18
19      public void sendMessage(View view){
20          Intent intent = new Intent( packageContext: this, DisplayMessageActivity.class);
21          EditText editText = (EditText) findViewById(R.id.editText);
22          String message = editText.getText().toString();
23          intent.putExtra(EXTRA_MESSAGE, message);
24          startActivity(intent);
25      }
26
27
28  }

```

El último paso es agregar esta característica al botón. En otras palabras, especificar el método sendMessage para que sea ejecutado cuando el botón sea presionado; se hace editando su atributo onClick.



nextFocusLeft		
nextFocusRight		
nextFocusUp		
numeric		
onClick	sendMessage	
orientation		
overScrollMo...		



Experiencia

Personalmente nunca había programado una aplicación en nativo. De hecho, como desarrollador front end prefiero el método híbrido el cual me es fácil por el uso de frameworks que integran el uso de Javascript, HTML y CSS.

No obstante, considero que Android Studio cuenta con sistema muy robusto para obtener un resultado final decente. La robustez del sistema puede ser un arma de dos filos, la primera que la curva de aprendizaje sea bastante larga y confusa; pero por otro lado se tiene un soporte extremadamente completo de la app a lo largo del desarrollo.

Por último, puedo decir que es necesario contar una computadora con recursos buenos para un desarrollo eficiente, ya que el IDE consume mucha RAM y suele alentarse e incluso la instalación es tardada. En contraste con la vía híbrida donde todo está integrado de manera eficaz en un npm (node package manager) y el desarrollo suele ser rápido en la línea de comandos, especialmente en sistema operativo Linux.

Referencias

- Android Developer. Documentation. Build your first app
<https://developer.android.com/training/basics/firstapp>