Introducción a Android - Parte II

Layouts y ViewGroups

Competencias:

- Estudiar las jerarquía de vistas, contenedores y su ordenamiento en una aplicación
- Principales layouts y grupos de vistas
- Crear un archivo layout
- Reemplazar desde nuestra actividad la instancia de un archivo layout por otro

Introducción

A continuación aprenderemos el detalle de un diseño (layout) y sus principales grupos de vistas (ViewGroups), que son a su vez, contenedores de vistas (Views).

Es muy importante dominar las distintas vistas y grupos, para construir nuestras pantallas dentro de la aplicación, teniendo la agilidad de crear, modificar y asignar espacios dentro de la pantalla. Por lo general, toma tiempo la construcción de las vistas, por lo tanto, para no retrasarnos en los proyectos es importante ser hábiles en esta etapa.

Layouts

Un diseño define la estructura visual para una interfaz de usuario, como la IU para una actividad o widget de una app. Puedes declarar un diseño de dos maneras:

- Declarar elementos de la IU en XML. Android proporciona un vocabulario XML simple que coincide con las clases y subclases de vistas, como las que se usan para widgets y diseños.
- Crear una instancia de elementos del diseño en tiempo de ejecución. Tu aplicación puede crear objetos View y ViewGroup (y manipular sus propiedades) programáticamente.

En palabras más sencillas, los layouts o diseños en nuestra aplicación Android, son nuestros archivos declarados en XML, donde agregamos vistas y grupos de vistas asociados a nuestra interfaz de usuario.

La ventaja de declarar tu IU en XML es que te permite separar mejor la presentación de tu aplicación del código que controla su comportamiento. Tus descripciones de la IU son externas al código de tu aplicación, lo que significa que puedes modificarlo o adaptarlo sin tener que modificar el código fuente y volver a compilar. Por ejemplo, puedes crear diseños XML para diferentes orientaciones de pantalla, diferentes tamaños de pantalla de dispositivos y diferentes idiomas. Además, declarar el diseño en XML facilita la visualización de la estructura de tu IU, de modo que sea más fácil depurar problemas.

En la imagen 1 se observa la jerarquía de vistas con parámetros de diseño asociados con cada vista, en donde el LinearLayout y el ConstraintLayout son ViewGroups.

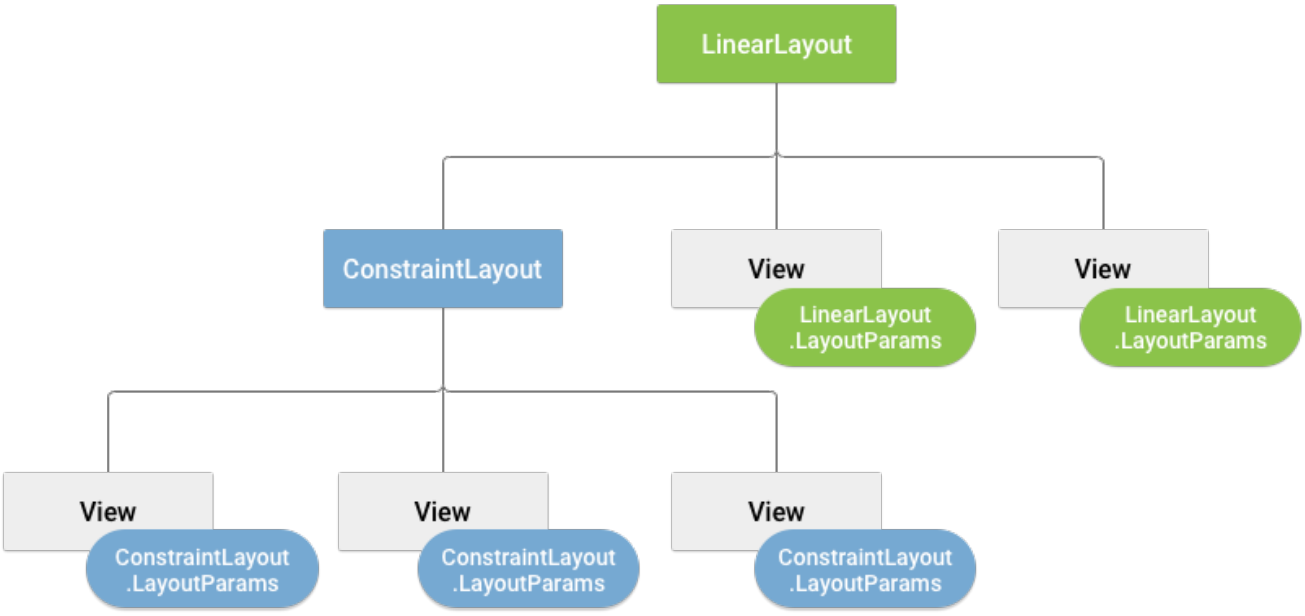


Imagen 1. Jerarquía de los Layouts.

Al usar vocabulario XML de Android, puedes crear rápidamente diseños de IU y de los elementos de pantalla que contienen, de la misma manera que creas páginas web en HTML, con una serie de elementos anidados.

Cada archivo de diseño debe contener exactamente un elemento raíz, que debe ser un objeto View o ViewGroup. Una vez que hayas definido el elemento raíz, puedes agregar widgets u objetos de diseño adicionales como elementos secundarios para crear gradualmente una jerarquía de vistas que defina tu diseño.

Revisemos un archivo de diseño del visto anteriormente, el que nos muestra el estado de la conexión a internet. Este código se encuentra en el archivo `activity_home_detail.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.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"
    tools:context=".HomeDetailActivity">

    <androidx.appcompat.widget.AppCompatTextView
        android:id="@+id/connectionStatus"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="25dp"
        android:text="Conexión establecida"
        app:layout_constraintBottom_toTopOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="parent"
        android:gravity="center"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

Veamos el detalle del código, profundizando en sus principales atributos:

- `androidx.constraintlayout.widget.ConstraintLayout` es nuestro ViewGroup contenedor de la vista AppCompatTextView.

  Este ViewGroup posee los siguientes parámetros:

  - `android:layout_width="match_parent"`

    Ocupa todo el espacio de ancho disponible de la Pantalla.
  - `android:layout_height="match_parent"`

    Ocupa todo el espacio de altura disponible de la Pantalla.
  - `tools:context=".HomeActivity"`

    Indica cuál es la actividad asociada a este elemento

- `androidx.appcompat.widget.AppCompatTextView` es la View que muestra un mensaje del estado de la conexión con un fondo verde o rojo dependiendo de su resultado, como se muestra en las imágenes 31 y 32 de Introducción a Andorid - Parte I.

  Este posee los siguientes parámetros:

  - `android:layout_width="match_parent"`

    Ocupa todo el espacio de ancho disponible de la Pantalla.

  - `android:layout_height="wrap_content"`

    Ocupa solo el espacio de altura que requiera el texto del mensaje en la Pantalla.

  - `app:layout_constraintBottom_toTopOf="parent"`

    `app:layout_constraintLeft_toLeftOf="parent"`

    `app:layout_constraintRight_toRightOf="parent"`

    `app:layout_constraintTop_toBottomOf="parent"`

    Estos cuatro parámetros hacen el trabajo de ubicar la vista de texto exactamente en el centro de la pantalla, indicando cada posición desde la superior e inferior de la pantalla, hasta la izquierda y derecha de su ancho total. Constraint es un nuevo ViewGroup que te permite trabajar con vistas responsive design de una forma más amigable y eficiente.

Ejercicio 7: Sustituir el layout de la actividad HomeDetailActivity por uno nuevo.

1. Nos ubicamos en la vista de navegación del proyecto y creamos un nuevo archivo de recurso layout como lo muestra la imagen 2.

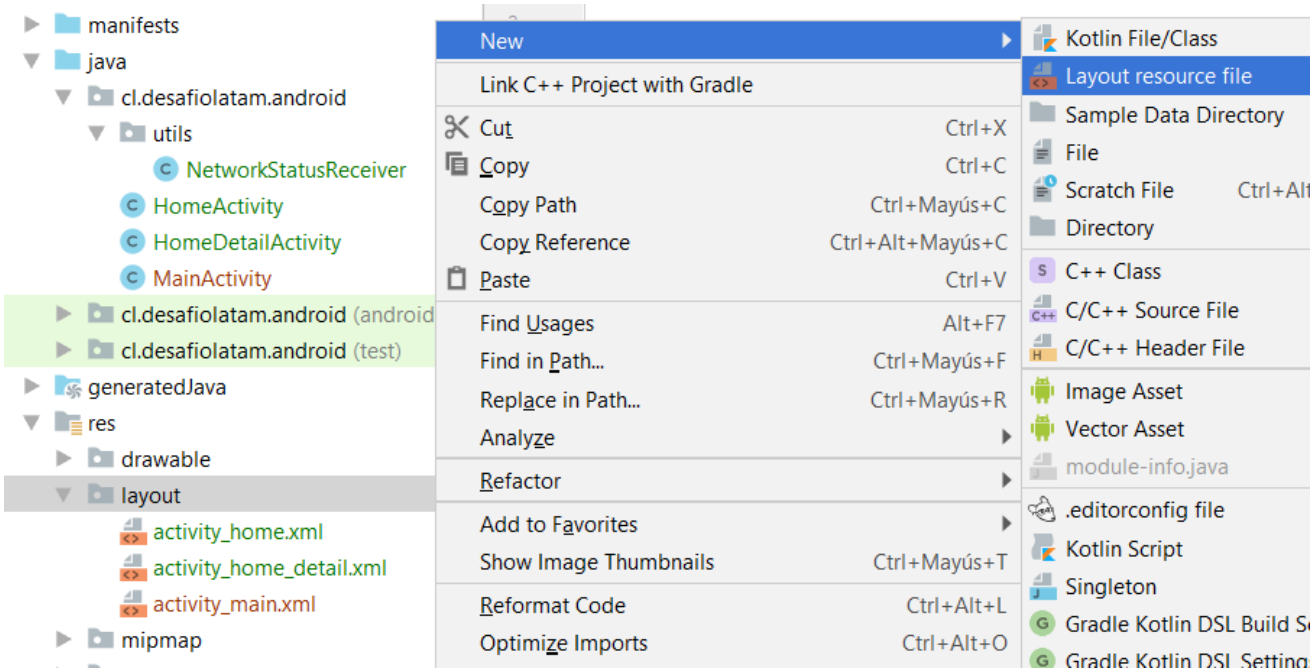


Imagen 2. Crear nuevo recurso layout.

2. Ingresamos el nombre y el ViewGroup base `ConstraintLayout` como lo indica la imagen 3.

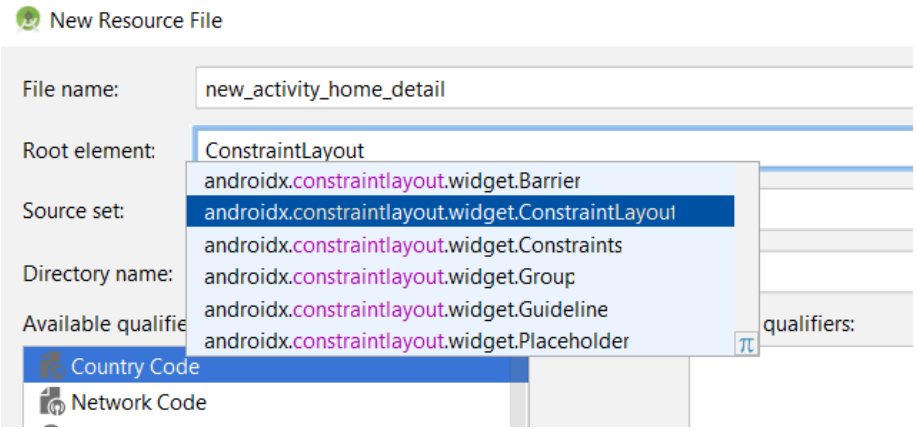


Imagen 3. Buscar ContraintLayout.

3. Siempre que les solicite agregar a git un archivo nuevo, háganlo, así mantienen la versión de cada archivo. El mensaje que aparece en el ide como referencia en la imagen 4.

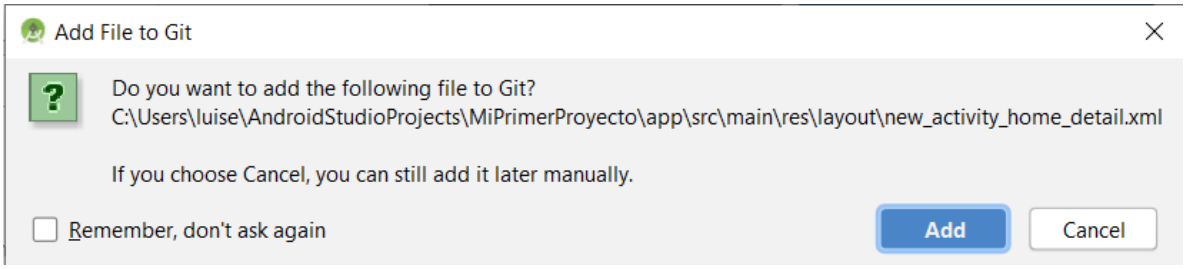


Imagen 4. Agregar a Git

4. En el archivo nuevo, agregaremos el siguiente código, que reutilizamos del archivo activity_home_detail.xml, y agregamos la imagen también reutilizada del archivo activity_home.xml con un parámetro destacado que posiciona nuestra imagen sobre el mensaje del estado de la conexión:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.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"
    tools:context=".HomeDetailActivity">
```

```
    <androidx.appcompat.widget.AppCompatImageView
        android:id="@+id/imgSaludo"
        android:layout_width="100dp"
        android:layout_height="100dp"
        android:contentDescription="saludo home"
        android:src="@drawable/internet_service"
        app:layout_constraintBottom_toTopOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@id/connectionStatus"
        />

    <androidx.appcompat.widget.AppCompatTextView
        android:id="@+id/connectionStatus"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="25dp"
        android:text="Conexión establecida"
        app:layout_constraintBottom_toTopOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="parent"
        android:gravity="center"/>

</androidx.constraintlayout.widget.ConstraintLayout>
```

5. Abrimos nuestro archivo `HomeDetailActivity.java` y modificamos el layout al que hace referencia en su método de clase `onCreate()`, por el que recientemente hemos creado, como lo muestra la imagen 5.

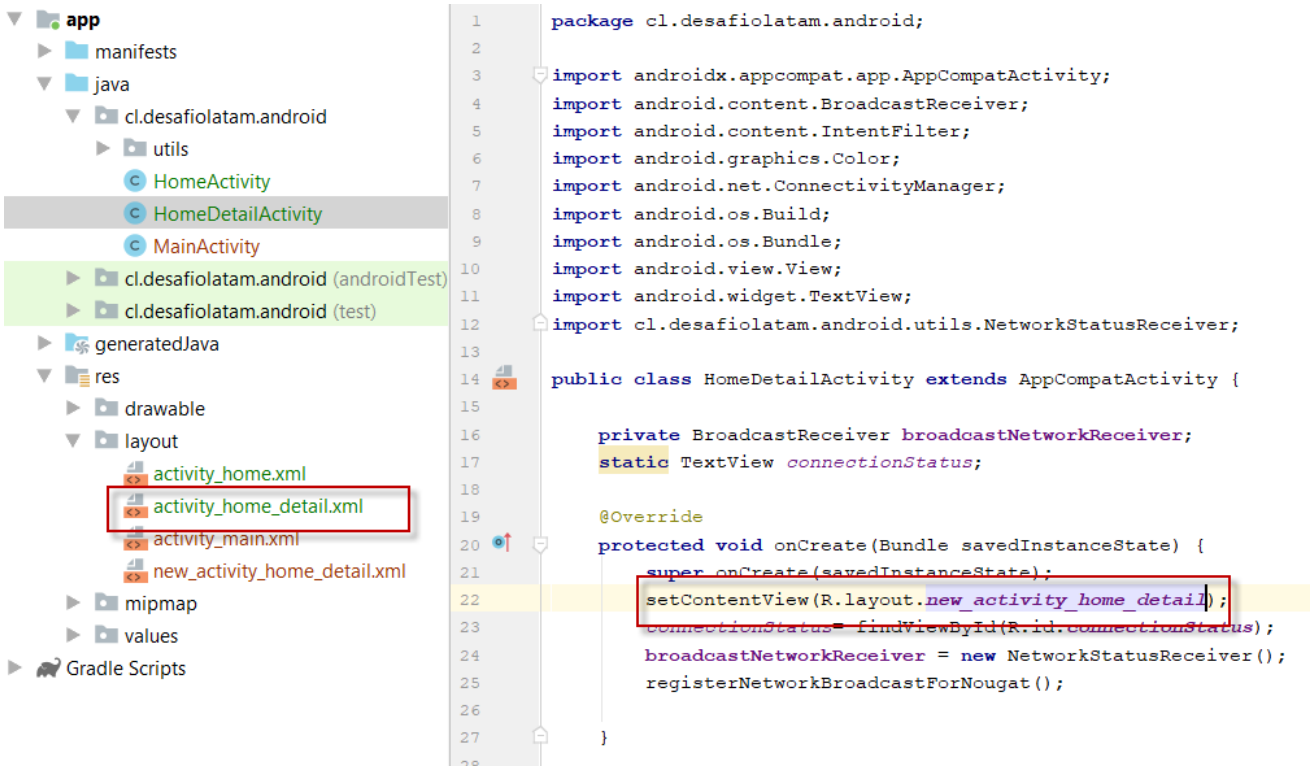


Imagen 5. Cambiamos referencia en el mñetodo onCreate()

6. El resultado debería de ser el que nos muestra la imagen 6.

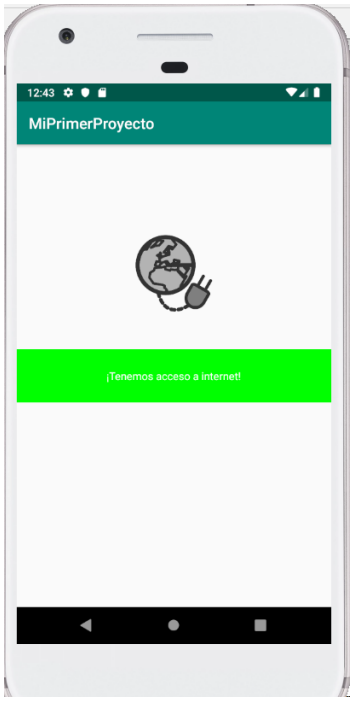


Imagen 6: Resutado ejercicio.

ViewGroups

Los ViewGroups como hemos indicado anteriormente son vistas principales que agrupan múltiples vistas dentro de un layout de actividad en nuestra aplicación. Aprender a trabajar con distintos ViewGroup es muy importante para crear aplicaciones que se adapten a cualquier tamaño de pantalla, manteniendo el orden de sus elementos internos.

Los ViewGroups más usados son:

- **ConstraintLayout**, siendo el más común en la actualidad por su flexibilidad.
- **GridLayout**, cuya función principal es separar en celdas tipo lista, el contenido que deseamos mostrar.
- **TableLayout**, es un ViewGroup que básicamente alinea sus vistas internas en columnas o filas.
- **RelativeLayout**, era muy común antes de la llegada del ConstraintLayout, y permite posicionar a sus vistas internas refiriendo a otra de sus vistas, es decir, una vista de imagen puede posicionarse debajo de la vista de texto.
- **LinearLayout**, te permite alinear vertical u horizontalmente tus vistas internas. Es importante no abusar de la implementación de muchos LinearLayout en una vista, dado que puede llegar a hacerse pesada y lenta la aplicación.

Ejercicio 8: Reemplazemos el ConstraintLayout del archivo activity_home.xml, por un LinearLayout.

1. Abrimos nuestro archivo activity_home.xml ubicado en la carpeta /res/layout, eliminando los siguiente bloques de código al inicio y final de nuestro archivo respectivamente:

```
<androidx.constraintlayout.widget.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"
   tools:context=".HomeActivity">
```

```
 </androidx.constraintlayout.widget.ConstraintLayout>
```

2. A continuación agregamos el siguiente código al inicio de nuestro archivo, por supuesto después de nuestro header `<?xml>`, de la siguiente manera:

```
<androidx.appcompat.widget.LinearLayoutCompat
android:layout_width="match_parent"
   android:layout_height="match_parent"
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   android:gravity="center"
   android:orientation="vertical"
   >
```

Y agregamos la etiqueta de cierre correspondiente al final del archivo:

```
</androidx.appcompat.widget.LinearLayoutCompat>
```

3. Ahora necesitaremos modificar nuestra imagen y texto contenidos en el ViewGroup, destacando los cambios requeridos para su entendimiento de la siguiente forma:

```
<androidx.appcompat.widget.AppCompatImageView
   android:id="@+id/imgSaludo"
   android:layout_width="200dp"
   android:layout_height="200dp"
   android:contentDescription="saludo home"
   android:src="@drawable/internet_service"
   android:layout_gravity="center"
   />

<androidx.appcompat.widget.AppCompatTextView
   android:id="@+id/txtSaludo"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:text="Está es nuestra actividad HomeActivity con un ViewGroup
LinearLayout base"
   android:textSize="24sp"
   android:padding="20sp"
   android:gravity="center"
   />

<androidx.appcompat.widget.AppCompatButton
   android:id="@+id/btnCheckInternet"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:text="VERIFICAR CONEXION"
```

```
    android:onClick="openActivityCheckConnection"
    android:background="@color/colorPrimary"
    android:textColor="#FFFFFF"
    android:textSize="24dp"/>
```

4. Del código anterior se destaca lo siguiente:

- El cambio de width de 0dp a match_parent en el TextView.
- El cambio de width de 0dp a 200 density pixels para el caso del ImageView.
- La eliminación de parametros asociados al ConstraintLayout antiguo, y agregando android:gravity y layout_gravity para el posicionamiento de las vistas en el centro de la pantalla.

Estos cambios son requeridos para adaptar nuestras vistas a su ViewGroup padre, con esto nos referimos a que las vistas ocupan espacios y acciones dentro de su grupo de vista padre, quien es el que las contiene. También un Layout puede contener otro layout declarando una etiqueta `<include> </include>`, como por ejemplo:

```
<include layout="@layout/mi_sengundo_layout" />
```

Luego de estos cambios deberíamos de notar que la vista no sufrió cambios mayores manteniendo su aspecto y con un mensaje distinto, como lo muestra la imagen 7.

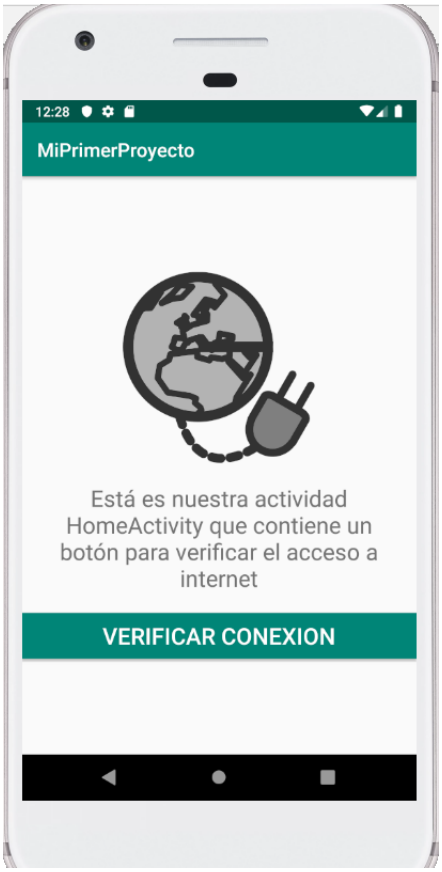


Imagen 7. Resultado del ejercicio

Views

Competencias:

- Conocer los tipos de vistas más utilizados
- Agregar otras vistas a nuestro archivo layout
- Trabajar sobre el archivo strings.xml

Introducción

A continuación profundizaremos todo lo visto anteriormente sobre las vistas, crearemos nuevas vistas sobre los diseños existentes y explicaremos cuales son las más utilizadas, además de sus detalles.

Views

La clase View representa el componente básico para los componentes de la interfaz de usuario. Una vista ocupa un área rectangular en la pantalla y es responsable del manejo de eventos y dibujos. View es la clase base para los widgets, que se utilizan para crear componentes de IU interactivos (botones, campos de texto, etc.).

La subclase ViewGroup es la clase base para los diseños(layouts), que son contenedores invisibles que contienen otras Vistas (u otros ViewGroups) y definen sus propiedades de diseño.

Las vistas más recurrentes para los programadores serían las siguientes:

- **TextView**: es la vista que nos permite mostrar una cadena de texto (mensaje).

  En la imagen 8 observa un TextView mostrando un mensaje.

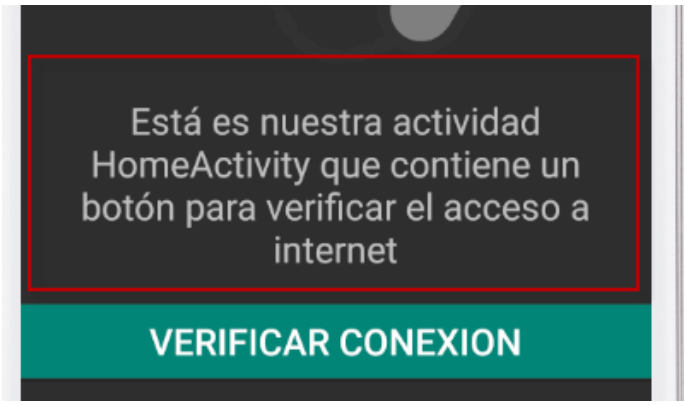


Imagen 8. TextView mostrando un mensaje.

- **EditText**: esta vista nos permite ingresar datos desde el teléfono, es decir son un input que recoge información del usuario. Ver imagen 9 de referencia.

Imagen 9. EditText indicando una fecha.

- **Button**: simplemente nos lleva a una acción, mostrando un mensaje referencial. Ver imagen 10 de referencia.

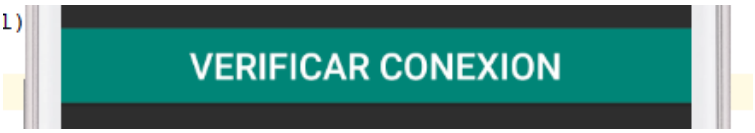


Imagen 10. Button.

- **ImageView**: como hemos visto, nos permite mostrar imágenes en nuestro diseño (layout). Ver imagen 11 de referencia.

Imagen 11. ImageView

- **ImageButton**: Igual que el ImageView, pero este hace la imágen clickable, es decir, al presionar sobre la imagen esta nos lleva a una acción.
- **CheckBox**: muy utilizada en formularios para selección múltiple como las respuestas a una pregunta. Ver imagen 12 de referencia.

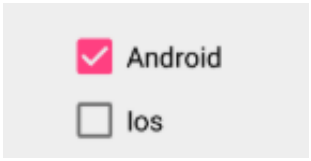


Imagen 12. CheckBox

- **Radio button**: es típico círculo de selección que usualmente es utilizado en los sistemas para escoger el sexo de una persona, de entre masculino y femenino. Ver imagen 13 de referencia.

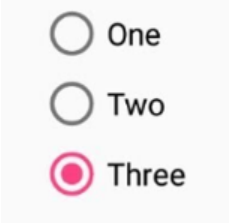


Imagen 13. Radio button

- **RadioGroup**: es la vista que agrupa los radiobuttons.
- **ListView**: es una vista que agrupa vistas en forma de lista.
- **Spinner**: el spinner es parecido a un combobox tradicional pero en este caso su selección puede variar sobre la misma posición. Ver imagen 14 de referencia.

Imagen 14. Spinner

Ejercicio 9: Agregar una vista tipo spinner a nuestro new_activity_home_detail layout.

1. Abrir el archivo de layout `new_activity_home_detail.xml`, agregar el siguiente código antes de la etiqueta de cierre del ConstraintLayout.

```
<androidx.appcompat.widget.AppCompatTextView
    android:id="@+id/tituloSpinner"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:padding="25dp"
    android:text="EJEMPLO DE VISTA SPINNER, escoge una opción de valoración..."
    app:layout_constraintBottom_toTopOf="@id/connectionStatus"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toBottomOf="parent"
    android:gravity="center"/>
```

```
<Spinner
    android:id="@+id/spinner1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    app:layout_constraintTop_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintBottom_toTopOf="@id/tituloSpinner"
    android:entries="@array/valoracion_arrays"
    android:prompt="@string/inicio_spinner" />
```

2. Del código anterior es importante destacar las propiedades entries y prompt de la vista Spinner, donde entries indica un arreglo de datos, es decir, nuestra lista de selección, y prompt que viene a ser la cadena de texto inicial que se mostrará antes de una selección.
3. Para crear el objeto `valoracion_arrays` e `inicio_spinner` abriremos nuestro archivo strings.xml de nuestra carpeta /res/values, en donde agregaremos el siguiente código antes de la etiqueta de cierre `:

```
<string name="inicio_spinner">¿qué te ha parecido este ejemplo?</string>

<string-array name="valoracion_arrays">
    <item>Muy bueno</item>
    <item>Bueno</item>
    <item>Regular</item>
    <item>Malo</item>
    <item>Muy Malo</item>
</string-array>
```

4. El ejercicio nos debe mostrar una pantalla de detalle con dos nuevas vistas, una de texto informativo y otra, un spinner de selección como lo muestran las imágenes 15 y 16.

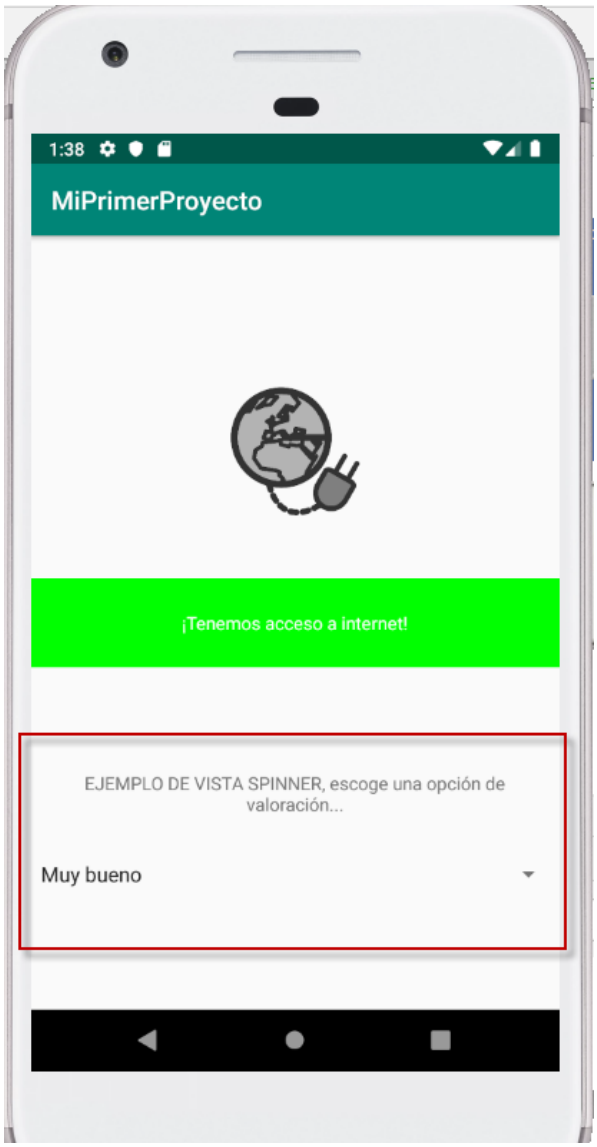


Imagen 15. Texto informativo.

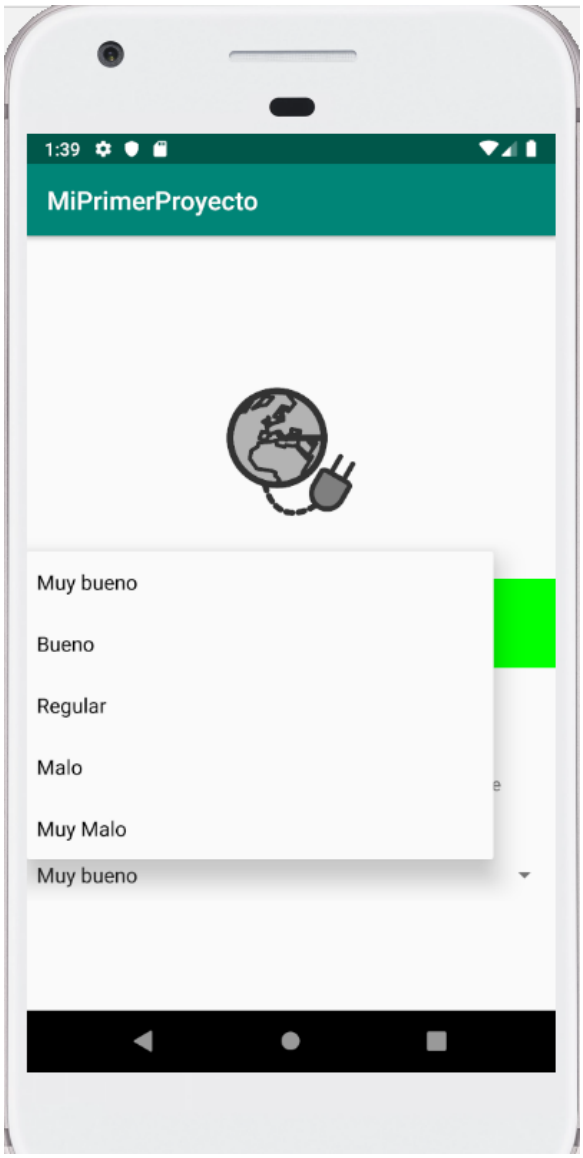


Imagen 16.Spinner.

Material Design y Widgets

Competencias:

- Conocer themes y widgets
- Entendiendo patrones UX y principios de material design
- Instalación de plugins
- Modificación de themes generales de un proyecto

Introducción

Desarrollaremos aún más nuestras vistas con los patrones de diseño material design, los cuales incluyen recomendaciones de diseño, imágenes y convenciones comunes que nos ayudarán a crear aplicaciones expresivas que funcionan a la perfección en cualquier dispositivo. A medida que los usuarios se familiaricen con este lenguaje visual, esperando encontrarlo en las aplicaciones. Este diseño te permitirá crear aplicaciones que resulten atractivas al instante, mejoren la usabilidad y aumenten la participación y retención de los usuarios.

Material Design

Material Design es un lenguaje visual que sintetiza los principios clásicos del buen diseño con la innovación de la tecnología y la ciencia. Material design pasa a ser un guideline de diseño que se asemeja al sistema operativo android, para mejorar y hacer más familiar la experiencia UX.El diseño de materiales (material design) es también una guía completa para el diseño visual, de movimiento e interacción en plataformas y dispositivos. Para usar el diseño de material en sus aplicaciones de Android, siga las pautas definidas en la especificación de diseño de materialy use los nuevos componentes y estilos disponibles en la biblioteca de soporte de diseño de material.

Android proporciona las siguientes funciones para ayudarlo a crear aplicaciones de diseño con material:

- Un tema de aplicación de diseño de materiales (material theme) para personalizar todas tus vistas y widgets de UI (interfaz de usuario).
- Widgets para vistas complejas como listas y tarjetas.
- Nuevas APIs para sombras y animaciones personalizadas.

En las siguientes imágenes, podemos ver un ejemplo de lo que nos propone material design para un listado de eventos tipo card, y con el requerimiento de un menú inferior con un botón float para la acción de editar integrado visualmente.

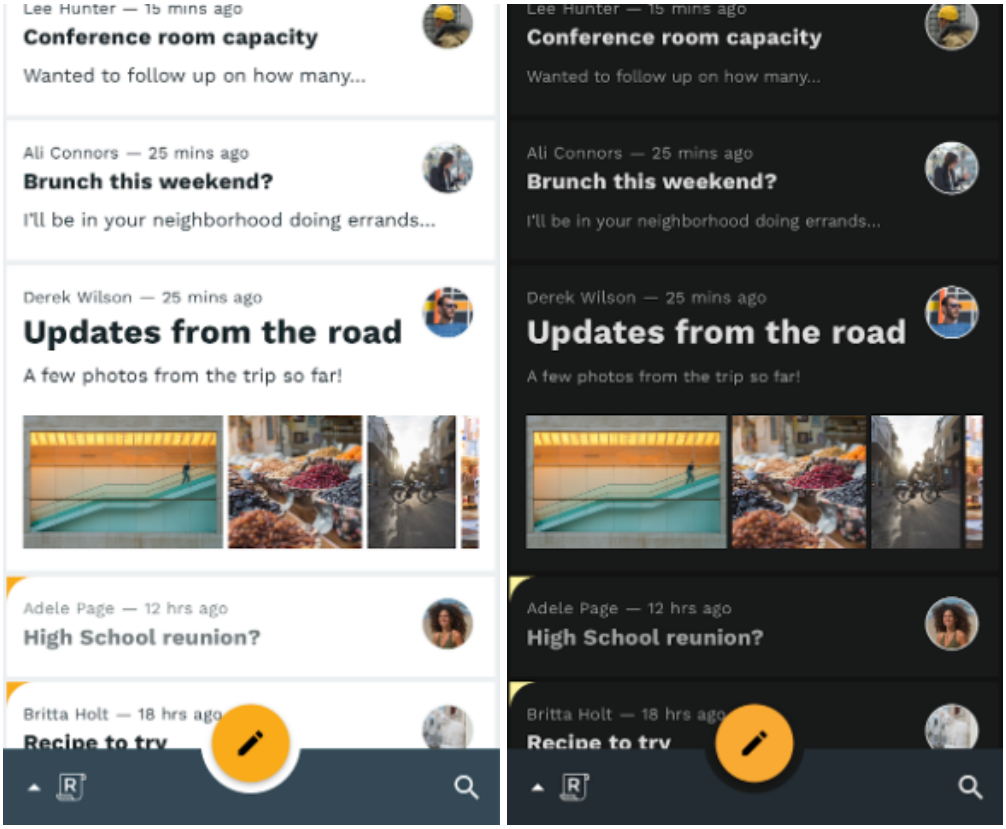


Imagen 17. ejemplos utilizando Material Design

Principios de diseño

Material Design está inspirado en el mundo físico y sus texturas, incluida la forma en que reflejan la luz y proyectan sombras. Las superficies de los materiales re-imaginan los medios de papel y tinta. Expliquemos más en detalle:

- Usa **superficies y sombras** como metáforas. Los elementos de la IU deben incluir superficies con capas individuales apiladas o ubicadas una al lado de la otra. Se deben usar sombras para mostrar qué superficies están delante de otras; esto permite centrar la atención y establecer jerarquías.
- Las **imágenes** no deben tener sangrado completo. Minimiza el espacio y los márgenes entre las fotos, así como los bordes de las pantallas, para obtener una IU más envolvente y llena de contenido.
- Los **colores** deben ser intensos para destacar la marca y los elementos importantes de la IU. Asegúrate de elegir un color primario para los elementos destacados a fin de ofrecer una experiencia coherente en toda la app. Considera usar la **API de paleta** para tomar colores directamente desde el contenido y ofrecer una experiencia más envolvente.

- Se deben usar **métricas y líneas clave**para alinear y ajustar el contenido según una **cuadrícula de base** de 8 dp con el fin de facilitar la lectura y mantener la coherencia visual. El posicionamiento de texto más preciso puede usar una cuadrícula de 4 dp. Cuando corresponda, ciertos elementos, como el texto del cuerpo, las miniaturas, los títulos de las barras de las apps y otros similares, se deben alinear según **las líneas clave**estándar.
- Se debe usar **movimiento significativo**, como **animaciones y transiciones**, para ayudar a guiar y centrar a los usuarios, además de ofrecer una continuidad entre una pantalla y la siguiente. En las narraciones visuales, el movimiento significativo puede proporcionar una experiencia del usuario placentera y contribuir con la continuidad visual.
- Las **interacciones de respuesta**, como la opción básica para dejar comentarios de forma táctil, deben tener un efecto de **ondas en la superficie**. No se suele incluir la función de ondas en los dispositivos anteriores a la versión 5.0, por lo que debes usar el comportamiento predeterminado android:selectableItemBackground. Además de las ondas de la superficie, las superficies interactivas también deben **elevarse al tacto** para encontrarse con el dedo del usuario durante la entrada, y cualquier cambio debe **irradiarse** desde el punto de contacto.

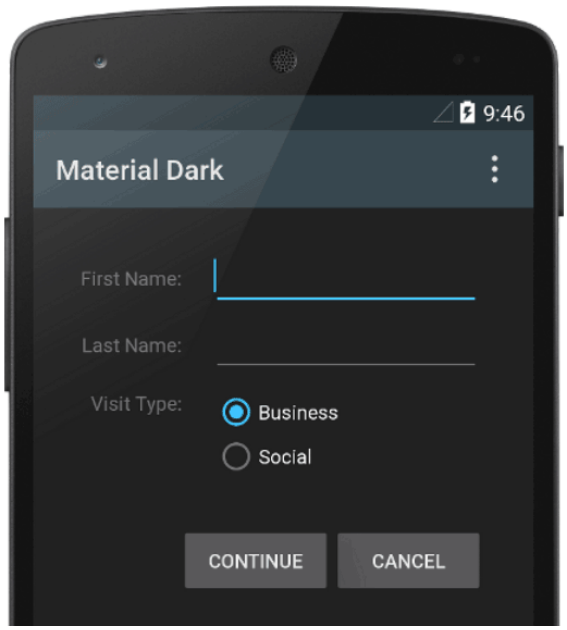
El Objetivo de Material Design

- **Crear:** un lenguaje visual que sintetice los principios clásicos del buen diseño con la innovación y la posibilidad de la tecnología y la ciencia.
- **Unir**: desarrollar un sistema subyacente único que unifique la experiencia del usuario en plataformas, dispositivos y métodos de entrada.
- **Estandarizar:** ampliar el lenguaje visual de material y proporcionar una base flexible para la innovación y la expresión de la marca.

Themes

Un tema es un tipo de estilo que se aplica a toda una aplicación, actividad o jerarquía de vistas, no solo a una vista individual. Cuando aplica su estilo como un tema, cada vista en la aplicación o actividad aplica cada atributo de estilo que admite. Los temas también pueden aplicar estilos a elementos que no se ven, como la barra de estado y el fondo de la ventana.

Los estilos y temas se declaran en un archivo de recursos de estilo en res/values/, normalmente llamado styles.xml. Las siguientes imágenes 18 y 19, nos muestran estilos (themes) aplicados en una misma actividad, la primera denominada Theme.AppCompat, y la segunda Theme.AppCompat.Light.



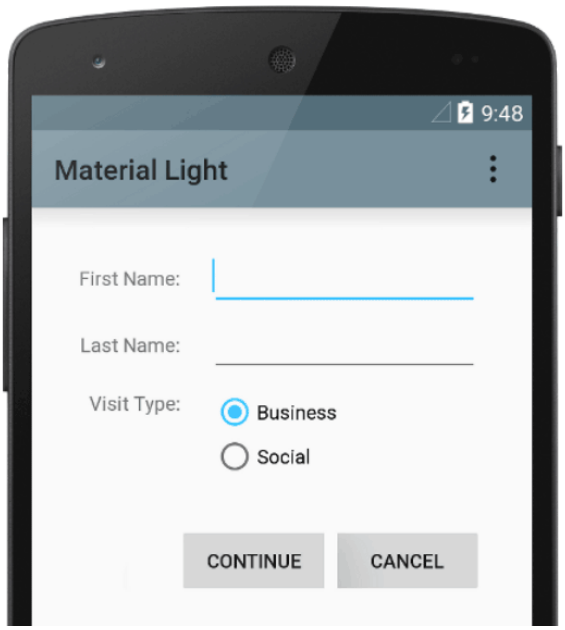


Imagen 18 y 19. Usando distintos themes.

Ejercicio 10: Modificación de estilo material en nuestra aplicación MiPrimerProyecto.

1. Abrimos nuestro archivo styles.xml ubicado en la carpeta /res/values de nuestro proyecto.
2. Editamos la siguiente línea de código:

```
<style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
```

3. Por la siguiente línea de código, con la cual estamos editando nuestro theme material, el cual es cargado desde los repositorios indicados en nuestro archivo build.gradle:

```
<style name="AppTheme" parent="Theme.AppCompat">
```

4. Nuestra MainACtivity se debería ver con un estilo modificado parecido a la imagen 41. Para referencias, ver la imagen 20.

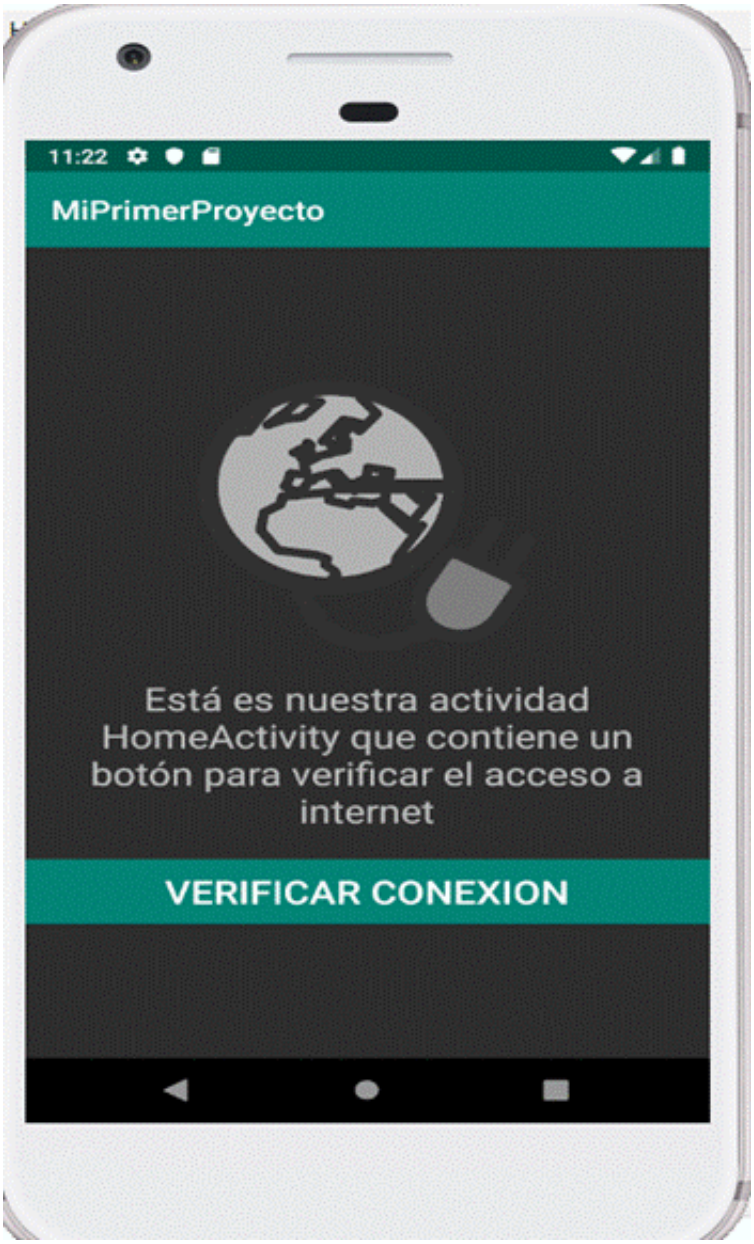


Imagen 19. Estilo cambiado.

Widgets

Los widgets en Material Design muestran información comparativa de los datos y la funcionalidad más importantes de una aplicación. Se pueden entender a los widgets como aplicaciones miniaturas que funcionan dentro de nuestra aplicación principal, también existen componentes más simples como lo es el `FloatingButton, que también son considerados widgets.

Tipos de Widgets

Los widgets muestran el contenido nuevo e interesante de su aplicación de forma consolidada en una pantalla de inicio móvil. Se vinculan a un detalle más rico dentro de la aplicación.Los usuarios pueden interactuar con múltiples widgets, y si es compatible con su tipo de teléfono, cambiar el tamaño de los widgets en los paneles de la pantalla de inicio. Veámos cuáles son los tipos de widgets según material design:

- **App Widgets:** son todos aquellos widgets que se muestran dentro de nuestra aplicación. Haremos un ejemplo del widget **Floating Button**, uno de los más usados en el standard de Material Design y menos complejos de desarrollar para nuestra unidad introductoria. Ver imagen 20.

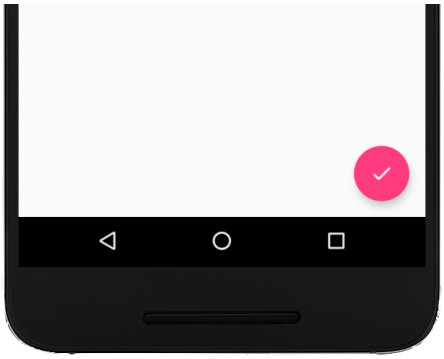


Imagen 20.

- **Information widgets:** los widgets de información muestran algunos elementos de importancia para un usuario y rastrean cómo cambia esa información a lo largo del tiempo, como el clima o las puntuaciones deportivas. Al tocar el widget se inicia la aplicación asociada en una pantalla de detalle. Ver la imagen 21 de referencia.

Imagen 21. Imformation widget

- **CollectionWidgets**: los widgets de colección muestran varios elementos del mismo tipo, como una colección de artículos de una aplicación de noticias. Se centran en dos interacciones:Navegando por una colecciónAbriendo la pantalla de detalles de un elemento. Los widgets de colección pueden desplazarse verticalmente. En la imagen 22 se muestra un collection widget.

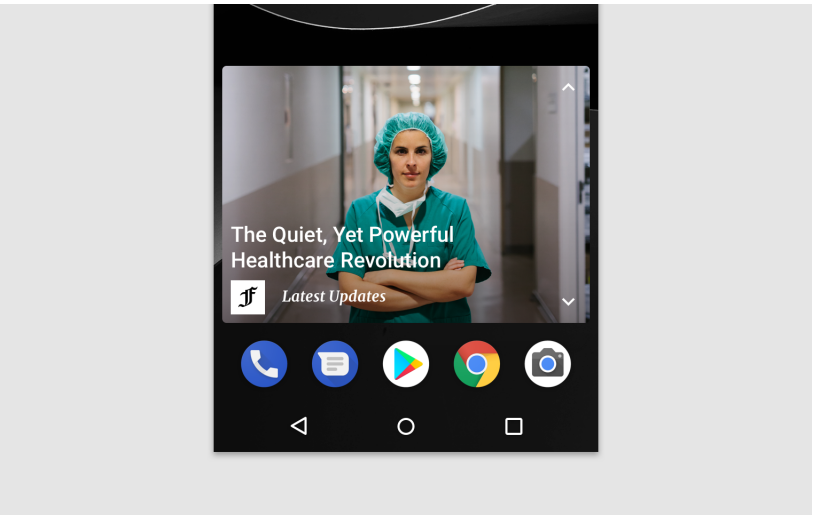


Imagen 22. CollectionWidget

- **ControlWidgets**: los widgets de control muestran funciones de uso frecuente. Estas funciones pueden activarse desde la pantalla de inicio sin abrir la aplicación. Por ejemplo, los widgets de la aplicación de música permiten al usuario reproducir, pausar u omitir pistas de música desde fuera de la aplicación de música. Ver Imagen 23.

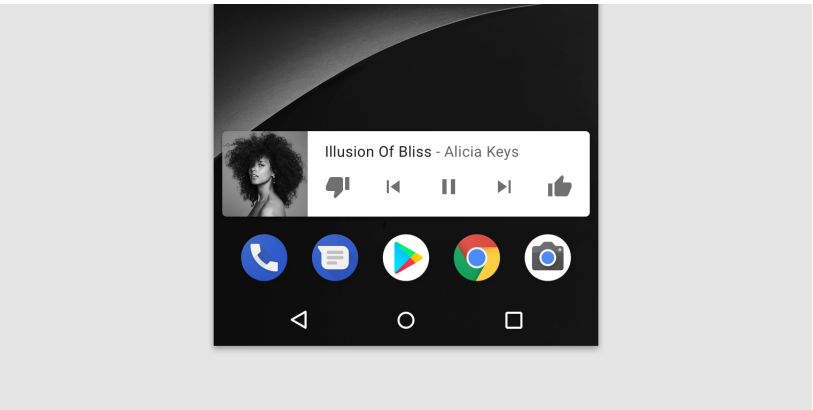


Imagen 23. ControlWidgets.

Los widgets de control pueden o no progresar a una pantalla detallada.

- **HybridWidgets**: muchos widgets son híbridos que combinan elementos de los diferentes tipos anteriores. Se recomienda centrar un widget en torno a uno híbrido y agregue elementos según sea necesario. Ver Imagen 24 de referencia y nótese la diferencia con respecto a la Imagen 55. Un ejemplo, un widget de reproductor de música híbrido combina un widget de control con elementos de un widget de información. El resultado mantiene informado al usuario sobre qué pista se está reproduciendo actualmente.

Imagen 24. HybridWidget

Ejercicio 11: Instalar plugin de Material Design a nuestro Android Studio.

Un plugin es un recurso de tipo librería que está asociado con nuestro IDE, y podemos acceder a estos desde el menú principal Tools -> SDK Manager -> Plugins.

1. Buscar el plugin “material” como lo muestra la siguiente imagen:

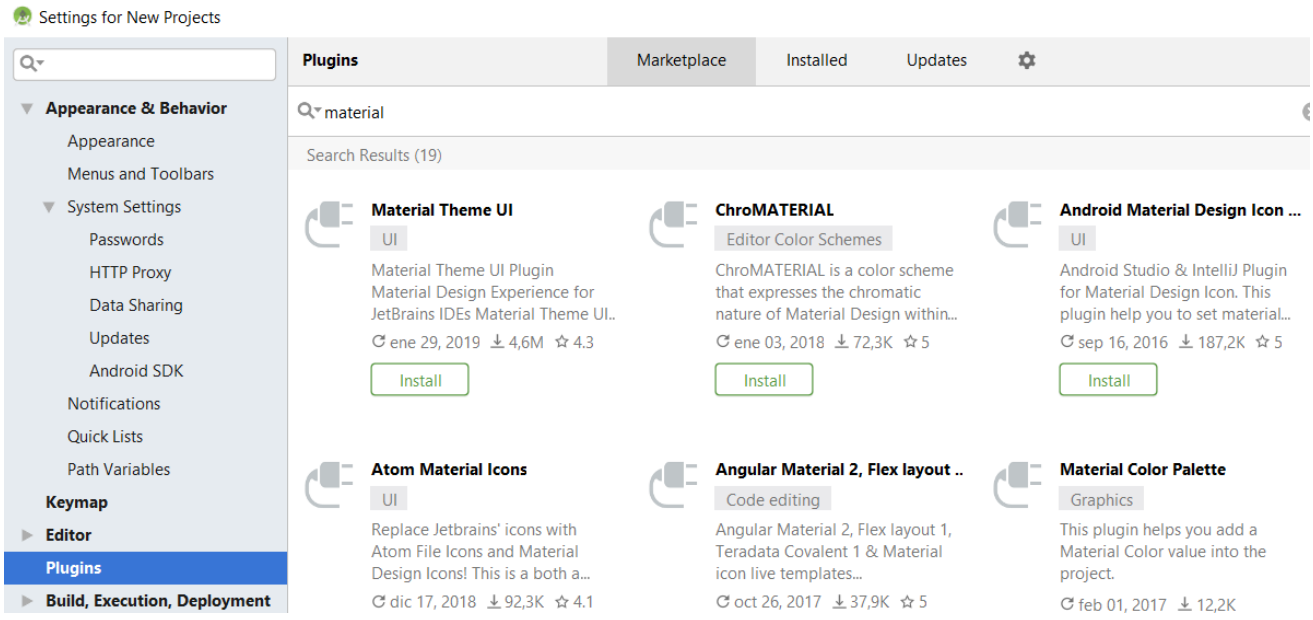


Imagen 25. Buscar material

2. Hacemos click en “Install”.

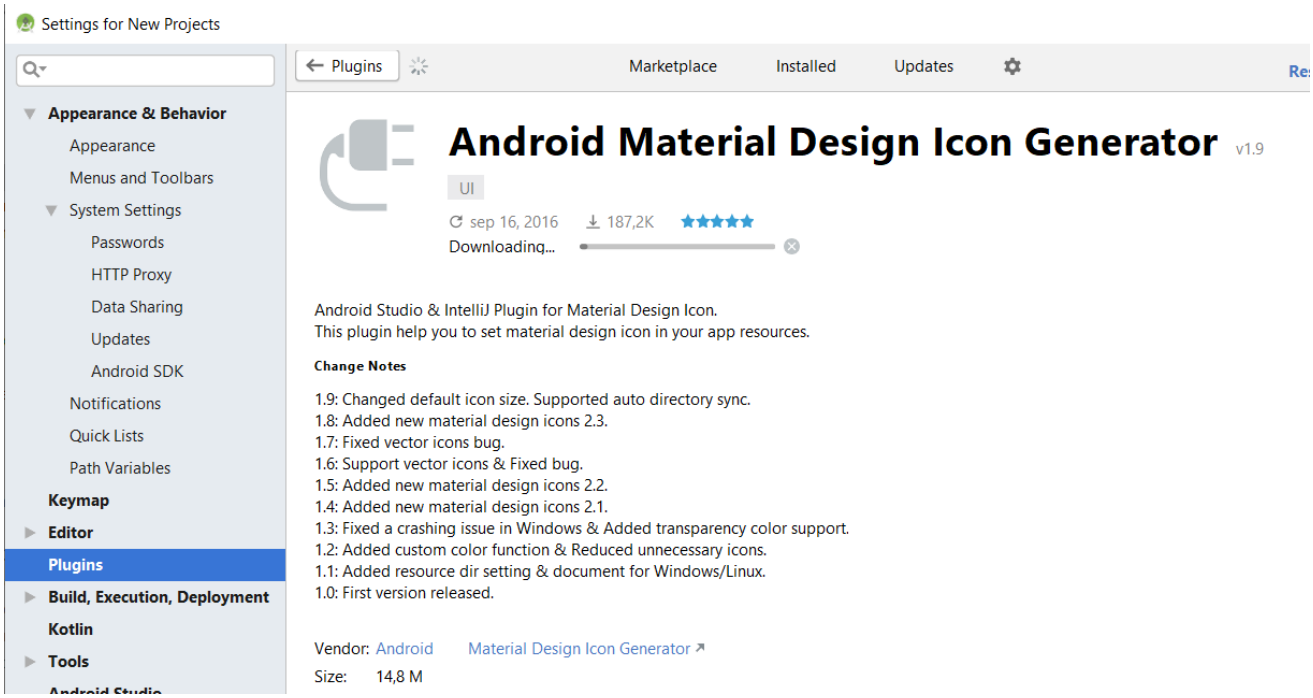


Imagen 26. Instalar

3. Hacemos click en “Aplicar” y posteriormente en “Reiniciar IDE”.
4. Al ingresar, en la vista de navegación de nuestro proyecto hacemos click derecho sobre nuestra carpeta `/res/drawable`, seleccionando a continuación `New -> Material Design Icon.

5. Seleccionamos el ícono **ic_language** (se asemeja mucho a un enlace web) y hacemos click en "aceptar". Ver imagen 27.

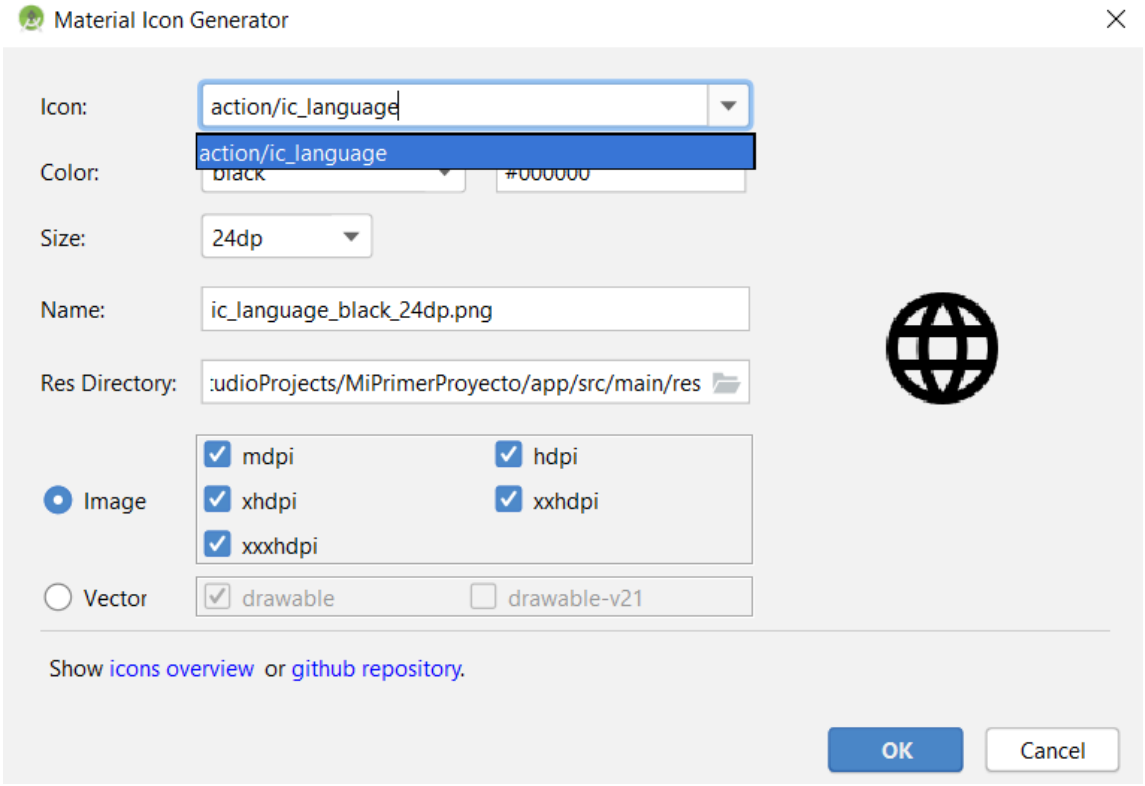


Imagen 27. Seleccionamos el ícono

6. Si hicimos todo bien, deberíamos de visualizar el nuevo recurso de icono en nuestra carpeta drawable, como lo muestra la imagen 28.

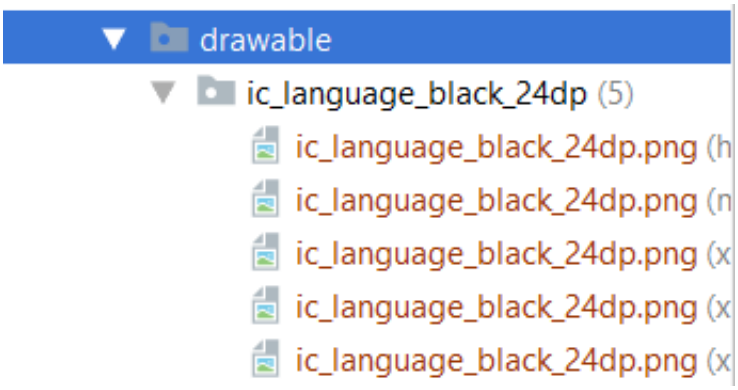


Imagen 28. Revisamos que se haya incorporado el recurso.

Ejercicio 12: Agregar un widget Floating Button de material design a nuestra HomeActivity que al hacer click, nos redirija a la página web oficial de Desafío Latam.

1. Abrimos nuestro archivo activity_home.xml desde la carpeta /res/layouts y agregamos antes del inicio de la etiqueta AppCompatImageView el siguiente código:

```
<com.google.android.material.floatingactionbutton.FloatingActionButton
    android:id="@+id/fab"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="end"
    android:src="@drawable/ic_language_black_24dp"
    android:layout_margin="16dp"
    app:backgroundTint="#FFFFFF"
    app:borderWidth="2dp"
    />
```

2. Hacemos “run” de nuestra aplicación y en la pantalla de inicio debería de aparecer nuestro nuevo FloatingButton como lo muestra la imagen 29.

Imagen 29. Aparece botón flotante.

3. Agregamos el siguiente código dentro de nuestra clase java HomeActivity.java , en el método `onCreate()` antes de su cierre (`}`):

```
FloatingActionButton myFab = findViewById(R.id.fab);
myFab.setOnClickListener(new View.OnClickListener() {
    public void onClick(View v) {
        String url = "http://www.desafiolatam.com";
        Intent i = new Intent(Intent.ACTION_VIEW);
        i.setData(Uri.parse(url));
        startActivity(i);
    }
});
```

4. Para finalizar, ejecutamos nuestra aplicación y confirmamos que los cambios funcionan correctamente, haciendo click en nuestro float button y visualizando en el navegador la web de desafío latam, como lo muestra la imágen 30.

Imagen 30. Visualizando navegador web