KOTLIN

ANDROID STUDIO

❖ DESARROLLO DE SOFTWARE PARA MÓVILES

Ing. Kevin Jiménez

❖ **Trabajo de Investigación #01**

Universidad Don Bosco

❖ **Integrates:**

- Miguel Angel Herrera Arreaga
- Diego Ismael Chavez Acevedo
- Mario Steven Cabrera Velasco

❖ **Carnet:**

HA160257
CA230991
CV230385

❖ **Grupo teórico: 01**

índice

Contenido

introducción

Este documento describe la creación de una aplicación de lista de tareas (To-Do List) con el objetivo de proporcionar una herramienta práctica y funcional para la gestión de tareas diarias. El proyecto se centra en el desarrollo de una aplicación para dispositivos Android utilizando el entorno de desarrollo Android Studio. A continuación, se detallan los objetivos y los requisitos funcionales de la aplicación:

Objetivos

- **Familiarizarse con el entorno de desarrollo de Android Studio**: Este proyecto servirá como una introducción práctica a Android Studio, permitiendo a los desarrolladores comprender su estructura y herramientas.
- **Aprender a crear una interfaz de usuario básica**: La aplicación contará con elementos esenciales de la interfaz de usuario, como EditText, Button y ListView.
- **Implementar la lógica para la gestión de tareas**: Los usuarios podrán agregar, eliminar y marcar tareas como completadas, gestionando así sus listas de manera eficiente.
- **Investigar y aplicar el concepto de persistencia de datos**: Se implementarán técnicas para asegurar que las tareas se guarden y persistan entre sesiones de la aplicación.

Requisitos Funcionales

- **Ingreso de nuevas tareas**: La aplicación permitirá al usuario ingresar una nueva tarea a través de un campo de texto.
- **Adición de tareas a la lista**: Al presionar un botón, la tarea ingresada se agregará a una lista visible en la interfaz de usuario.
- **Checkbox para completar tareas**: Cada tarea mostrará un checkbox que permitirá al usuario marcarla como completada.
- **Eliminación de tareas**: La aplicación incluirá un botón para eliminar una tarea seleccionada de la lista.
- **Diferenciación de tareas completadas**: Las tareas marcadas como completadas se mostrarán de forma distinta, por ejemplo, con un texto tachado.
- **Persistencia de datos**: Las tareas ingresadas y su estado (completadas o no) se guardarán y estarán disponibles al cerrar y reabrir la aplicación, garantizando la continuidad de la información.

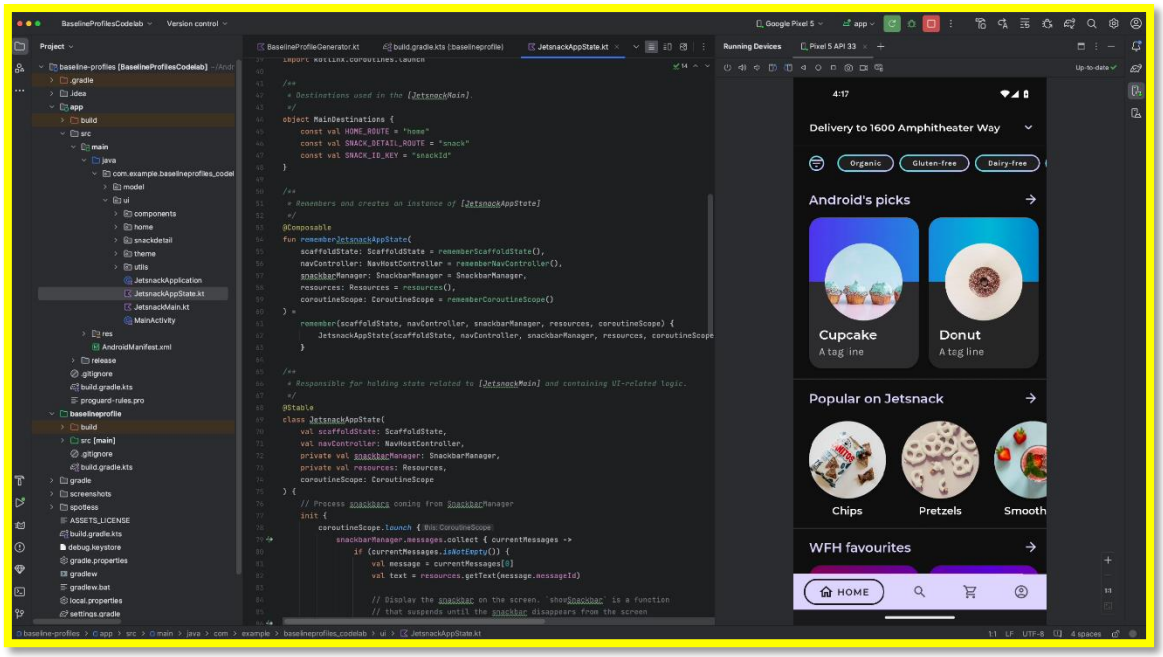
Paso 1: Investigar y aprender

Qué es y para qué sirve Android Studio:

Normalmente, toda aplicación, herramienta, página, o servidor digital que ofrece algún tipo de tarea en internet, posee lenguajes de programación o entornos de trabajo especializados. Por ejemplo, Python, que es un lenguaje muy utilizado en el desarrollo de Inteligencia Artificial.

Así, tal cual, pasa con el sistema operativo Android. Todas las aplicaciones y herramientas que se desarrollan para este SO en concreto poseen su propia área o entorno de trabajo. Ese entorno es Android Studio, que permite una flexibilidad en cuanto al **desarrollo de características y funciones** que puede tener una herramienta o app de dicho sistema.

Este entorno sirve para que las aplicaciones que se estén desarrollando sean mucho más eficiente y autosuficientes. Esto permite, incluso, tener compatibilidades con otros sistemas o plataformas.



Qué características tiene Android Studio

Android Studio permite la integración de características y funciones bastante positivas para las aplicaciones que, con el tiempo, se perfeccionan. De esta forma, tenemos lo siguiente:

El sistema de compilación es flexible, además de ser compatible con Gradle, la cual permite la automatización de compilaciones de forma flexible y con gran

rendimiento. Groovy y Kotlin DSL son los lenguajes utilizados para los scripts de compilación.

- Esta plataforma te permite desarrollar aplicaciones para cualquier dispositivo Android.
- Contiene plantillas de compilación que te ayudan a otorgar funciones comunes de otras apps de forma mucho más rápida, además de importar códigos de muestra.
- Proporciona compatibilidad con servicios en la nube tal como Google Cloud Platform.

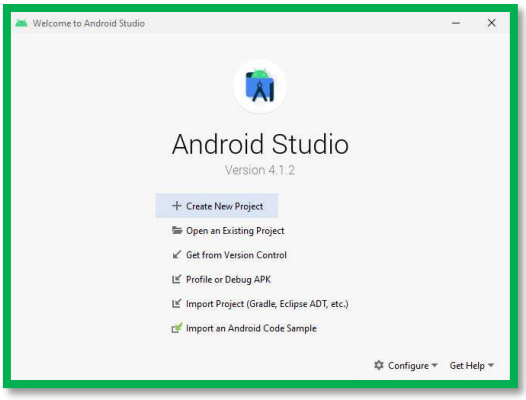
Qué lenguaje se utiliza para programar en Android Studio

Desde siempre, el sistema operativo de Android se ha desarrollado a través del lenguaje de programación Java. No hay que confundir propiamente Android Studio con un lenguaje, ya que esto es solo el entorno para desarrollar el código Java, para que se puedan crear las aplicaciones propiamente dichas. puede llegar a ser compatible con lenguajes como Kotlin (uno de los principales), NDK y C++.

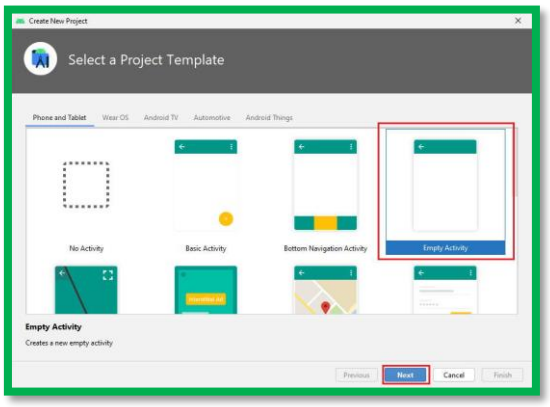
Características de Kotlin

Una de las maravillas de lenguaje de programación Kotlin, es que permite una interoperabilidad natural con Java, pudiendo incluso desarrollar código para proyectos utilizando ambos lenguajes a la vez sin ningún problema.

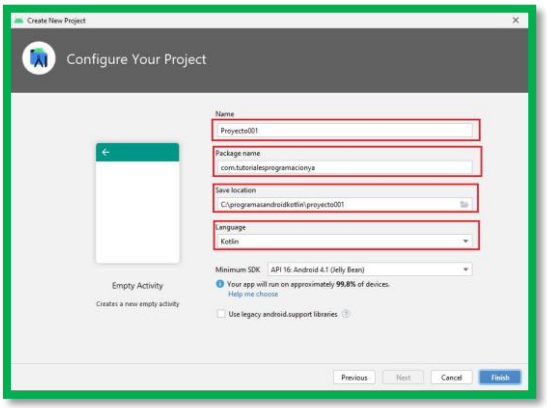
2 - Pasos para crear el primer proyecto en Android Studio con Kotlin: una vez que iniciamos el entorno del Android Studio aparece el diálogo principal



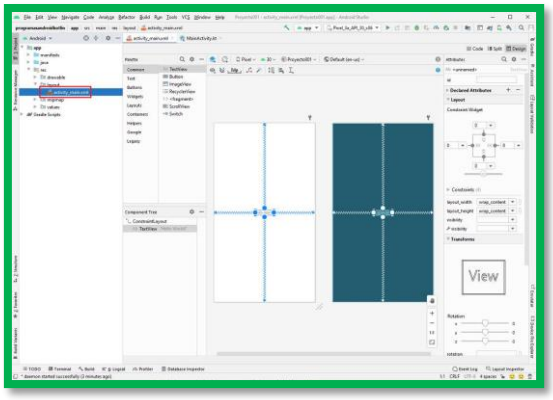
Elegimos la opción "Create New Proyect" Primero especificamos el esqueleto básico de nuestra aplicación, seleccionaremos "Empty Activity":



En la segunda ventana debemos especificar el Nombre de la aplicación, la url de nuestra empresa (que será el nombre del paquete que asigna para los archivos fuentes), la ubicación en el disco de nuestro proyecto y debemos seleccionar que incluya el soporte para programar con el lenguaje Kotlin:

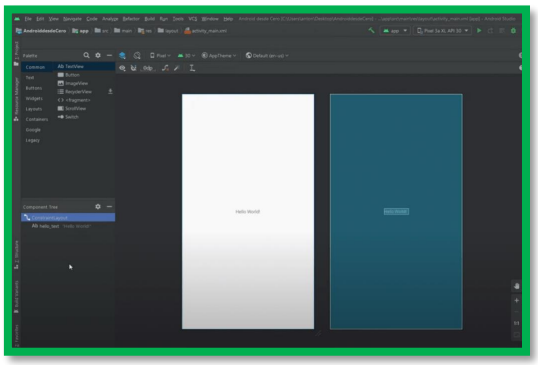


Tenemos finalmente creado nuestro primer proyecto en Android Studio configurado para programar con Kotlin y podemos ahora ver el entorno del Android Studio para codificar la aplicación:

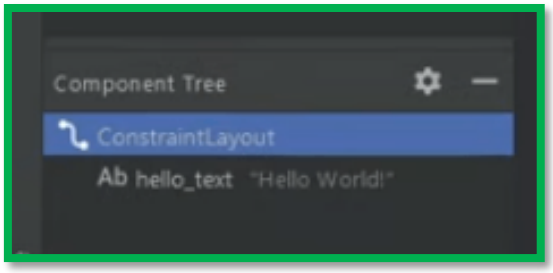


El Android Studio nos genera todos los directorios y archivos básicos para iniciar nuestro proyecto, los podemos ver en el lado izquierdo del entorno de desarrollo

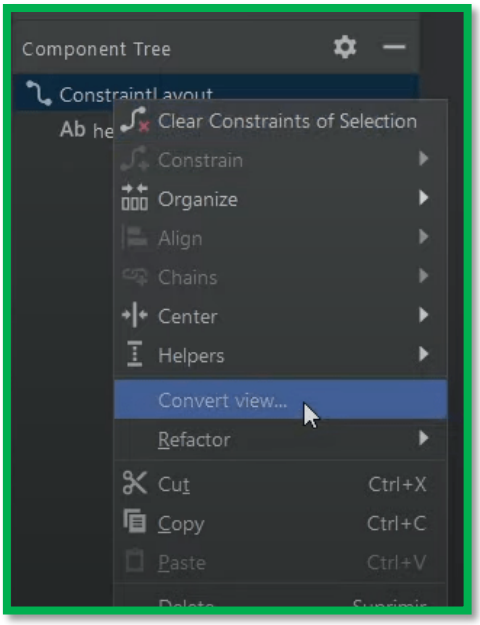
En el diseñador lo que podemos hacer es ir creando nuestra interfaz de usuario.



La interfaz estará formada en este apartado por un Layout que vendría hacer un ViewGroup.

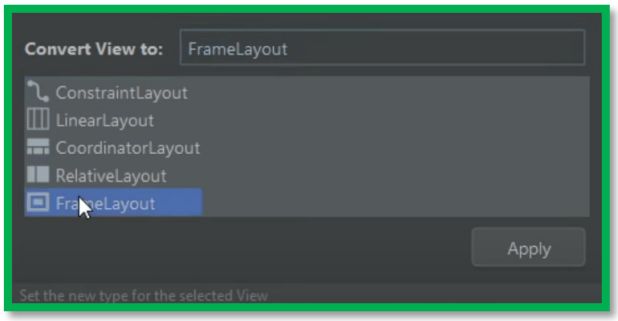


Para cambiar este ConstraintLayout que tiene a su vez una vista adentro de tipo TextView vamos hacer click derecho sobre el elemento “ConstrainLayout” donde nos aparecerá un menú desplegable y seleccionaremos la opción “Convert View...”

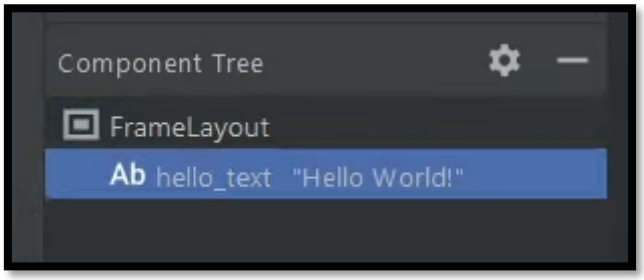


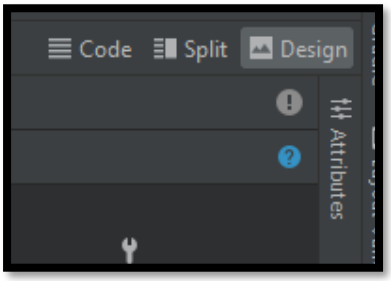
FrameLayout

El cual nos mostrara una ventana con varias opciones y vamos a elegir FrameLayout ya que es uno de los mas sencillo de usar y la única forma de tenemos de organizar las vistas es mediante un atributo que se llama layout_gravity. Se hace uso de posiciones relativas ya que no podemos definir que un elemento tendrá una separación en pixeles u otra unidad de medida fija, ya que hay distintos tipos de dispositivos con diferentes resoluciones y al rotar también cambian.

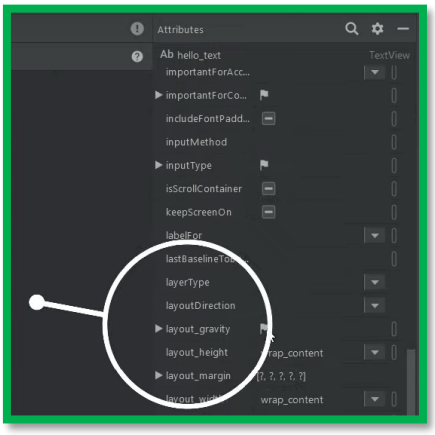


Seleccionado el TextView y luego en la parte lateral derecha de nuestro Android Studio, hacemos click en Attributes y se nos desplegaran los atributos.

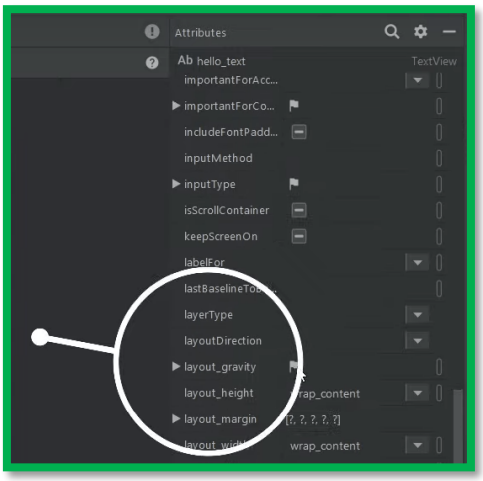




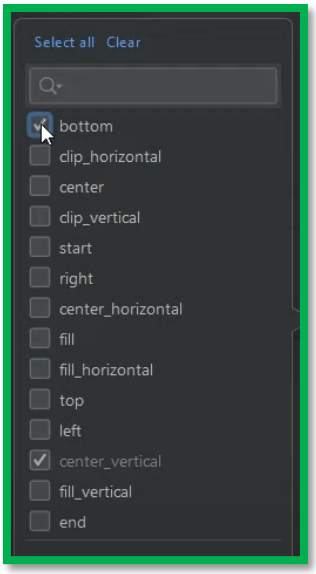
Luego nos dirigimos a la propiedad layout_gravity haciendo click sobre ella.



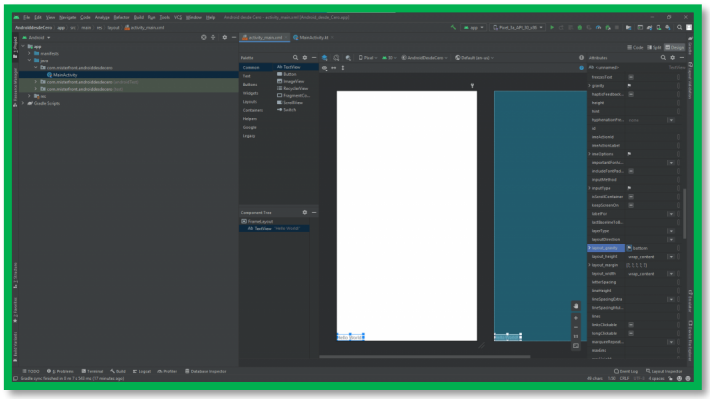
Nos desplegara todas las opciones disponibles para posicionar nuestro elemento en la pantalla



Elegiremos la opción “bottom” teniendo en cuenta que las podemos combinar, ejemplo: buttom y center_horizontal para mostrar nuestro TextView centrado al final de la pantalla. Este comportamiento se aplica para otros elementos como botones, imágenes, entre otros.



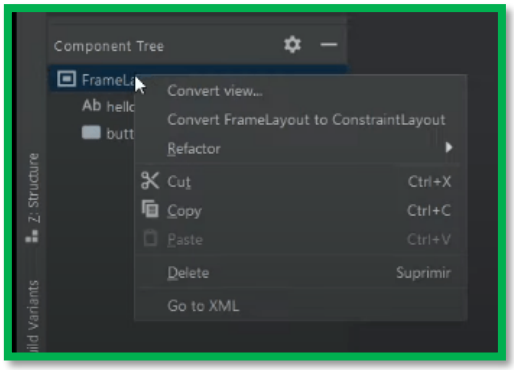
para que nuestro TextView se ponga al final de la pantalla.

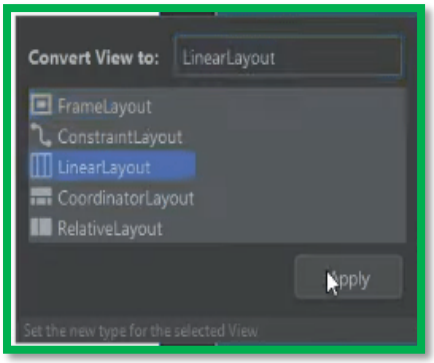


LinearLayout

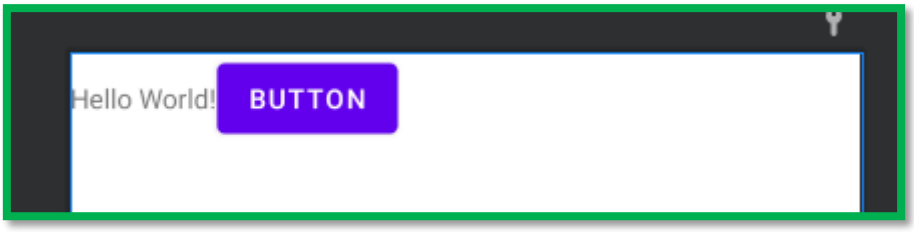
Este layout lo que hará es organizar nuestras vistas, una al lado de la otra, de izquierda a derecha como si se tratase de columnas con la orientación horizontal y con la orientación vertical pasarían hacer como filas.

Cambiaremos un poco el ejemplo anterior y los pasos ya se te harán familiares, convertiremos el FrameLayout en LinearLayout.

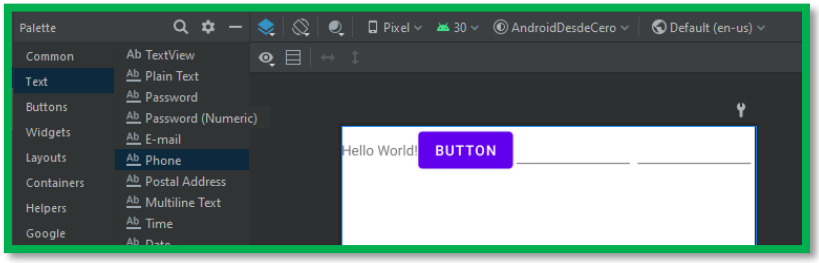




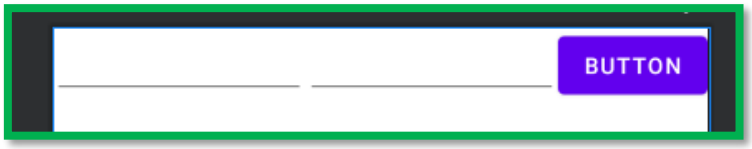
Ahora lo que tenemos que hacer es quitarles a nuestros elementos TextView y Bottom los atributos de layout_gravity para que queden por defecto y notaras que estos elementos se ponen uno al lado del otro.



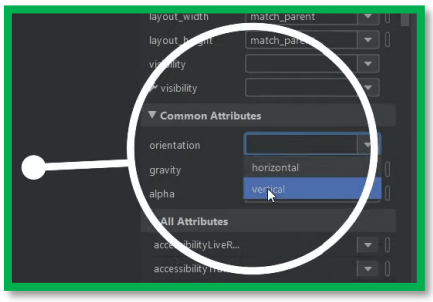
Como siguiente paso agregaremos dos elementos más, uno de tipo E-mail y el otro de tipo Phone, arrastrando uno al lado del otro.



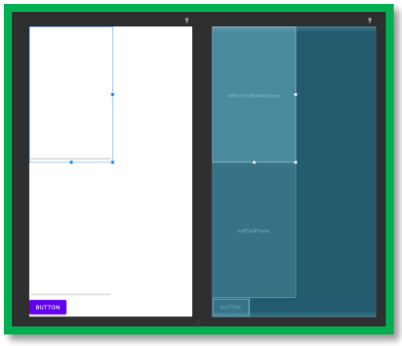
Por consiguiente, haremos dos cosas primero eliminaremos nuestro TextView y pondremos nuestro botón al final.



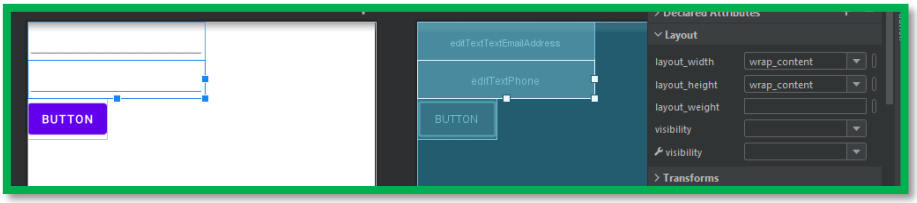
Luego procederemos a poner nuestra vista de forma vertical para que los elementos estén uno encima del otro de arriba hacia abajo, lo haremos seleccionando primero nuestro LinearLayout, por consiguiente yendo a los atributos de este Layout y seleccionando Common Attributes>Orientation ya habremos cambiado nuestra orientación.



La pantalla al momento de hacer este cambio aparece de la siguiente manera.

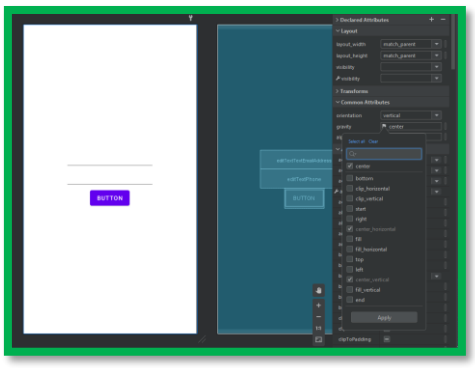


Esto pasa porque hay unas propiedades que se agregaron a los elementos de tipo texto que son el E-mail y el Phone. Para quitar la propiedad solo tenemos que ir a los atributos de cada elemento y dejar en blanco el campo layout_weight.



Lo que queremos ahora es centrar nuestros elementos pero en vez de usar layout_gravity ya que identifica lo que hacen los componentes con respecto al ViewGroup, o sea Layout padre, lo que vamos hacer es el inverso; Vamos a utilizar el gravity del padre, para decirle todo el contenido que tengas dentro de mi mismo, colócalo en equis posición.

En las propiedades del LinearLayout ahora iremos a gravity y seleccionaremos center.

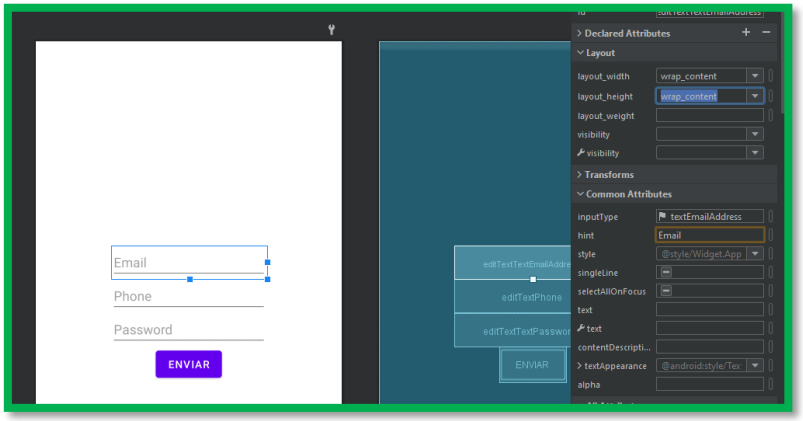


Cambiar valores a nuestros elementos:Una forma sencilla para el botón es ir a sus atributos y seleccionar el que dice text, ahí podemos cambiar el texto por defecto por “enviar” en este caso; para el texto de tipo E-mail y Phone vamos a los atributos y escribimos en el campo hint. Todos estos campos de texto son de tipo EditText.

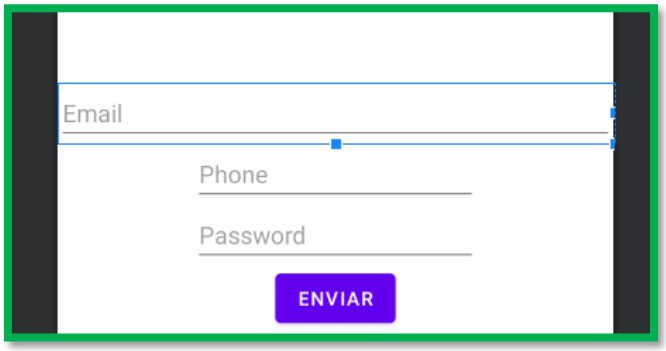
Por ultimo vamos a ver tres atributos:

- layout_width: es el ancho del elemento y puede tener dos propiedades wrap_content y match_parent.
- layout_height: es el alto del elemento y puede tener dos propiedades wrap_content y match_parent.
- layout_weight: aquí repartimos la porción de espacio que cada elemento ocupara.

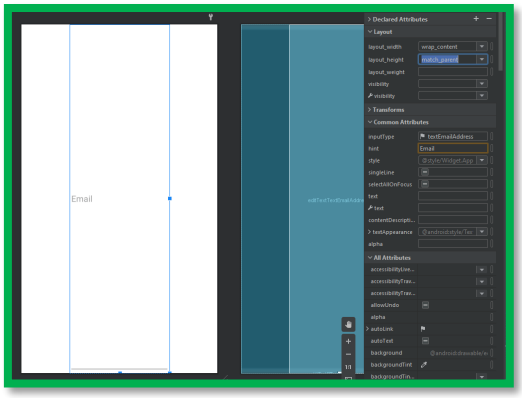
El wrap_content nos ocupara el espacio mínimo del elemento como ya hemos visto por defecto, esto se aplica tanto para el widht como el heigth.



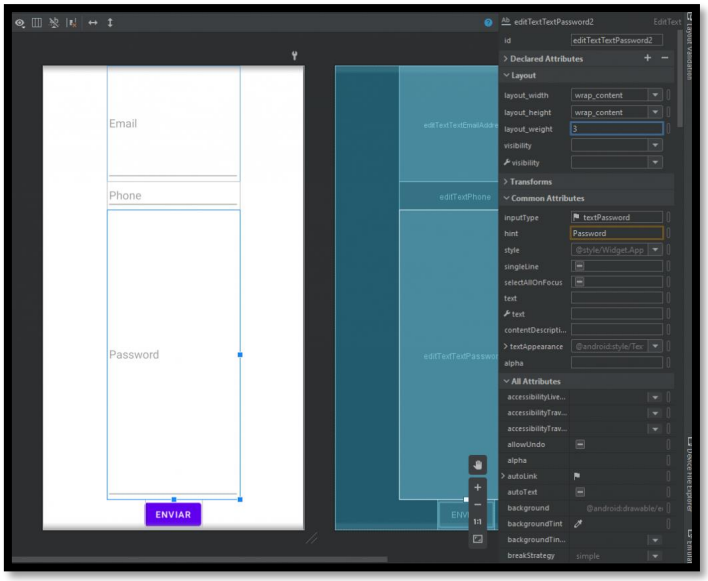
El match_parent es el contrario del wrap_content ya que buscara ocupar todo el espacio disponible para el widht y el heigth.



En LinearLayout no es muy recomendable usarlo porque empujaría al resto de elementos fuera de la pantalla.



La última propiedad vista en este apartado es la del layout_weight que dependiendo del valor que se le dé repartirá los espacios. Vamos a ver un ejemplo si le damos el valor de 1 y a otro elemento el valor de 3. Lo que sucederá es que se distribuyó al elemento con un mayor valor.



Paso 2: Diseñar la interfaz de usuario con EditText, Button y ListView en Android Studio

Diseñar la interfaz de usuario en XML:

- Crear un archivo de diseño con la extensión .xml.
- Utilizar elementos XML para definir los elementos de la interfaz de usuario (EditText, Button, ListView).
- Personalizar los atributos de los elementos para ajustar su apariencia y comportamiento.

1. Elementos básicos

EditText

- Permite al usuario ingresar texto.
- Atributos importantes:
- android:inputType: Tipo de teclado (texto, número, etc.).
- android:hint: Texto de sugerencia que se muestra cuando el campo está vacío.

```
<EditText
    android:id="@+id/editTextNombre"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="Ingrese su nombre"
    android:inputType="text" />
```

- android:id="@+id/editTextNombre": Asigna un ID único al EditText
- android:layout_width="match_parent": El ancho del EditText será igual al ancho de su contenedor.
- android:layout_height="wrap_content": El alto del EditText se ajustará al contenido.
- android:hint="Ingrese su nombre": Texto de sugerencia que se muestra
- android:inputType="text": Tipo de teclado que se mostrará al

Button:

- Permite al usuario iniciar una acción.
- Atributos importantes:
- android:text: Texto que se muestra en el botón.
- android:onClick: Método que se ejecuta cuando se hace clic en el botón.

```
<Button
    android:id="@+id/buttonEnviar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Enviar" />
```

- android:id="@+id/buttonEnviar": Asigna un ID único al Button
- android:layout_width="wrap_content": El ancho del Button
- android:layout_height="wrap_content": El alto del Button
- android:text="Enviar": Texto que se muestra en el botón.

ListView:

- Muestra una lista de elementos desplazables.
- Atributos importantes:
- android:layout_width: Ancho de la lista.
- android:layout_height: Alto de la lista.
- android:adapter: Adaptador que proporciona datos a la lista.

```
<ListView
    android:id="@+id/listViewDatos"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

- android:id="@+id/listViewDatos": Asigna un ID único al ListView
- android:layout_width="match_parent": El ancho del ListView será igual al ancho de su contenedor.
- android:layout_height="match_parent": El alto del ListView será igual al alto de su contenedor.

Paso 3: Implementar la lógica para agregar nuevas tareas

Para implementar la lógica de agregar nuevas tareas realizaremos cambios en el activity_main Este proceso incluye obtener el texto ingresado por el usuario, verificar que no esté vacío, crear una nueva tarea, agregarla a una lista existente, actualizar la interfaz de usuario (UI) y almacenar las tareas para su persistencia.

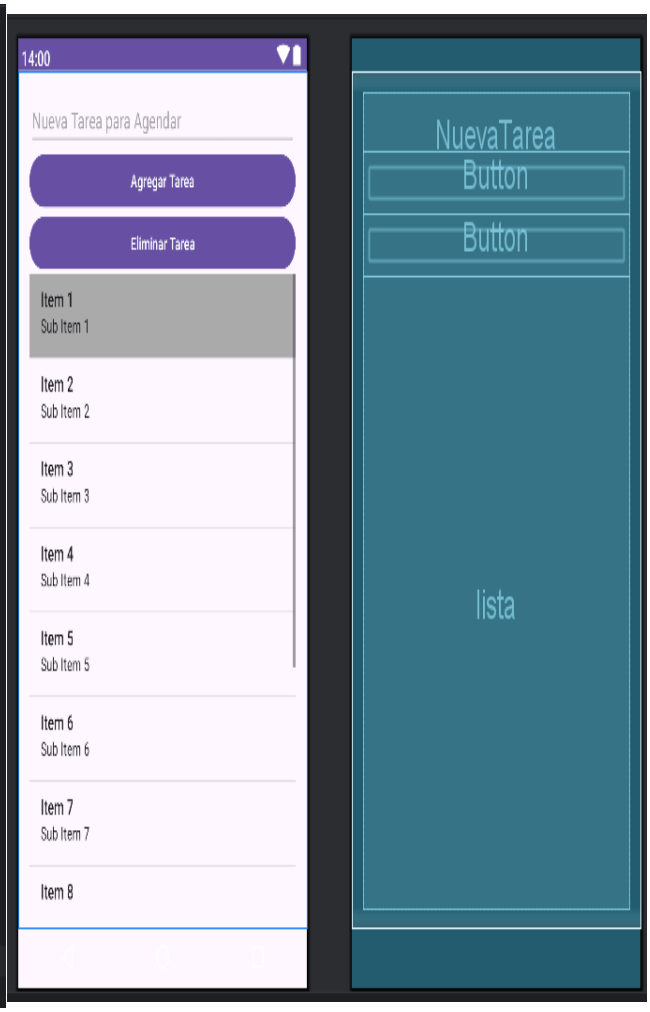
```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp">

    <EditText
        android:id="@+id/NuevaTarea"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Nueva Tarea para Agendar"
        android:inputType="text" />

    <Button
        android:id="@+id/buttonTarea"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Agregar Tarea" />

    <Button
        android:id="@+id/EliminarTarea"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Eliminar Tarea" />

    <ListView
        android:id="@+id/lista"
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:choiceMode="multipleChoice" />
</LinearLayout>
```



A continuación, agregamos la lógica de manejo dentro del MainActivity

Primero dándole función al botón para agregar

```
buttonAddTask.setOnClickListener {
    val taskTitle = editTextTask.text.toString()
    if (taskTitle.isNotEmpty()) {
        val newTask = Task(taskTitle, isCompleted: false)
        taskList.add(newTask)
        editTextTask.text.clear()
        adapter.notifyDataSetChanged()
        saveTasks()
    } else{
        Toast.makeText( context: this, text: "Por favor, ingresa una tarea", Toast.LENGTH_SHORT).show();
    }
}
```

Esta parte del código se encarga de manejar el evento de clic en el botón `buttonAddTask`, verificando si el campo de entrada de texto no está vacío, y agregando una nueva tarea a la lista si es válido. Si el campo de entrada está vacío, muestra un mensaje al usuario indicando que debe ingresar una tarea.

Agregamos una función llamada saveTasks que se encarga de guardar la lista de tareas actuales en `SharedPreferences` para que persistan entre sesiones de la aplicación

```
private fun saveTasks() {
    val editor = sharedPreferences.edit()
    val taskSet = taskList.map { "${it.title}:${it.isCompleted}" }.toSet()
    editor.putStringSet("tasks", taskSet)
    editor.apply()
}
```

Agregamos una función llamada loadTasks que se encarga de cargar las tareas previamente guardadas desde `SharedPreferences` y actualizar la lista de tareas (`taskList`) en la aplicación

```
private fun loadTasks() {
    val taskSet = sharedPreferences.getStringSet("tasks", setOf())
    if (taskSet != null) {
        taskList.clear()
        for (taskString in taskSet) {
            val parts = taskString.split( ...delimiters: ":")
            if (parts.size == 2) {
                taskList.add(Task(parts[0], parts[1].toBoolean()))
            }
        }
        adapter.notifyDataSetChanged()
    }
}
```

Paso 4: implementar lógica de eliminar tareas

Al igual que como hicimos para implementar la logica de implementar una nueva tarea haremos cambios en el activity_main agregando un button que se encargara de eliminar

```
<Button
    android:id="@+id/EliminarTarea"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="Eliminar Tarea" />
```

A continuación, agregamos la lógica de manejo dentro del MainActivity

```
buttonDeleteTask.setOnClickListener {
    val itemsToRemove = taskList.filter { it.isSelected }.toMutableList()

    if (itemsToRemove.isNotEmpty()) {
        taskList.removeAll(itemsToRemove)
        adapter.notifyDataSetChanged()
        saveTasks()
    } else {
        Toast.makeText( context: this,  text: "Seleccione las tareas a eliminar", Toast.LENGTH_SHORT).show()
    }
}
```