

DESARROLLO DE APLICACIONES MÓVILES

**PROYECTO FINAL INTEGRADOR**

**CONSIGNA**

**Desarrolla una aplicación que tenga un solo activity que cumpla con las siguientes premisas:**

**Contenga una única pantalla (sin importar el layout elegido) con:**

• 2 cuadros de textos (EditText)

• 1 botón con el texto “comparar”

• 1 texto (TextView) que en el que se escriba el resultado de la acción al presionar el botón.

**Asegurarse de que:**

1. Utiliza MVVM

2. Tiene al menos un test unitario

3. Tiene al menos un test de UI

**Función de la aplicación**

Cuando el usuario hace clic en el botón “comparar” debe comparar la entrada de ambos cuadros de texto y escribir en el texto (TextView) si ambas cadenas de caracteres son iguales o no.

**Requerimientos**

1. Usá un repositorio de GitHub pages para subir tu proyecto (Repositorio app laboratorio: <https://github.com/eaceto/ticmas-lab-android> Con lo visto en el último módulo del curso, vas a poder subir sin problema tu proyecto a la web.

2. Generá un archivo .doc con lo planteado en la consigna y el link de tu entregable.

3. Luego, subilo en el bloque "Proyectos" que se encuentra en la página principal del curso, en la plataforma.

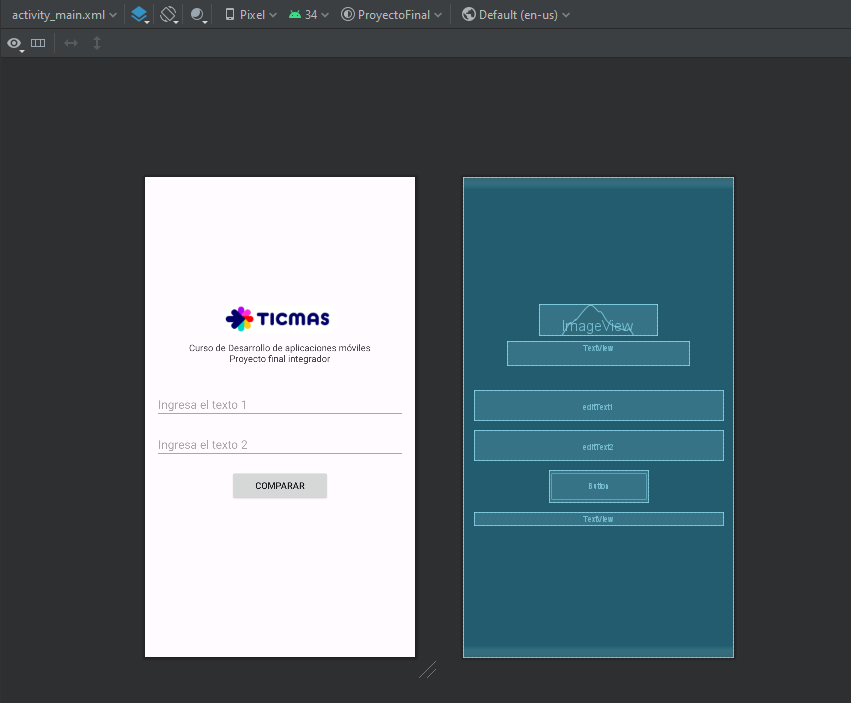
**Creación del Nuevo Proyecto**

Para el desarrollo del mismo, se eligió el modelo de plantilla **“Empty Views Activity”** considerando que nos otorgaba lo necesario para dar comienzo al proyecto; como SDK mínimo optamos por seleccionar el **API 21 – Android 5.0 Lollipop** ya que el mismo tenía una compatibilidad con el 100% de los dispositivos Android.

El nombre del proyecto es **“Proyecto Final”** y su correspondiente Package lo hemos nombrado **“com.ticmas.rodrigo.proyectofinal”.**

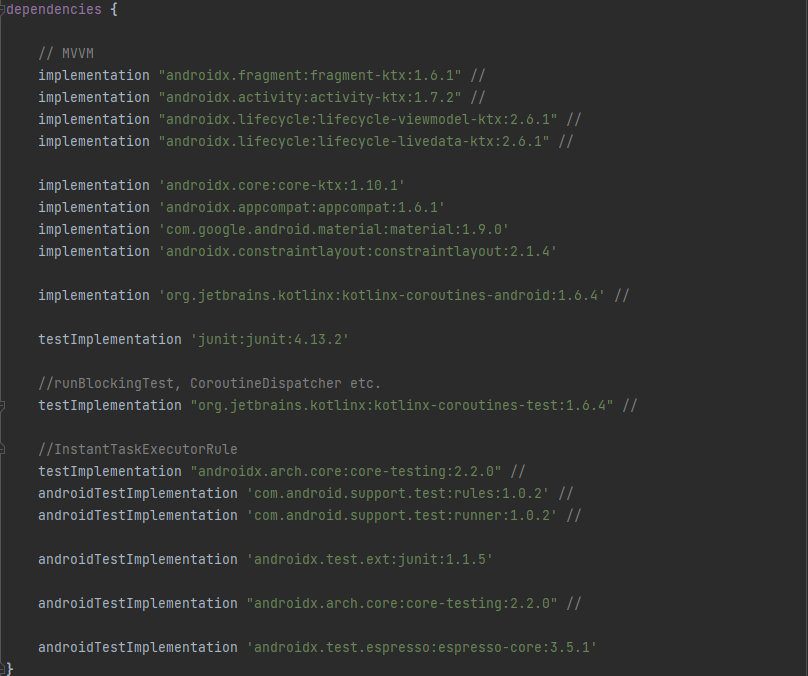
Cabe destacar que todo estará codificado en Lenguaje **Kotlin.**

**UI del Proyecto**



Se empleó un LinearLayout, sencillo y adaptable a los distintos tamaños y formatos de pantalla. Dicho archivo está ubicado en .app/res/layout

**Dependencias utilizadas en el proyecto**



* Para que Kotlin pueda trabajar con fragments utilizamos

implementation "androidx.fragment:fragment-ktx:1.6.1" //

* Para que Kotlin pueda trabajar con activities utilizamos

implementation "androidx.activity:activity-ktx:1.7.2" //

* Para que Kotlin pueda trabajar con ViewModel del ciclo de vida

implementation "androidx.lifecycle:lifecycle-viewmodel-ktx:2.6.1" //

* Para que Kotlin pueda trabajar con el LiveData del Ciclo de vida utilizamos

implementation "androidx.lifecycle:lifecycle-livedata-ktx:2.6.1" //

* Para proporcionar soporte para Coroutines en Android utilizamos

implementation 'org.jetbrains.kotlinx:kotlinx-coroutines-android:1.6.4' //

* Para brindar clases de ayuda para testing de Coroutines utilizamos

testImplementation "org.jetbrains.kotlinx:kotlinx-coroutines-test:1.6.4" //

* Para brindar clases de ayuda para testing de componentes de LiveData utilizamos

testImplementation "androidx.arch.core:core-testing:2.2.0" //

* Para brindar reglas y funcionalidades para testing de Android utilizamos

androidTestImplementation 'com.android.support.test:rules:1.0.2' //

* Para brindar el runner necesario para realizar pruebas instrumentadas utilizamos

androidTestImplementation 'com.android.support.test:runner:1.0.2' //

* Para brindar clases de ayuda para testing de JUnit 4 en pruebas instrumentadas utilizamos

androidTestImplementation 'androidx.test.ext:junit:1.1.5'

* Para brindar clases y métodos para realizar pruebas de UI utilizando Espresso contamos con

androidTestImplementation 'androidx.test.espresso:espresso-core:3.5.1'

* Para contar con la biblioteca de JUnit para realizar pruebas unitarias utilizamos

testImplementation 'junit:junit:4.13.2'

* Para trabajar con las bibliotecas de compatibilidad de Android utilizamos la extensión

implementation 'androidx.core:core-ktx:1.10.1'

* Para brindar compatibilidad con versiones anteriores de Android utilizamos

implementation 'androidx.appcompat:appcompat:1.6.1'

* Para dar estilos y componentes de diseño de Material Design para la UI utilizamos

implementation 'com.google.android.material:material:1.9.0'

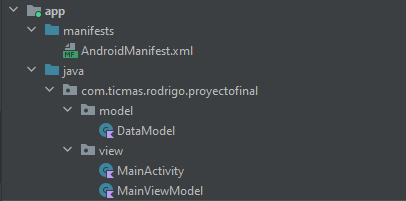
* Para contar con una biblioteca que nos permita trabajar con el diseño de constraints en las vistas utilizamos

implementation 'androidx.constraintlayout:constraintlayout:2.1.4'

**Funcionalidad de la aplicación**

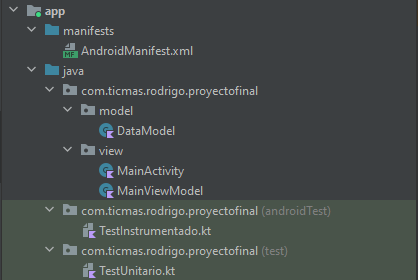
La aplicación debe cumplir la función solicitada respetando la arquitectura MVVM y los principios SOLID. Los archivos deben tener responsabilidades únicas y bien definidas, que faciliten el mantenimiento y la escalabilidad del código.

Por ello se optó por organizar el código de la siguiente manera:



* **DataModel.kt** contiene las propiedades text1, text2 y tvResult. Su responsabilidad es definir la estructura de datos
* **MainActivity.kt** representa la actividad principal de la aplicación y se encarga de la interacción con la interfaz de usuario. Interactúa con el ViewModel a través del patrón de observación (LiveData) para obtener el resultado de la comparación de los textos y actualizar la interfaz de usuario en consecuencia a ello.
* **MainViewModel.kt** representa el viewModel de la aplicación, que actúa como intermediario entre la actividad y el modelo de datos, Contiene la lógica de negocio para la comparación de textos y expone el resultado a través de LiveData para que la actividad pueda observarlo.

**Test Unitario**



**TestUnitario.kt**

Contiene las diferentes pruebas que se realizaran en la aplicación mediante el test Unitario, las cuales son las siguientes:

1. *mainViewModel\_TestCompareTexts\_EmptyFields():* Verifica si la función compareTexts() del MainViewModel maneja correctamente el caso en que ambos campos de texto están vacíos. Comprueba si el resultado mostrado en el LiveData result es "Campos incompletos".

2. *mainViewModel\_TestCompareTexts\_EqualTexts():* Verifica si la función compareTexts() del MainViewModel maneja correctamente el caso en que ambos campos de texto son iguales. Comprueba si el resultado mostrado en el LiveData result es "Los textos son IGUALES".

3. *mainViewModel\_TestCompareTexts\_DifferentTexts():* Verifica si la función compareTexts() del MainViewModel maneja correctamente el caso en que los campos de texto son diferentes. Comprueba si el resultado mostrado en el LiveData result es "Los textos son DIFERENTES".

**Test Instrumentado**

**TestInstrumentado.kt**

Contiene las diferentes pruebas que se realizaran en la aplicación mediante el test Instrumentado, las cuales son las siguientes:

1. *mainActivity\_compareTextsEqual():* Verifica si la actividad MainActivity muestra correctamente el resultado "Los textos son IGUALES" cuando se ingresan textos iguales en los EditText y se hace clic en el botón "comparar". Utiliza Espresso para interactuar con los elementos de la interfaz de usuario y verificar que el resultado se muestre correctamente en el TextView.

2. *mainActivity\_compareTextsDifferent():* Verifica si la actividad MainActivity muestra correctamente el resultado "Los textos son DIFERENTES" cuando se ingresan textos diferentes en los EditText y se hace clic en el botón "comparar". Utiliza Espresso para interactuar con los elementos de la interfaz de usuario y verificar que el resultado se muestre correctamente en el TextView.

3. *mainActivity\_compareTextsEmptyFields():* Verifica si la actividad MainActivity muestra correctamente el resultado "Campos incompletos" cuando no se ingresa texto en los EditText y se hace clic en el botón "comparar". Utiliza Espresso para interactuar con los elementos de la interfaz de usuario y verificar que el resultado se muestre correctamente en el TextView.

***Link Del Repositorio:*** <https://github.com/MrForii/Proyecto-Final-Argentina-Programa>