91. Componentes principales

En el programa. Usando vistas (XML)

```
1.TextView.
2.Button
3.ToggleButton
4.ImageButton
5.EditText
6.AutoCompleteTextView y MultiAutoCompleteTextView
7.Spinner
8.CheckBox
9.RadioButton
10.Switch
11.SeekBar
12.RatingBar
13.ProgressBar
```

En jetpack compose: 1. Text, 2. TextField 3. Imagenes 4. Canvas 5. Button 6. IconButton 7. TextField con DropdwonMenu (similar AutoCompleteTextView) 8. RadioButton 9. Switch 10. checkbox 11. Slider (como SeekBar) 12. No hay RatingBar (se usa Icon + Row) 13. LinearProgressIndicator y CircularProgressIndicator 14. Dialog

91.1 Text

Etiqueta de texto. (seguir en AD)

La declaración completa:

```
Text(
    text: String,
    modifier: Modifier = Modifier,
    color: Color = Color.Unspecified,
    fontSize: TextUnit = TextUnit.Unspecified,
    fontStyle: FontStyle? = null,
    fontWeight: FontWeight? = null,
    fontFamily: FontFamily? = null,
    letterSpacing: TextUnit = TextUnit.Unspecified,
    textDecoration: TextDecoration? = null,
    textAlign: TextAlign? = null,
    lineHeight: TextUnit = TextUnit.Unspecified,
    overflow: TextOverflow = TextOverflow.Clip,
    softWrap: Boolean = true,
    maxLines: Int = Int.MAX_VALUE,
    minLines: Int = 1,
    onTextLayout: ((TextLayoutResult) -> Unit)? = null,
    style: TextStyle = LocalTextStyle.current
)
```

Es recomendable usar recursos de string en lugar de codificar valores Text, ya que puedes compartir las mismas strings con Views de Android y preparar tu app para la internacionalización:

```
@Composable
fun StringResourceText() {
    Text(stringResource(R.string.hello_world))
}
```

91.2 TextField

Componente para la entrada de datos. (seguir en Android Developer AD) Se incluye en esta categoria:

- **TextField** es la implementación de Material Design. Sigue los lineamientos de Material Design
- El estilo **predeterminado** es relleno.
- **OutlinedTextField** es la versión de estilo de contorno.
- **BasicTextField** permite a los usuarios editar texto con el teclado en pantalla o físico, pero no proporciona decoraciones como sugerencias o marcadores de posición.

La declaración completa:

```
@Composable
fun TextField(
    value: String,
    onValueChange: (String) -> Unit,
    modifier: Modifier = Modifier,
    enabled: Boolean = true,
    readOnly: Boolean = false,
    textStyle: TextStyle = LocalTextStyle.current,
    label: @Composable (() -> Unit)? = null,
    placeholder: @Composable (() -> Unit)? = null,
    leadingIcon: @Composable (() -> Unit)? = null,
    trailingIcon: @Composable (() -> Unit)? = null,
    isError: Boolean = false,
    visualTransformation: VisualTransformation = VisualTransformation.None,
    keyboardOptions: KeyboardOptions = KeyboardOptions.Default,
    keyboardActions: KeyboardActions = KeyboardActions(),
    singleLine: Boolean = false,
    maxLines: Int = if (singleLine) 1 else Int.MAX_VALUE,
    minLines: Int = 1,
    interactionSource: MutableInteractionSource = remember { MutableInteractionSource()
},
    shape: Shape =
        MaterialTheme.shapes.small.copy(bottomEnd = ZeroCornerSize, bottomStart =
ZeroCornerSize),
    colors: TextFieldColors = TextFieldDefaults.textFieldColors()
    content: ()-> Unit
)
```

Para usar como password se permite el uso de VisualTransformation

```
@Composable
fun PasswordTextField() {
    var password by rememberSaveable { mutableStateOf("") }

    TextField(
        value = password,
        onValueChange = { password = it },
        label = { Text("Enter password") },
        visualTransformation = PasswordVisualTransformation(),
```

```
        keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Password)
    )
}
```

91.3 Emoji

Text y TextField tiene compatibilidad con el standard UNICODE de emoji (es una fuente más como cualquier texto) (seguir en AD)

91.4 Sustituto de AutocompleteTextView sencilla

En Jetpack Compose, el equivalente de `AutoCompleteTextView` de las vistas tradicionales de Android se puede crear utilizando una combinación de `TextField` (o OutlinedTextField) para la entrada de texto y `DropdownMenu` para mostrar las sugerencias. Esta combinación te permite construir una funcionalidad similar a la de AutoCompleteTextView.

Cómo se podría implementar:

TextField/OutlinedTextField: Para la entrada de texto. Aquí es donde el usuario escribirá su entrada.

DropdownMenu: Para mostrar una lista de opciones o sugerencias que cambian basándose en la entrada del usuario.

La idea general es que, a medida que el usuario escribe en el TextField, se actualiza una lista de sugerencias que se muestra en un DropdownMenu. Cuando el usuario selecciona una de estas sugerencias, actualizas el texto del TextField con esa selección.

```
@Composable
fun AutoCompleteTextViewExample(suggestions: List<String>) {
    var text by remember { mutableStateOf("") }
    var expanded by remember { mutableStateOf(false) }

    Column {
        TextField(
            value = text,
            onValueChange = {
                text = it
                expanded = it.isNotEmpty()
            },
            label = { Text("Escribe algo") }
        )
        DropdownMenu(
            expanded = expanded,
            onDismissRequest = { expanded = false }
        ) {
            suggestions.filter { it.startsWith(text, ignoreCase = true) }.forEach {
suggestion ->
                DropdownMenuItem(onClick = {
                    text = suggestion
                    expanded = false
                }) {
                    Text(suggestion)
                }
            }
        }
```

```
    }
}
```

91.5 Imagenes

(seguir en AD)
En este apartado tenemos:

- Carga de imágenes: Obtén más información para cargar una imagen desde un disco o de Internet.
- ImageBitmap frente a ImageVector: Obtén más información para trabajar con las imágenes de trama y vectoriales, que son los dos formatos de imagen más comunes.
- Íconos de Material: Descubre una manera conveniente de dibujar un ícono de color único en la pantalla, de acuerdo con los lineamientos de Material Design 3.
- Personaliza una imagen: Obtén más información para personalizar una imagen con las propiedades de un elemento Image componible.
- Pintor personalizado: Obtén más información sobre los objetos de pintores personalizados para editar la imagen como quieras.
- Optimización del rendimiento: Obtén más información para trabajar mejor con imágenes a fin de evitar problemas de rendimiento.

91.5.1 Carga de imágenes

Usa el elemento componible Image para mostrar un gráfico en la pantalla. Para cargar una imagen (por ejemplo: PNG, JPEG, WEBP) o un recurso vectorial del disco, se usa la API de **painterResource** con tu imagen de referencia. No necesitas conocer el tipo de elemento; solo usa painterResource en los modificadores Image o paint.

```
Image(
    painter = painterResource(id = R.drawable.dog),
    contentDescription = stringResource(id = R.string.dog_content_description)
)
```

Actualmente, painterResource admite los siguientes tipos de elementos de diseño:

- AnimatedVectorDrawable
- BitmapDrawable (PNG, JPG, WEBP)
- ColorDrawable
- VectorDrawable

91.5.1.1 Carga una imagen desde Internet

Para cargar una imagen desde Internet, hay varias bibliotecas de terceros disponibles que te ayudarán a controlar el proceso. Las bibliotecas de carga de imágenes hacen gran parte del trabajo pesado por ti: manejan la caché (para que no descargues la imagen varias veces) y la lógica de red para descargarla y mostrarla en la pantalla.

Por ejemplo, para cargar una imagen con Coil de Instacart, agrega la biblioteca a tu archivo Gradle y usa un AsyncImage para cargarla desde una URL:

```
AsyncImage(
    model = "https://example.com/image.jpg",
    contentDescription = "Translated description of what the image contains"
)
```

En gradle añade la dependencia:

```
implementation("io.coil-kt:coil:2.5.0")
```

91.6 Canvas

Área de dibujo libre (seguir en AD)

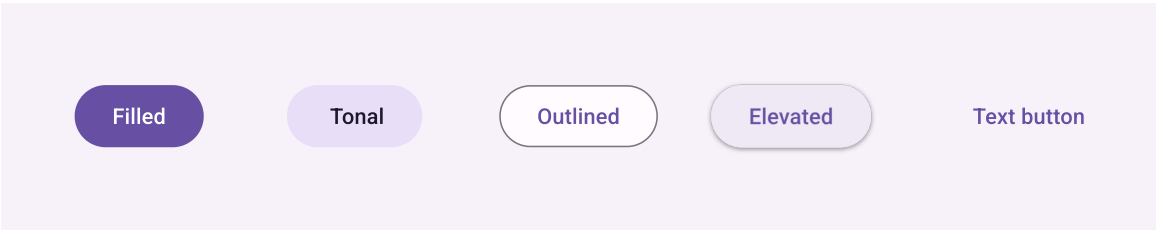
91.7 Button

Entrada (seguir en AD)

Disponemos de cinco tipos de botones:

Tipo	Apariencia	Proposito
Filled	Fondo sólido con texto contrastante	Botones de alto énfasis. Estas corresponden a las acciones principales en una aplicación, como "enviar" y "guardar". El efecto de sombra enfatiza la importancia del botón
Tonal relleno	El color de fondo varía según la superficie.	También para las acciones principales o importantes. Los botones rellenos proporcionan más funciones visuales de peso y trajes, como “agregar al carrito” y “Acceder”
Alta	Se destaca por tener una sombra.	Se ajusta a un rol similar a los botones tonales. Aumenta la elevación para que el botón se destaque aún más.
Outlined	Presenta un borde sin relleno.	Botones de énfasis medio, que contienen acciones que son importantes, pero no principales. Se vinculan bien con otros botones para indicar acciones secundarias alternativas, como "Cancelar" o "Atrás".
Texto	Muestra texto sin fondo ni borde	Botones de bajo énfasis, ideales para acciones menos críticas, como vínculos de navegación, o funciones secundarias como “Más información” o “Ver detalles”.

Ejemplo:



La declaración completa:

Button

```
@Composable
fun Button(
    onClick: () -> Unit,
    modifier: Modifier = Modifier,
    enabled: Boolean = true,
    shape: Shape = ButtonDefaults.shape,
    colors: ButtonColors = ButtonDefaults.buttonColors(),
    elevation: ButtonElevation? = ButtonDefaults.buttonElevation(),
    border: BorderStroke? = null,
    contentPadding: PaddingValues = ButtonDefaults.ContentPadding,
    interactionSource: MutableInteractionSource = remember { MutableInteractionSource()
},
    content: @Composable RowScope.() -> Unit
): Unit
```

Ejemplo

```
Button(
    onClick = { /* ... */ },
    // Uses ButtonDefaults.ContentPadding by default
    contentPadding = PaddingValues(
        start = 20.dp,
        top = 12.dp,
        end = 20.dp,
        bottom = 12.dp
    )
) {
    // Inner content including an icon and a text label
    Icon(
        Icons.Filled.Favorite,
        contentDescription = "Favorite",
        modifier = Modifier.size(ButtonDefaults.IconSize)
    )
    Spacer(Modifier.size(ButtonDefaults.IconSpacing))
    Text("Like")
}
```

91.8 IconButton

(contiuar en AD)

Imagen similar a botón. El área mínima para pulsaciones es de 48 x 48dp.

```
@Composable
IconButton(
```

```
    onClick: () -> Unit,
    modifier: Modifier,
    enabled: Boolean,
    interactionSource: MutableInteractionSource,
    content: @Composable () -> Unit
)
```

content normalmente es un Icon que se puede usar de androidx.compose.material.icons.Icons.El tamaño típico de los iconos internos es de 24 x 24 dp.

```
import androidx.compose.material.Icon
import androidx.compose.material.IconButton

IconButton(onClick = { /* doSomething() */ }) {
    Icon(Icons.Filled.Favorite, contentDescription = "Localized description")
}
```

91.9 Checkbox

Seguir en AD

```
@Composable
fun Checkbox(
    checked: Boolean,
    onCheckedChange: ((Boolean) -> Unit)?,
    modifier: Modifier = Modifier,
    enabled: Boolean = true,
    interactionSource: MutableInteractionSource = remember { MutableInteractionSource()
},
    colors: CheckboxColors = CheckboxDefaults.colors()
): Unit
```

Necesitamos un estado para controlar el checkbox y una lambda para recoger los cambios.

```
var checked by remember { mutableStateOf(false) }
Checkbox(
    checked = checked,
    onCheckedChange = { checked = it }
)
```

Personalización (seguir en AD)

```
Checkbox(
    checked = checked,
    onCheckedChange = { checked = it },
    colors = CheckboxDefaults.colors(checkedColor = Color.Green)
)
```

Uso con etiquetas Para proporcionar información sobre el checkbos

```
Row(verticalAlignment = Alignment.CenterVertically) {
    Checkbox(
        checked = checked,
        onCheckedChange = { checked = it }
    )
```

```
    Text(text = "Acepto los términos y condiciones")
}
```

91.10 Slider

Descripción y referencia

El control deslizante contiene una barra, un círculo, una etiqueta de valor y comillas simples:

- Seguimiento: Es la barra horizontal que representa el rango de valores que puede tomar el control deslizante.
- Thumb: el círculo es un elemento de control arrastrable en el control deslizante que permite al usuario seleccionar un valor específico dentro del rango definido por el segmento.
- Marcas: Las marcas son marcadores o indicadores visuales opcionales que aparecen a lo largo del recorrido del control deslizante.

Declaración:

```
@Composable
@ExperimentalMaterial3Api
fun Slider(
    state: SliderState,
    modifier: Modifier = Modifier,
    enabled: Boolean = true,
    colors: SliderColors = SliderDefaults.colors(),
    interactionSource: MutableInteractionSource = remember { MutableInteractionSource()
},
    thumb: @Composable (SliderState) -> Unit = {
        SliderDefaults.Thumb(
            interactionSource = interactionSource,
            colors = colors,
            enabled = enabled
        )
    },
    track: @Composable (SliderState) -> Unit = { sliderState ->
        SliderDefaults.Track(
            colors = colors,
            enabled = enabled,
            sliderState = sliderState
        )
    }
): Unit
```

Dispone de variantes como RangeSlider

91.11 Progress Bar

Descripción progress bar

LinearProgressIndicator

Declaración:

```
@Composable
fun LinearProgressIndicator(
    modifier: Modifier = Modifier,
    color: Color = MaterialTheme.colors.primary,
    backgroundColor: Color = color.copy(alpha = IndicatorBackgroundOpacity),
    strokeCap: StrokeCap = StrokeCap.Butt
): Unit
```

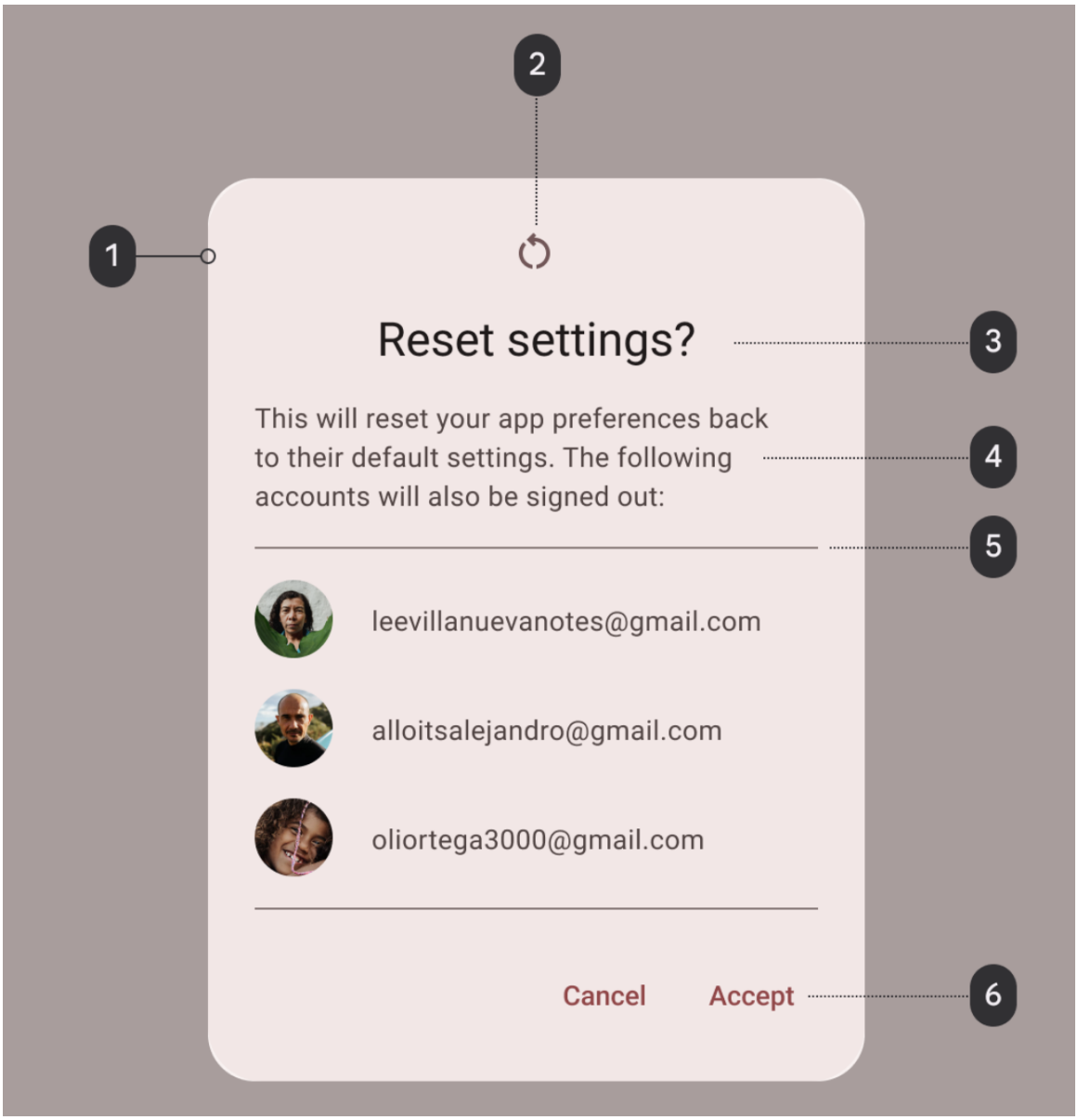
CircularProgressIndicator

Declaración

```
@Composable
fun CircularProgressIndicator(
    modifier: Modifier = Modifier,
    color: Color = MaterialTheme.colors.primary,
    strokeWidth: Dp = ProgressIndicatorDefaults.StrokeWidth,
    backgroundColor: Color = Color.Transparent,
    strokeCap: StrokeCap = StrokeCap.Square
): Unit
```

91.12 Dialog

Un Dialog es una ventana emergente sobre la pantalla actual sin llegarla a tapar.



1. Contenedor
2. Ícono (opcional)
3. Título (opcional)
4. Texto complementario
5. Divisor (opcional)
6. Acciones

91.13 SnackBar

El componente de la barra de notificaciones funciona como una breve notificación que se muestra en la parte inferior de la pantalla. Proporciona comentarios sobre una operación o acción sin interrumpir la

experiencia del usuario. Las barras de notificaciones desaparecen después de unos segundos. El usuario también puede descartarlo con una acción, como presionar un botón.

Ejemplo de snackbar con acción

```
val scope = rememberCoroutineScope()
val snackbarHostState = remember { SnackbarHostState() }
Scaffold(
    snackbarHost = {
        SnackbarHost(hostState = snackbarHostState)
    },
    floatingActionButton = {
        ExtendedFloatingActionButton(
            text = { Text("Show snackbar") },
            icon = { Icon(Icons.Filled.Image, contentDescription = "") },
            onClick = {
                scope.launch {
                    val result = snackbarHostState
                        .showSnackbar(
                            message = "Snackbar",
                            actionLabel = "Action",
                            // Defaults to SnackbarDuration.Short
                            duration = SnackbarDuration.Indefinite
                        )
                    when (result) {
                        SnackbarResult.ActionPerformed -> {
                            /* Handle snackbar action performed */
                        }
                        SnackbarResult.Dismissed -> {
                            /* Handle snackbar dismissed */
                        }
                    }
                }
            }
        )
    }
) { contentPadding ->
    // Screen content
}
```

91.14 Apendice

Enlaces:

- Resumen de paquetes de componentes

Versión 0.5, 12-12-23 Versión 0.8 10-1-24

¿Fue útil esta página?

☺ ☹