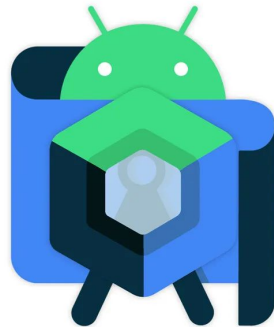


Android Superpoderes



1 | Temario: **Bloques**

- T1: Intro a Jetpack Compose
- T2: Layouts Básicos
- T3: Componentes básicos
- T4: Flows
- T5: Compose under the hood
- T6: Layouts Avanzados
- T7: Navegación
- T8: Componentes avanzados
- T9: Extras



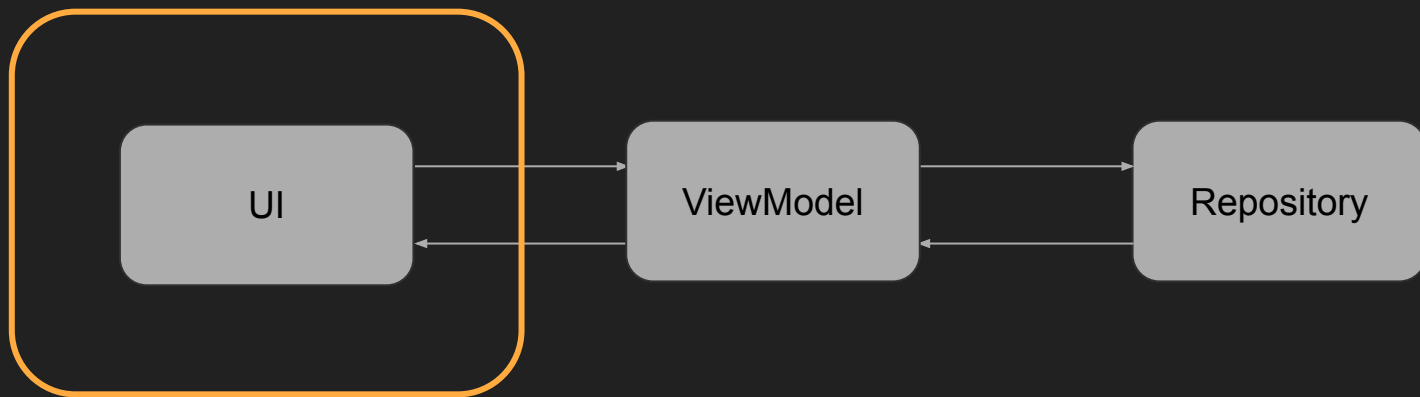
T1: Intro a Jetpack Compose



- Nuevo framework UI para Kotlin
 - Android/Desktop -> Stable
 - iOS -> Alpha
 - Web -> Experimental
 - [Github repo](#)
- Declarativo
- Compatible
- Kotlin 100%
- Conciso e idiomático
- [Documentación oficial](#)



- Auto-Refresh en las Previews.
- Compose specs para los diferentes dispositivos típicos.
- Recomposiciones visibles en Layout Inspector
- Resizable Emulator
- <https://androidgeek.co/what-is-new-in-android-studio-electric-eel-764576ffa39c>



Solo cambiamos esto

T1 | Compose: **Hello World**

- Activity padre
 - Single activity
 - No hay fragments (en principio)
- *setContent*
- Tema padre que incluye a toda nuestra app
- *@Composable*
 - Atomización
- *@Preview*
- Modifier

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            AndroidSuperpoderesTheme {
                // A surface container using the 'background'
                // color from the theme
                Surface(
                    modifier = Modifier.fillMaxSize(),
                    color = MaterialTheme.colors.background
                ) {
                    Greeting(name: "Android")
                }
            }
        }
    }
}

@Composable
fun Greeting(name: String) {
    Text(text = "Hello $name!")
}

@Preview(showBackground = true)
@Composable
fun DefaultPreview() {
    AndroidSuperpoderesTheme {
        Greeting(name: "Android")
    }
}
```

- Parte principal de Jetpack Compose
- Anotación que define una función “composable”.
 - Función que proporciona una parte del UI
- Permite atomizar el UI.
 - Design system
- Una función puede llamar a más funciones composables.
- PascalCase para los nombres de las funciones.
 - Lo normal para funciones es camelCase.
 - Si usamos linters, es necesario modificar las reglas.

```
@Composable  
fun MyApp(){
```


- Previsualización de vistas:
 - Componentes, pantallas, estilos, etc.
- Anotación para el visualizador.
 - Podemos incluir más de una.
- Rápido.
 - No necesita ejecutarse en el emulador.
 - Se puede ejecutar una vista sola en el emulador sin necesidad de la app entera.
- Diferentes configuraciones.
- Vista en los distintos dispositivos.
- Creación de capturas de pantalla.

```
@Preview
@Composable
fun MyApp(){
```

Las previews no pueden tener parámetros

- Elementos principales para modificar el comportamiento y apariencia de una vista.
- Fluent api.
 - Patrón de diseño Builder.
- Herencia de propiedades:
 - Cada componente proporciona distintas opciones.

El orden de los factores altera el producto

T2: Layouts Básicos



- Componente equivalente al `FrameLayout` en AVS.
- Organización de componentes en el eje Z.
- Modifiers típicos:
 - `Background`
 - `FillMaxSize` / `FillMaxWidth` / `FillMaxHeight`
 - `Size` / `Width` / `Height`
 - `Align`
 - `Padding`

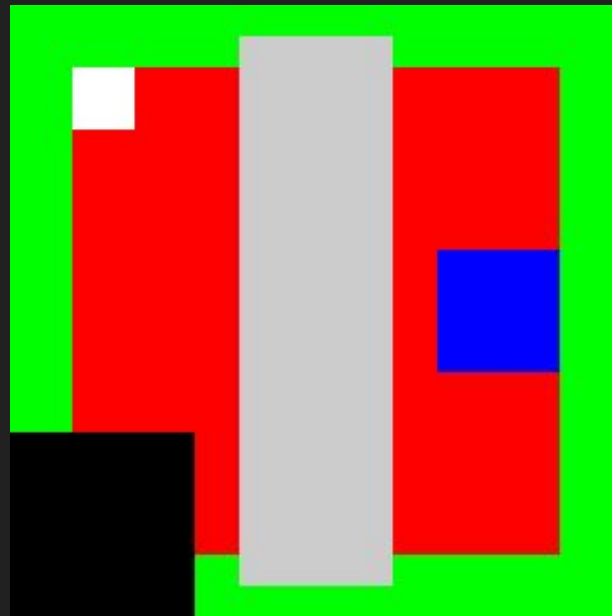
```
@Composable
fun MyBox() {
    Box() { this: BoxScope
    }
}
```

AVS = Android View System

Actividad:

Realizar la figura usando solo Box

- No hay sólo una solución válida
- Se puede anidar



- Componente equivalente a los **margins**.
- Permite separar otros componentes.
- Los padding son de una vista hacia dentro.
- Se trata como un componente más pero solo se configura el tamaño.

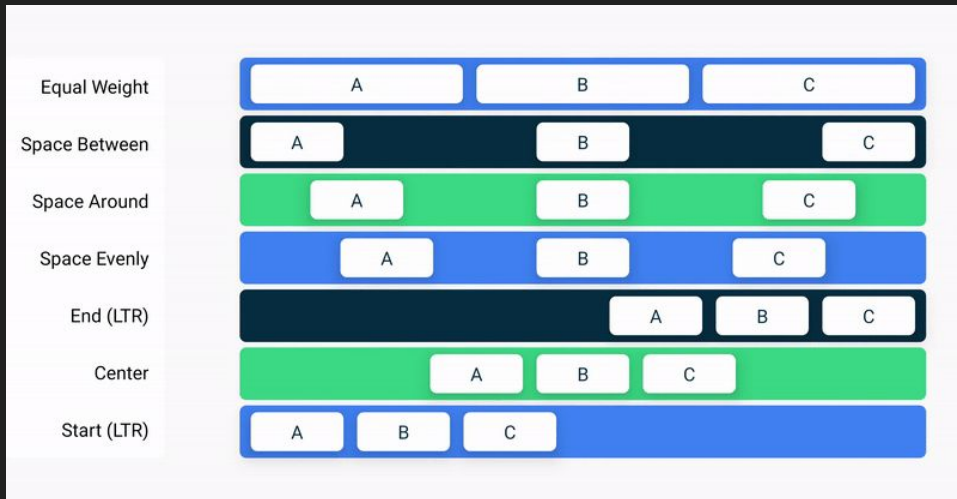
```
@Composable
fun MySpacer() {
    Spacer(modifier = Modifier.size(10.dp))
}
```

- Componente equivalente al `LinearLayout` en AVS.
- Organización de componentes en los ejes X o Y.
 - Row - Horizontal / Column - Vertical
- No todos los parámetros se cambian a través de modifiers.
- Atributos típicos:
 - `verticalAlignment` - `horizontalAlignment`
 - `horizontalArrangement` - `verticalArrangement`
- Modifiers típicos:
 - `weight`

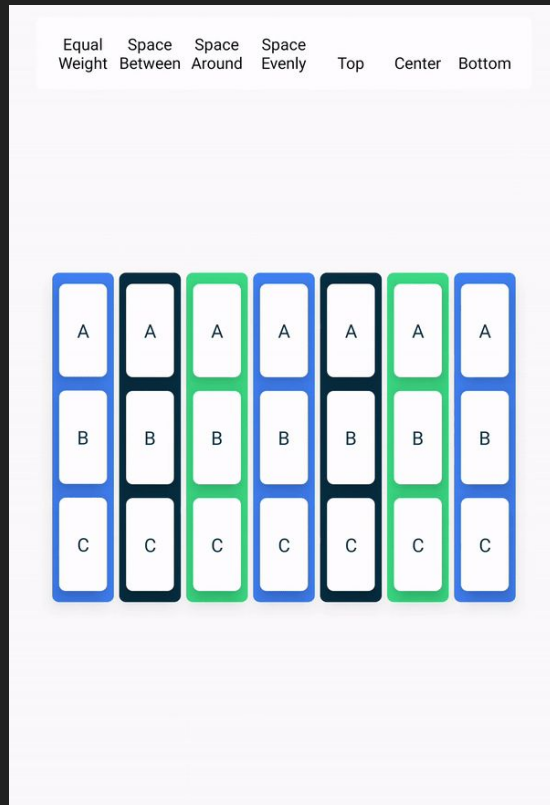
```
@Composable
fun MyRow() {
    Row() { this: RowScope
    }
}
```

```
@Composable
fun MyColumn() {
    Column() { this: ColumnScope
    }
}
```

T2 | Layouts Básicos: **Arrangements**



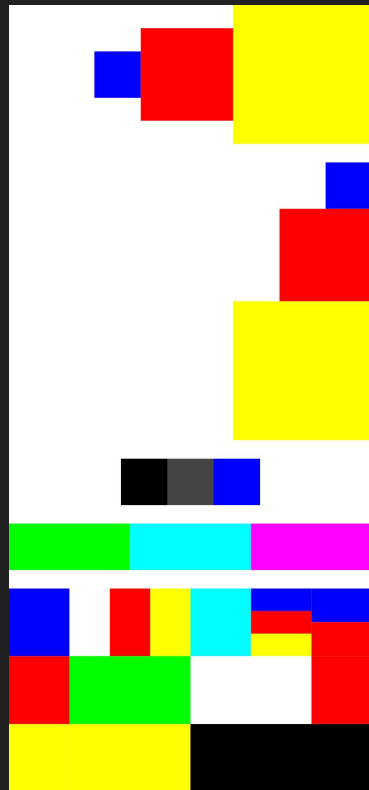
[Click para ver el gif](#)



- Realizar la figura usando solo Row, Column, Box y Spacer.
- No hay sólo una solución válida.
- Los colores dan igual mientras se respeten las secciones.
- Se DEBE anidar.

Pistas:

- Mezclar uso de height, width, weight.
- Utilizar los Alignments y los Arrangements



T3: Componentes básicos



- Componente equivalente al TextView en AVS.
- Mostrar textos estáticos.
- El único parámetro obligatorio es “text”.
- Atributos típicos:
 - fontSize
 - color
 - style
 - textDecoration
 - fontWeight
 - maxLines

```
@Composable
fun MyText(){
    Text(text: "Hello World")
}
```

- Componente equivalente al Button en AVS.
- Es un wrapper de otros componentes.
- Permite el click.
 - Aunque no es la única forma.
- Atributos típicos:
 - colors
 - enabled
 - border

```
@Composable
fun MyButton(){
    Button(onClick = {

    }) { this: RowScope
        Text(text = "Click me")
    }
}
```

- Componente equivalente al `ImageView` en AVS.
- Muestra imágenes de diferentes formatos.
 - Painters
 - Por el momento no desde URL.
- Se refuerza la accesibilidad.
 - `ContentDescription`
- Atributos típicos:
 - `alpha`
 - `contentScale`
- Modifiers típicos:
 - `clip`
 - `border`

```
@Composable
fun MyImage() {
    Image(painter = painter,
          contentDescription = "My Image")
}
```

- No hay componente equivalente en AVS.
- Es un Image pero con valores predeterminados.
- Se usa para iconos.
 - Iconos disponibles por defecto.
 - Librería de extensión.
- Atributos típicos:
 - tint

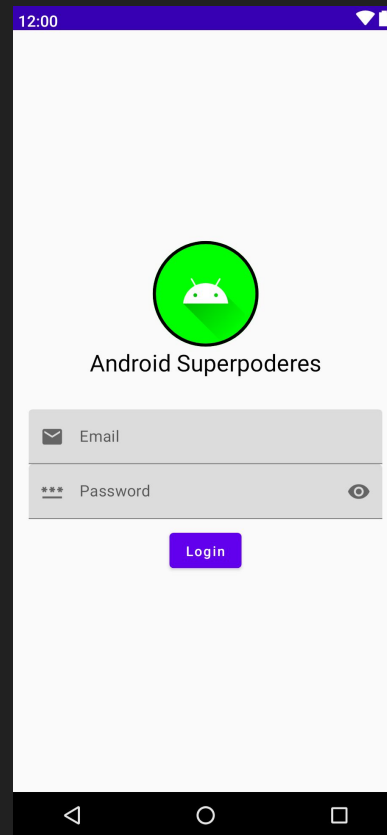
```
@Composable
fun MyIcon() {
    Icon(
        imageVector = imageVector,
        contentDescription = "My Icon"
    )
}
```

- Componente equivalente al EditText en AVS.
- Introducción de texto.
 - Valor
 - onChange
- Atributos típicos:
 - leadingIcon / trailingIcon
 - placeholder, label
 - colors
 - maxlines, singleLine
 - keyboardOptions
- Operaciones de validación.
 - Regex

```
@Composable
fun MyTextField(){
    TextField(value = value,
        onChange = { it: String
    })
}
```

No funciona bien...
Veremos por qué
en la siguiente
sección!

- Realizar un login: Estilo libre
 - Row, Column, Box y Spacer.
 - Text, Button, Image, Icon, TextInput.
- Atomizar el desarrollo en funciones.
 - Parametrizar los componentes cuando sea necesario.
- Mejorar todo lo que podáis la UX.
 - Pensad qué os gustaría que hiciera.
- No es necesario dar funcionalidad a los TextFields.



T4: Flows



- 100% Kotlin.
- Parte de la librería de corrutinas.
- Cold Streams.
 - No emiten valores hasta que no se escuchan los cambios.
- Heredan el contexto de corrutina donde se lanzan.
- [Documentación oficial](#)

- Formas de inicializar un flow.
 - Dependiendo de lo que tengamos usaremos una u otra.
- Sobre algunas colecciones:
 - `asFlow()`.
- Generar un flow inicializando con ciertos valores:
 - `flowOf()`.
- Genérico para el resto de operaciones:
 - `flow {}`.

- Instrucciones típicas que podemos implementar en un flow.
 - En general, podemos usar las operaciones de programación funcional.
- **onEach:**
 - Añade una operación al flow pero no lo transforma.
- **map:**
 - Para cada valor del flow, ejecuta la operación y devuelve el resultado.
- **filter:**
 - Para cada valor, comprueba la condición y emite el resultado si se satisface.
- **transform:**
 - Similar al map pero que nos permite emitir más elementos dentro.
- **take:**
 - Permite reducir el flow a un número máximo de elementos.

- Instrucciones con las que se finalizan las modificaciones y se recupera el resultado.
- `collect`:
 - Resultado individual de cada elemento después de las operaciones.
- `toList`:
 - Resultado del conjunto de datos emitidos en una lista después de las transformaciones.
- `toSet`:
 - Resultado del conjunto de datos emitidos en un set después de las transformaciones.

- **first:**
 - Obtiene el primer resultado del flow después de las transformaciones.
- **reduce:**
 - Recorre todos los elementos del flow después de las transformaciones y los combina según la instrucción proporcionada.
- **fold:**
 - Similar al reduce pero indicando un valor inicial.

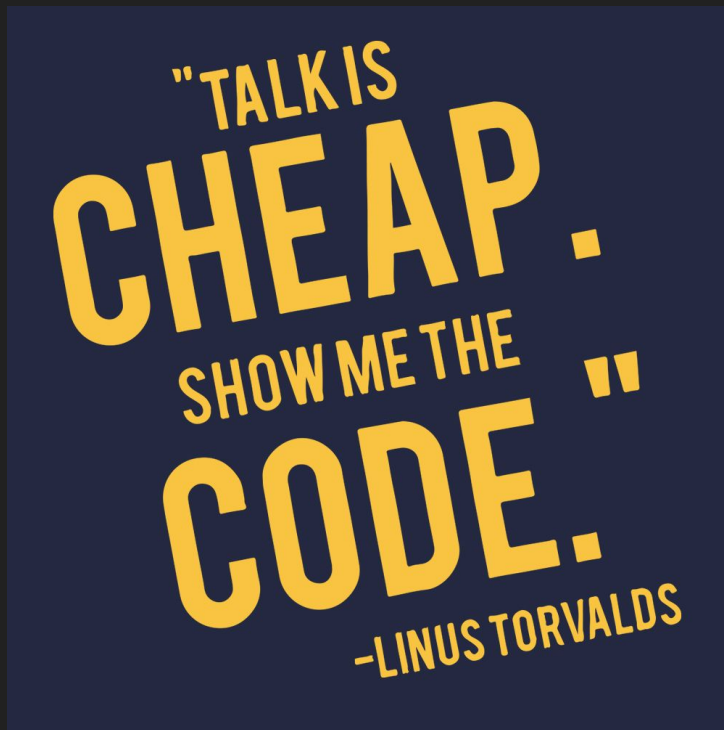
1. Flow que emita un String por cada valor inicial de un listado de superhéroes aplicando las transformaciones necesarias para conseguir lo siguiente “Hola Goku, estás en la posición 1”.
2. Flow que emita sólo el número de letras de un listado de superhéroes que contienen un número de letras impar.
3. Flow que emita los dos primeros nombres de superhéroes con un número de letras impar y devuelva un solo String del estilo “Bienvenidos, Mr. Satán y Bulma”

- Combinación de flows para juntar datos.
 - Caso típico: Remoto y local.
- Cada elemento se evalúa en la recolección.
- Operaciones:
 - Zip: relación 1-1 entre los elementos.
 - Combine: cada elemento actualiza la recolección con lo existente.

- **StateFlow:**
 - Equivalente al LiveData pero sin depender de Android.
 - No conoce el ciclo de vida de android.
- **launchOnLifecycle:**
 - Lanza una coroutine que permite dar el contexto de la actividad/fragment a los flows.
- **repeatOnLifecycle:**
 - Similar al anterior pero que se ejecuta cada vez que estamos en un cierto estado.
- **collectAsState:**
 - Mapper que nos permite convertir un StateFlow en un State de Compose.

Actividad:

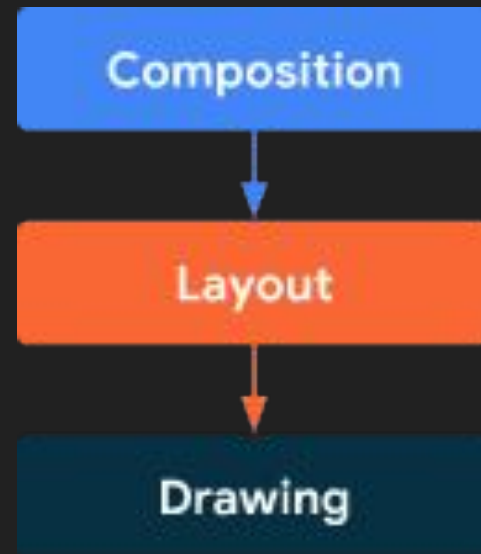
Implementar
Compose en el
proyecto



T5: Compose under the hood



- **Composition**
 - Se evalúan todos los Composables que forman parte de una vista.
 - Se comprueba si forman parte de la composición final.
- **Layout**
 - Se evalúan todas las medidas de los componentes y se genera la relación entre ellos.
- **Drawing**
 - Se evalúan las características de los componentes y se repintan las propiedades.



<https://developer.android.com/jetpack/compose/phases>

1. Creamos un botón.
2. Creamos un entero.
3. Sumamos uno al contador cuando hacemos click en el botón.
4. Pintamos el contador dentro del texto del botón.
5. Comprobar el resultado -> **No funciona correctamente.**
6. Imprimimos el valor del contador dentro del click, antes y después de modificarlo.
7. Imprimimos el valor de la variable fuera del botón.
8. Analizamos los logs que obtenemos.

T5 | Compose under the hood: **Estados**

- Hacen que una vista pueda ser recompuesta.
- Wrapper de las clases que queramos mostrar.
- `mutableStateOf()`

- Cada vez que se recompone un componente, pierde su valor a no ser que hagamos algo.
 - Igual que cuando rotamos el móvil en AVS, se vuelve a llamar al onCreate().
- remember()
- Wrapper de los estados que nos permite guardar el estado antes de la recomposición.
- 2 versiones:
 - Igualdad.
 - Delegado.

1. Envolvemos el estado con un remember.
2. Ajustamos los logs para ver el valor del estado recordado.
3. Hacemos click en el botón.
4. Comprobamos el resultado -> **Funciona!**
5. Giramos la pantalla del dispositivo.
6. Comprobar el resultado -> **Todo se ha borrado.**
7. Analizamos los logs que obtenemos.
8. ¿Se está actualizando la variable?
9. ¿Hay recomposición?

- Cada vez que se recompone un componente, por un cambio de configuración, pierde su valor aunque lo recordemos.
 - La Activity se recrea por lo que necesitamos guardarlo.
- `rememberSaveable()`
- Wrapper de los estados que nos permite guardar el estado antes de la recreación.

T5 | Compose under the hood: **Caso de uso 3**

1. Envolvemos el estado con un rememberSaveable.
2. Hacemos click en el botón.
3. Comprobamos el resultado -> Funciona!
4. Giramos la pantalla del dispositivo.
5. Comprobar el resultado -> Funciona!
6. Analizamos los logs para entender qué está pasando.



- Tipo de componentes dependiendo si tenemos referencias al estado dentro o no.
 - Este concepto está en Flutter.
- Dependier de un estado hace el componente más rívido.
- Stateful:
 - Poco configurables.
 - Opacos.
- Stateless:
 - Configurables.
 - Exposición de los atributos.

- Patrón de diseño para hacer los componentes más escalables.
- Flujos:
 - Los datos van hacia “abajo”.
 - Los eventos van hacia “arriba”.
- Componentes Stateless.
- El estado se mueve al padre más bajo que lo necesite.
 - Ese componente será Stateful.
- [Documentación oficial.](#)

T6: Layouts avanzados



- Componente equivalente al **RecyclerView** en AVS.
 - Eficiente.
- Organización de un listado de componentes en los ejes X o Y.
 - LazyRow - Horizontal / LazyColumn - Vertical.
- Componente que más código ahorra.
- Múltiples tipos de datos sin necesidad de diferentes ViewHolders.

```
@Composable
fun MyLazyRow() {
    LazyRow { this: LazyListScope

    }
}
```

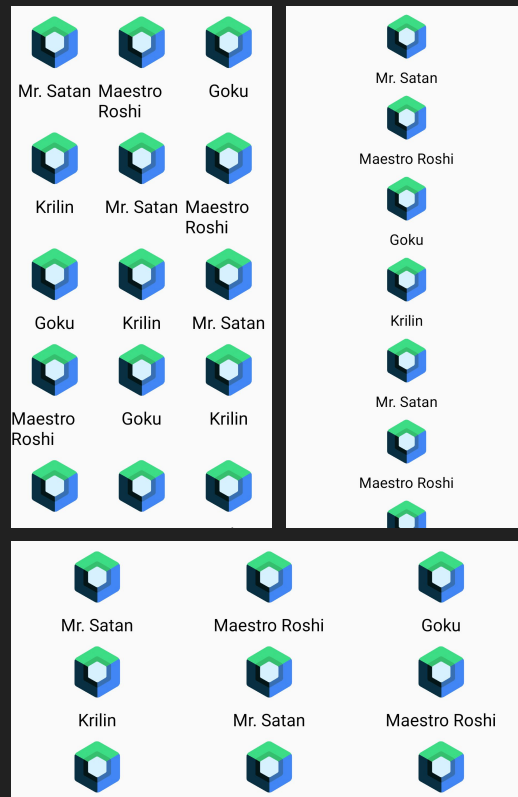
```
@Composable
fun MyLazyColumn() {
    LazyColumn { this: LazyListScope

    }
}
```

- Componente equivalente al **GridView** en AVS.
 - También al RecyclerView con un GridLayoutManager.
 - Vertical - Horizontal
- Organización de un listado de componentes en los ejes X e Y.
- GridCells:
 - Fixed.
 - Adaptive.
- Atributos típicos:
 - ContentPadding.

```
@Composable
fun MyLazyVerticalGrid() {
    LazyVerticalGrid(columns = cells) { this: LazyGridScope
    }
}
```

- Adaptar el ejercicio anterior para que si se muestra en una tablet, se use un Grid.
 - Si es vertical, 3 columnas fijas.
 - Si es horizontal, adaptativo según lo que creamos.
- Simular el “isTablet” y la orientación con parámetros.
- Mostrar previews de todas las combinaciones.



- Componente equivalente a **SurfaceView** en AVS.
- Relacionado con los conceptos de Material Design.
- Superficies con elevación y sombra.
- Manejo de colores a nivel de tema.
- Se puede conseguir el mismo efecto con modifiers.

```
@Composable
fun MySurface() {
    Surface() {

    }
}
```

- Equivalente a las **CardView** en AVS.
- Es una extensión de las Surface.
 - Tiene algunos parámetros por defecto.
- Efecto de card tridimensional.

```
@Composable
fun MyCards() {
    Card() {

    }
}
```

- Equivalente al **CoordinatorLayout** en AVS.
- Componente padre para las UI típicas.
 - Es una plantilla para los componentes característicos.
 - Gestiona por nosotros la colocación e interacción.
- Componentes típicos:
 - TopBar.
 - BottomBar.
 - SnackBar.
 - FAB.
 - Drawer.

```
@Composable
fun MyScaffold() {
    Scaffold() { it: PaddingValues

    }
}
```

T7: Navegación



- Misma filosofía que el **Navigation Component** en AVS.
- Elementos:
 - NavHost.
 - Navigation Controller.
 - Navigation Graph.
 - Composables.
- Parámetros obligatorios u opcionales.
- Backstack entries.
- Deep Links.

- Buena práctica para agrupar todas las posibilidades de navegación junto con los parámetros necesarios.
- Sealed class para limitar las opciones.
- Métodos auxiliares para crear las rutas con parámetros.

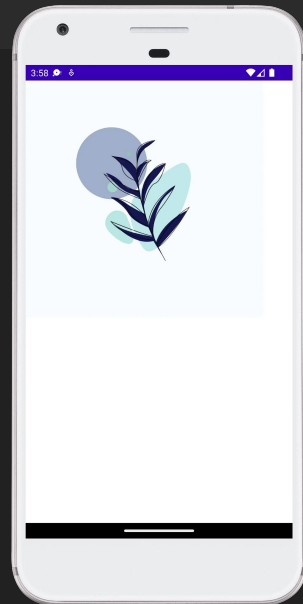
T8: Componentes avanzados



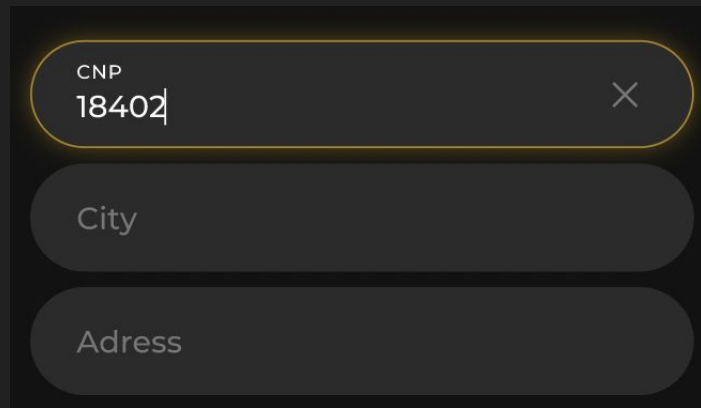
T8 | Componentes avanzados: **Async Image**

- Coil Async Image.
 - Permite cargar imágenes desde una URL.
- Librería externa.
 - Igual que en AVS.
- `rememberAsyncImagePainter()`.
- No funcionan las previews por ahora.
- <https://coil-kt.github.io/coil/compose/>

```
@Composable
fun MyAsyncImage() {
    AsyncImage(
        model = url,
        contentDescription = "My Async Image"
    )
}
```



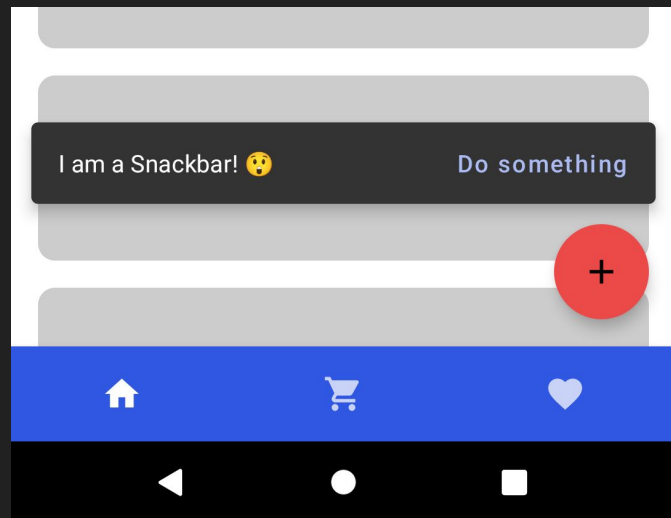
- No tiene equivalente directo en AVS.
 - Puede conseguirse modificando un `TextInputLayout`.
- Funcionamiento igual que un `TextField`.
- Adaptar según las necesidades.
- [Guía de diseño](#)



The image shows three instances of the `OutlinedTextField` component stacked vertically. The top field is active, indicated by a yellow glow, and contains the text 'CNP' and '18402' with a close button (X) on the right. The middle field is labeled 'City' and the bottom field is labeled 'Adress'.

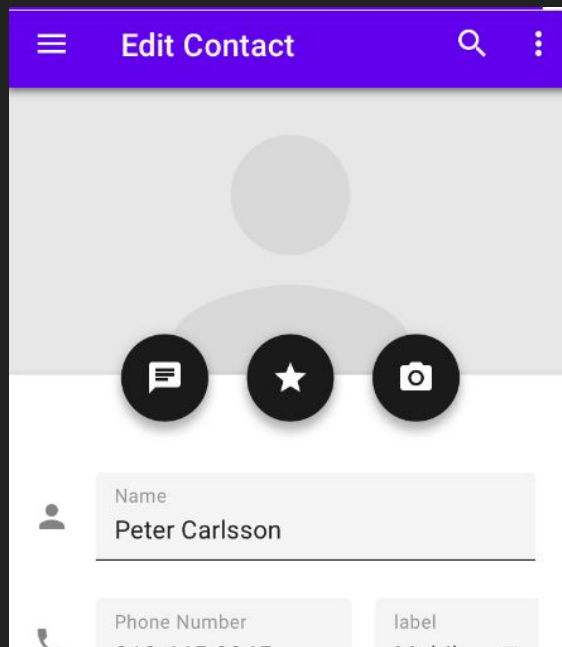
- No tiene equivalente directo en AVS.
 - Puede conseguirse modificando los valores del XML.
- Funcionamiento igual que un Button.
 - Ciertos valores por defecto.
- Expresa diferentes grados de importancia:
 - Primary -> Button.
 - Secondary -> OutlinedButton.
 - Tertiary -> TextButton.
- **Guía de diseño**

- Componente equivalente a la Snackbar en AVS.
- Muestran mensajes sencillos.
- Pueden contener acciones.
- Se recomienda dentro del Scaffold.
 - Puede incluirse en cualquier otro contenedor manejando el estado.
- <https://proandroiddev.com/advanced-work-with-the-snackbar-in-the-jetpack-compose-9bb7b7a30d60>



T8 | Componentes avanzados: **Floating Action Button**

- Componente equivalente al FloatingActionButton en AVS.
- Botón flotante para ciertas acciones.
- Diversas configuraciones dependiendo de Material2 o Material3.
- Se recomienda dentro del Scaffold.
 - Puede incluirse en cualquier otro contenedor manejando el estado.
- [Guía de diseño Material 2](#)
- [Guía de diseño Material 3](#)



T9: Extras



- Equivalente al ConstraintLayout en AVS.
- Genera un layout “aplanado”.
- Requiere una librería externa al core de Compose.
- Enlazamos a los componentes según nos convenga.
- Componentes extra:
 - Chains.
 - Guidelines.
 - Barriers.

- Compatibilidad Kotlin
 - <https://developer.android.com/jetpack/androidx/releases/compose-kotlin?hl=es-419>
- Proyecto existente
 - <https://developer.android.com/jetpack/compose/setup?hl=es-419>

1. Convertimos la variable en un estado.
2. Ajustamos los logs para ver el valor del estado.
3. Comprobar el resultado -> **No funciona correctamente.**
4. Analizamos los logs que obtenemos.
5. ¿Se está actualizando la variable?
6. ¿Hay recomposición?



KEEPCODING

Tech School

Madrid | Barcelona | Bogotá

Datos de contacto

Juanje Cilla
juanje.cilla@gmail.com