Bloque 1 · Tema 1B

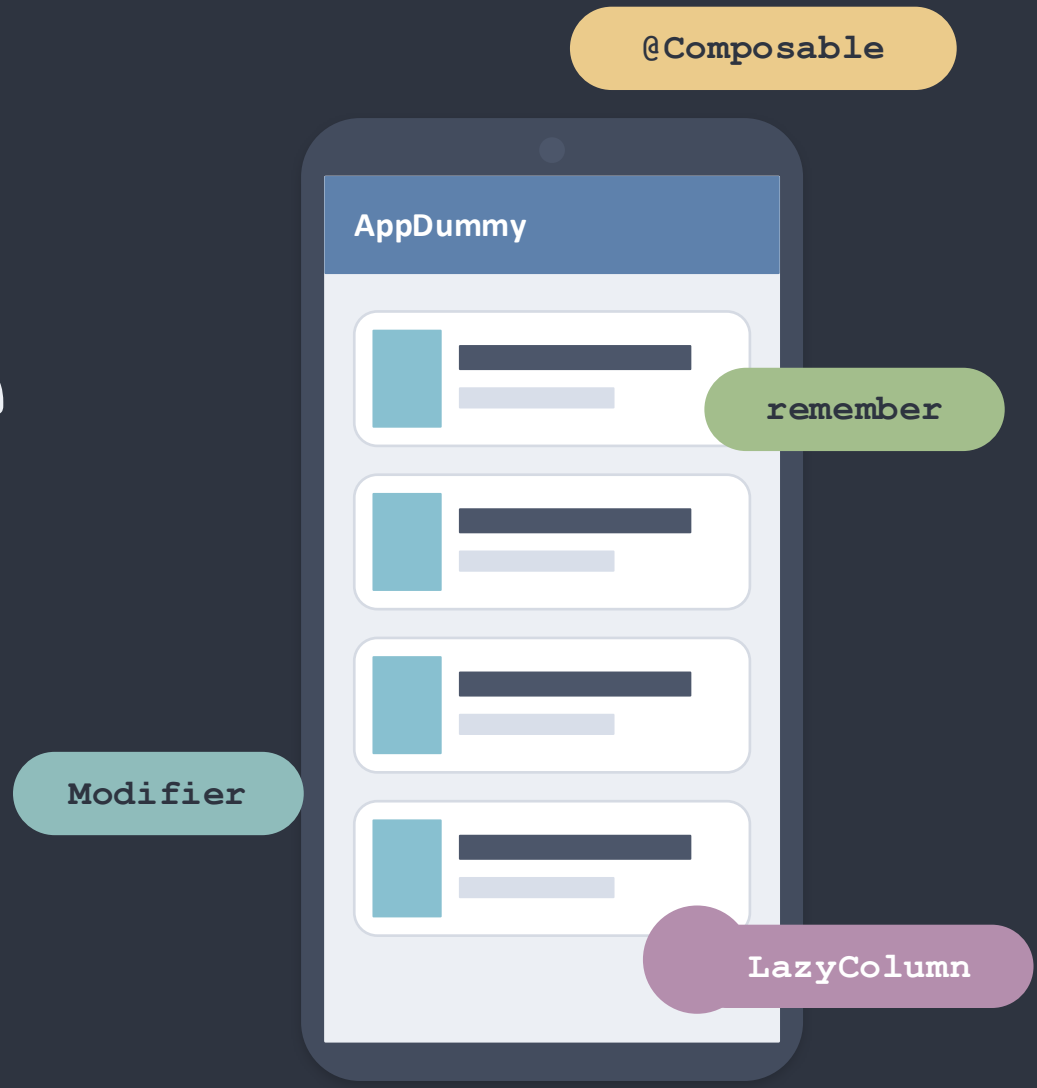
Jetpack Compose

Introducción a la interfaz de usuario declarativa en Android

3 horas

AppDummy · biblioteca personal

RA1



Programación Multimedia y Dispositivos Móviles (0489) · 2º CFGS DAM

Hoja de ruta del tema

12 apartados

1 **¿Qué es?**
Declarativo frente a imperativo

2 **Arquitectura**
UI · Runtime · Compiler

3 **Proyecto**
Estructura y dependencias

4 **@Composable**
Reglas y recomposición

5 **Layouts**
Column · Row · Box

6 **Componentes**
Text, Button, TextField, Image

7 **Modifier**
El orden importa

8 **Estado**
remember y estado guardado

9 **Listas**
LazyColumn y LazyRow

10 **Scaffold**
Estructura de pantalla

11 **Diálogos**
AlertDialog y Snackbar

12 **@Preview**
Ver la UI sin ejecutar

¿Qué vamos a evaluar?

RA1

RA1

Aplica tecnologías de desarrollo para dispositivos móviles evaluando sus características y capacidades.

RA1-b	Identificar las tecnologías de desarrollo móvil
RA1-c	Instalar, configurar y usar entornos de trabajo
RA1-f	Analizar la estructura de aplicaciones existentes
RA1-g	Realizar modificaciones sobre aplicaciones existentes
RA1-h	Usar emuladores para comprobar el funcionamiento

★ *Compose: la tecnología actual de UI en Android*

Antes de empezar: el proyecto AppDummy

AppDummy

1



Empty Activity

Proyecto nuevo con la plantilla Kotlin + Jetpack Compose.

2



Paquete propio

`edu.apellido1Nombre.appdummy`

3



Git desde el inicio

VCS ▸ Enable Version Control Integration y commits frecuentes.

Evolución de AppDummy en el Bloque 1

0 **v0** Proyecto inicial → 1 **v1 · T1B** Bienvenida → 2 **v2 · T1C** Listado → 3 **v3 · T1D** Permisos

01

Fundamentos

Qué es Compose, cómo funciona y cómo se configura



Imperativo frente a declarativo

Paradigma

CÓMO cambiar la UI Sistema clásico: XML + Kotlin

activity_main.xml

```
<TextView android:id="@+id/textContador"
    android:text="0" />
<Button android:id="@+id/btnIncrementar" />
```

MainActivity.kt

```
var contador = 0
btnIncrementar.setOnClickListener {
    contador++
    // actualizar la vista a mano
    textContador.text = contador.toString()
}
```

El desarrollador sincroniza vista y datos

QUÉ mostrar Jetpack Compose

ContadorScreen.kt

```
@Composable
fun ContadorScreen() {
    var contador by remember {
        mutableIntStateOf(0) }
    Column {
        Text(text = "$contador")
        Button(onClick = { contador++ }) {
            Text("+")
        }
    }
}
```

Compose actualiza el Text automáticamente

UI = f(Estado)

Compose



Menos código

Sin XML, findViewById ni Adapters

Reactividad

La UI se actualiza sola

Kotlin nativo

Lambdas y extensiones

Previsualizaciones

Ver la UI sin ejecutar

Interoperable

Convive con vistas XML

Arquitectura interna: tres capas

Compose

Compose UI

Text, Button, Column, Row, Image…

Donde trabaja el desarrollador

Compose Runtime

Gestiona el estado, detecta cambios

Decide qué partes recomponer

Compose Compiler

Plugin del compilador de Kotlin

Transforma las funciones @Composable

Capa de UI intercambiable

Se mantiene runtime y compilador y se cambia la capa visual:



Android



Desktop



Web



TV

Estructura de un proyecto Compose

Empty Activity

```
app/
├─ src/main/
│  ├─ java/com.ejemplo.app/
│  │  ├─ MainActivity.kt
│  ├─ res/
│  │  ├─ values/themes.xml
│  │  ├─ values/strings.xml
│  │  ├─ drawable/
│  ├─ AndroidManifest.xml
├─ build.gradle.kts
```

MainActivity.kt

```kotlin
class MainActivity : ComponentActivity() {
    override fun onCreate(b: Bundle?) {
        super.onCreate(b)
        enableEdgeToEdge()
        setContent {
            AppTheme {            // tema Material 3
                AppDummyApp()     // composable raíz
            }
        }
    }
}
```

- Punto de entrada
- Configuración de la app
- Dependencias

`setContent { }`

Puente entre Activity y Compose

Dependencias y Compose BOM

Gradle KTS

build.gradle.kts (módulo app)

```
plugins {
    id("com.android.application")
    id("org.jetbrains.kotlin.plugin.compose")
}
android {
    compileSdk = 36
    defaultConfig { minSdk = 30; targetSdk = 36 }
    buildFeatures { compose = true }
}
dependencies {
    implementation(platform(
        "androidx.compose:compose-bom:2026.06.01"))
    implementation("androidx.compose.ui:ui")
    implementation("androidx.compose.ui:ui-tooling-preview")
    implementation("androidx.compose.material3:material3")
    implementation("androidx.activity:activity-compose:1.13.0")
    debugImplementation("androidx.compose.ui:ui-tooling")
}
```

compose-bom

Bill of Materials: fija versiones compatibles



sin versión individual

Kotlin 2.x

El compilador de Compose es un plugin de Kotlin. composeOptions y kotlinCompilerExtensionVersion ya no se usan.

02

Composables y layouts

@Composable, recomposición, Column, Row, Box y componentes



Funciones @Composable

`@Composable`

Saludo.kt

```kotlin
@Composable
fun Saludo(nombre: String) {
    Text(text = "Hola, $nombre")
}

@Preview(showBackground = true)
@Composable
fun SaludoPreview() { Saludo("Ana") }
```

`Saludo` **Nombre en PascalCase, como una clase**

Una función composable describe un fragmento de UI. No devuelve nada: emite elementos.

1

Solo desde otro @Composable

o desde un ámbito de composición como setContent.

2

Sin efectos secundarios

Nada de escribir en BD ni llamadas de red directas.

3

Idempotentes

Mismos parámetros → mismo resultado.

4

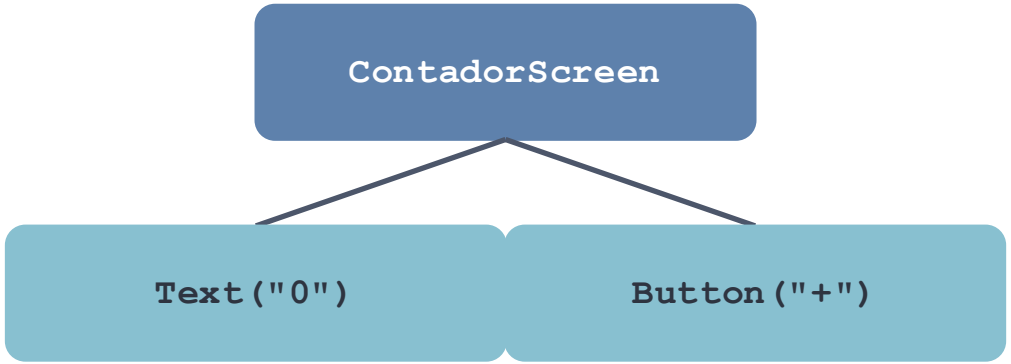
En cualquier orden

El runtime puede ejecutarlas en paralelo.

Composición y recomposición

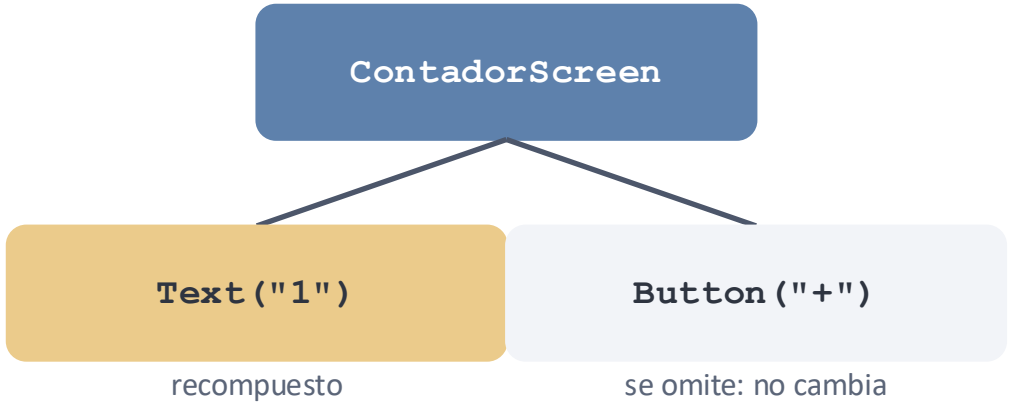
Recomposición

Estado inicial · contador = 0



click

Tras pulsar + · contador = 1



Compose es selectivo

Cuando cambia un estado, vuelve a ejecutar solo los Composables que lo leen. Si los parámetros de un Composable no cambian, se salta su recomposición. Por eso la UI sigue siendo eficiente con muchos elementos.

Layouts básicos: Column, Row y Box

Layouts

Column

Apila en vertical

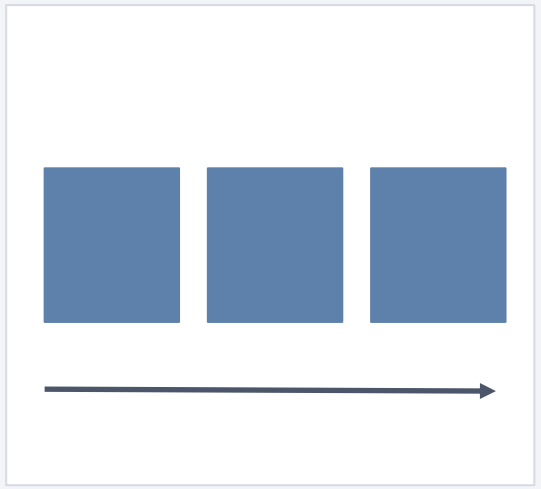


```
horizontalAlignment
verticalArrangement

Arrangement.spacedBy(8.dp)
```

Row

Apila en horizontal

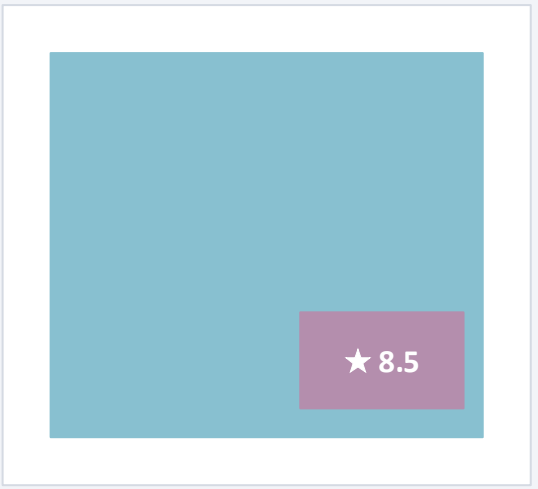


```
verticalAlignment
horizontalArrangement

Arrangement.SpaceBetween
```

Box

Superpone en el eje Z

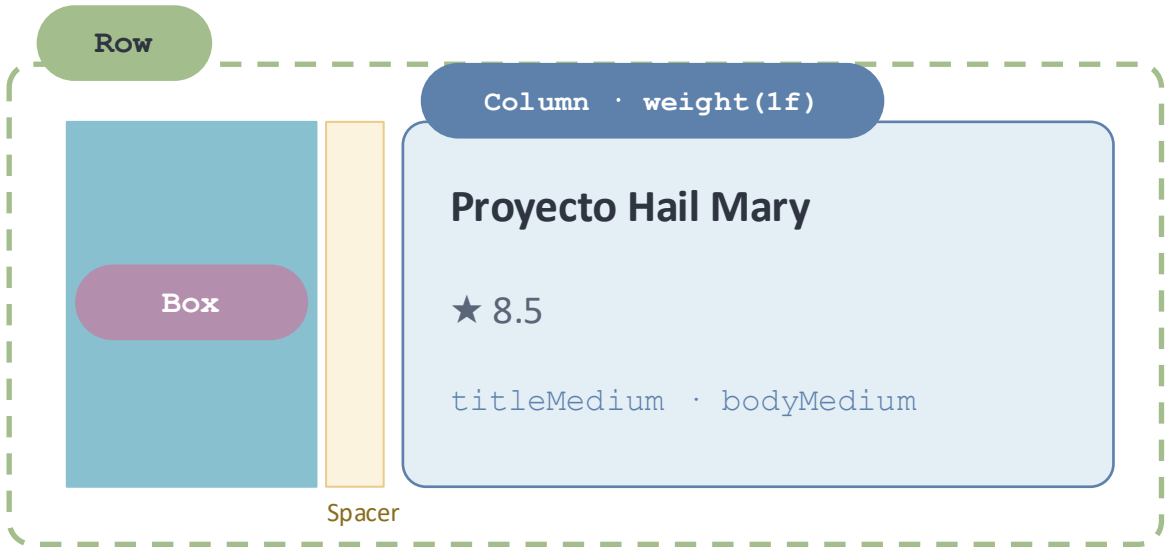



```
contentAlignment
Modifier.align()

Alignment.BottomEnd
```

Anidar layouts: la tarjeta de un elemento

Layouts



Lectura del árbol

- **Row** contenedor horizontal
- **Box** hueco fijo para la portada
- **Spacer** separación de 12 dp
- **Column** ocupa el resto con weight(1f)

TarjetaPelicula.kt

```
@Composable
fun TarjetaPelicula(titulo: String,
                    puntuacion: Double) {
    Row(Modifier.fillMaxWidth().padding(8.dp)) {
        Box(Modifier.size(80.dp, 120.dp)) {
            // AsyncImage de Coil (T1C)
        }
        Spacer(Modifier.width(12.dp))
        Column(Modifier.weight(1f)) {
            Text(titulo, style =

MaterialTheme.typography.titleMedium)
            Text("★ $puntuacion")
        }
    }
}
```

Catálogo de componentes básicos

Material 3

Text

Título principal

Subtítulo

Texto largo que no cabe y termina en…

```
style = MaterialTheme.typography.*
maxLines + TextOverflow.Ellipsis
```

Button y variantes

Reproducir — acción principal

`Button`

Añadir a favoritos — acción secundaria

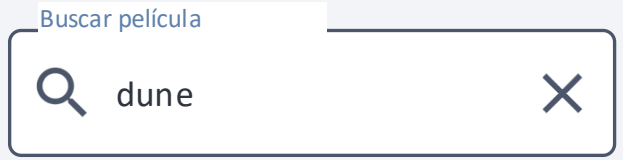
`OutlinedButton`

Ver detalles — acción terciaria

`TextButton`

`FloatingActionButton`

OutlinedTextField




```
value + onValueChange
```

Image



```
clip(CircleShape)
ContentScale.Crop
contentDescription
```

Modifier: el orden importa

Modifier

```
Text("Texto", Modifier
    .background(Color.Yellow)
    .padding(16.dp))
```



El fondo incluye el padding

```
Text("Texto", Modifier
    .padding(16.dp)
    .background(Color.Yellow))
```



El padding queda fuera del fondo

Modificadores más usados

Modifier



Tamaño

```
fillMaxSize()
fillMaxWidth()
size(100.dp)
height(50.dp)
wrapContentSize()
```



Espaciado

```
padding(16.dp)
padding(horizontal =
  16.dp)
padding(start =
8.dp)
```



Apariencia

```
background(color)
clip(CircleShape)
border(1.dp, Gray)
alpha(0.5f)
```



Interacción

```
clickable { }
clickable(enabled)
```



Posición

```
offset(x, y)
align(…)
weight(1f)
```

03

Estado

remember, mutableStateOf, rememberSaveable y estado derivado



¿Por qué remember?

Estado

Variable normal

```
@Composable
fun DemoSinRemember() {
    var x = 0  // Compose no la observa
    Button(onClick = { x++ }) { Text("x = $x") }
}
```

click 1	click 2	click 3
x = 0	x = 0	x = 0

x++ cambia la variable, pero no dispara recomposición. Y si se recompone por otro motivo, x vuelve a 0.

remember + mutableIntStateOf

```
@Composable
fun DemoConRemember() {
    var x by remember { mutableIntStateOf(0) }
    Button(onClick = { x++ }) { Text("x = $x") }
}
```

click 1	click 2	click 3
x = 1	x = 2	x = 3

mutableIntStateOf crea un valor observable; remember lo conserva entre recomposiciones. by delega get/set.

¿Hasta dónde sobrevive el estado?

`rememberSaveable`

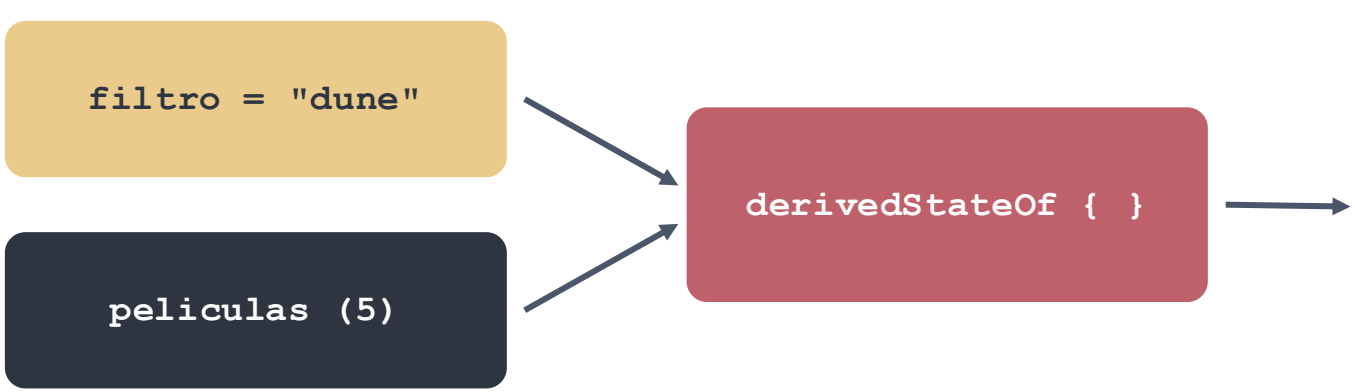
Mecanismo	Recomposición	Rotación de pantalla	Muerte del proceso
`var x = 0`	✗ Se reinicia	✗ Se pierde	✗ Se pierde
`remember { }`	✓ Se conserva	✗ Se pierde	✗ Se pierde
`rememberSaveable { }`	✓ Se conserva	✓ Se conserva	✓ Se conserva
`ViewModel (Bloque 2)`	✓ Se conserva	✓ Se conserva	Con SavedStateHandle

```
var texto by rememberSaveable { mutableStateOf("") }
OutlinedTextField(value = texto, onValueChange = { texto = it })
```

rememberSaveable guarda en un Bundle: tipos simples o Parcelable.

Estado derivado con derivedStateOf

derivedStateOf



ListaConFiltro.kt

```kotlin
var filtro by remember { mutableStateOf("") }
val peliculasFiltradas by remember(filtro) {
    derivedStateOf {
        if (filtro.isBlank()) peliculas
        else peliculas.filter { it.contains(filtro, ignoreCase = true) }
    }
}
```

04

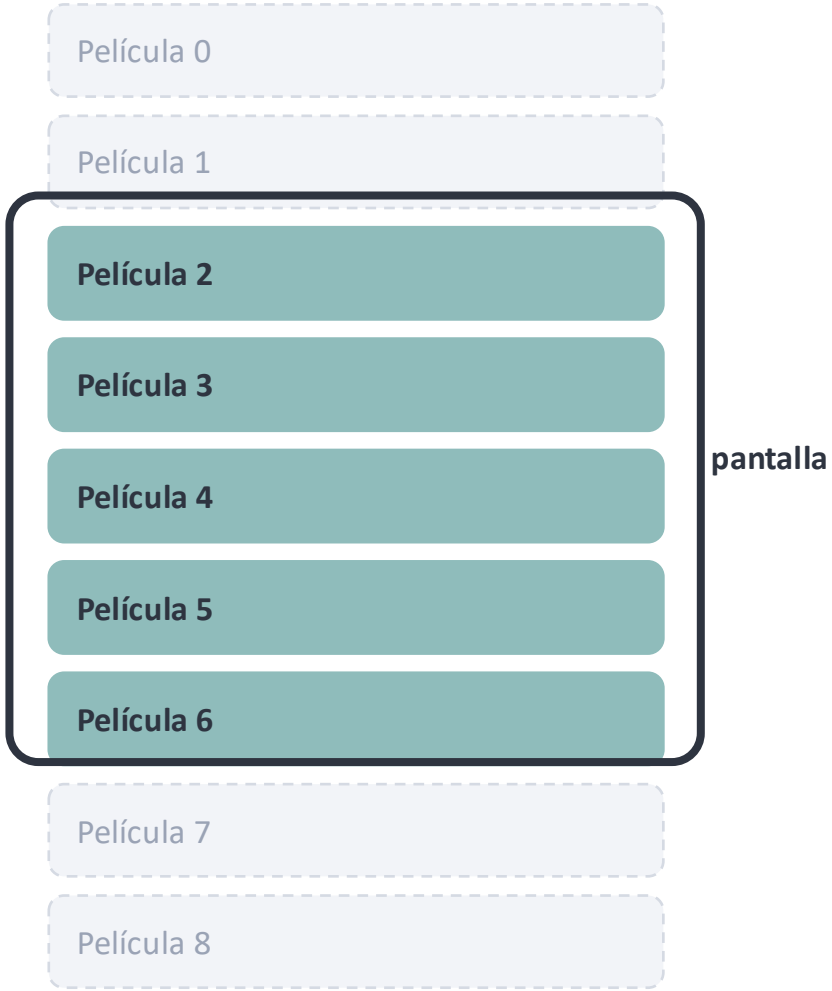
Listas y pantallas

LazyColumn, Scaffold, diálogos y previsualizaciones



LazyColumn: solo se compone lo visible

`LazyColumn`



fuera de pantalla: no se componen

```
ListaPeliculas.kt

LazyColumn(
    contentPadding = PaddingValues(16.dp),
    verticalArrangement = Arrangement.spacedBy(8.dp)
) {
    items(items = peliculas,
          key = { it }) { pelicula ->
        TarjetaPelicula(pelicula)
    }
    item { HorizontalDivider() }   // elemento suelto
}
```

`items()` Un elemento por dato

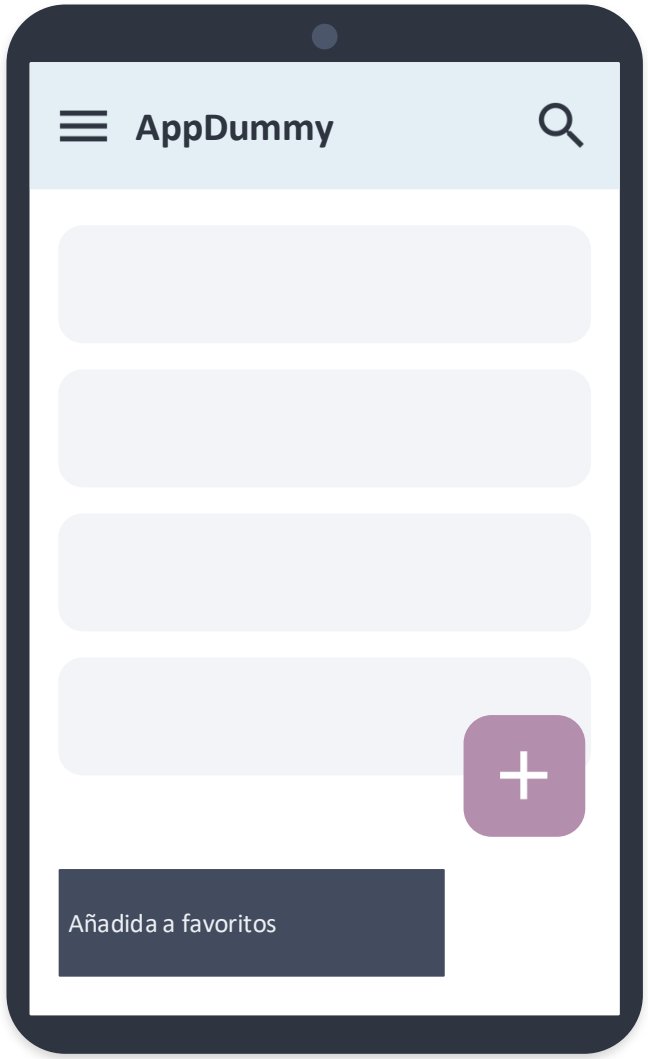
`item { }` Cabecera o separador

`key` Identidad estable

`LazyRow` Versión horizontal

Scaffold: el esqueleto de una pantalla

Scaffold



topBar

TopAppBar con navigationIcon y actions

contenido

lambda que recibe paddingValues

floatingActionButton

Acción principal de la pantalla

snackbarHost

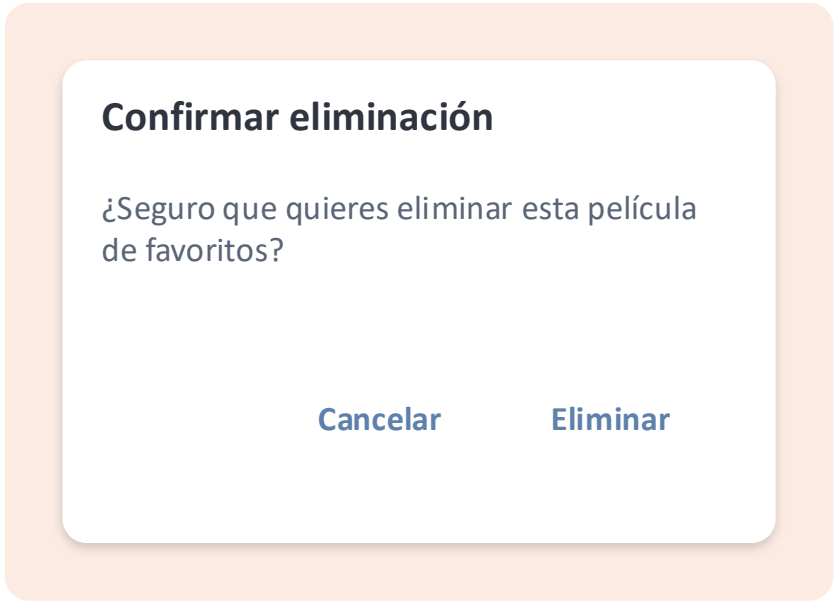
Mensajes temporales

```
Scaffold(
  topBar = { TopAppBar(
    title = { Text("AppDummy") }) },
  floatingActionButton = {
    FloatingActionButton(
      onClick = { }) { … } },
  snackbarHost = {
    SnackbarHost(state) }
) { paddingValues ->
  LazyColumn(contentPadding =
    paddingValues) { … }
}
```

Aplica siempre paddingValues
o el contenido quedará bajo la barra superior.

AlertDialog y Snackbar

Diálogos



mostrarDialogo = true

↓

if (mostrarDialogo) AlertDialog(…)

↓

confirm / dismiss → false

AlertDialog — controlado por estado

```
if (mostrarDialogo) {
    AlertDialog(
        onDismissRequest = { mostrarDialogo = false },
        title = { Text("Confirmar eliminación") },
        confirmButton = { TextButton(onClick = {
            mostrarDialogo = false }) { Text("Eliminar") } },
        dismissButton = { … })
}
```

Snackbar — función suspend

```
scope.launch {   // rememberCoroutineScope()
    val r = snackbarHostState.showSnackbar(
        message = "Película añadida a favoritos",
        actionLabel = "Deshacer")
    if (r == SnackbarResult.ActionPerformed) { /* deshacer */ }
}
```

@Preview: ver la UI sin ejecutar

`@Preview`

Básica




```
showBackground = true
```

Modo oscuro




```
uiMode = UI_MODE_NIGHT_YES
```

Dispositivo

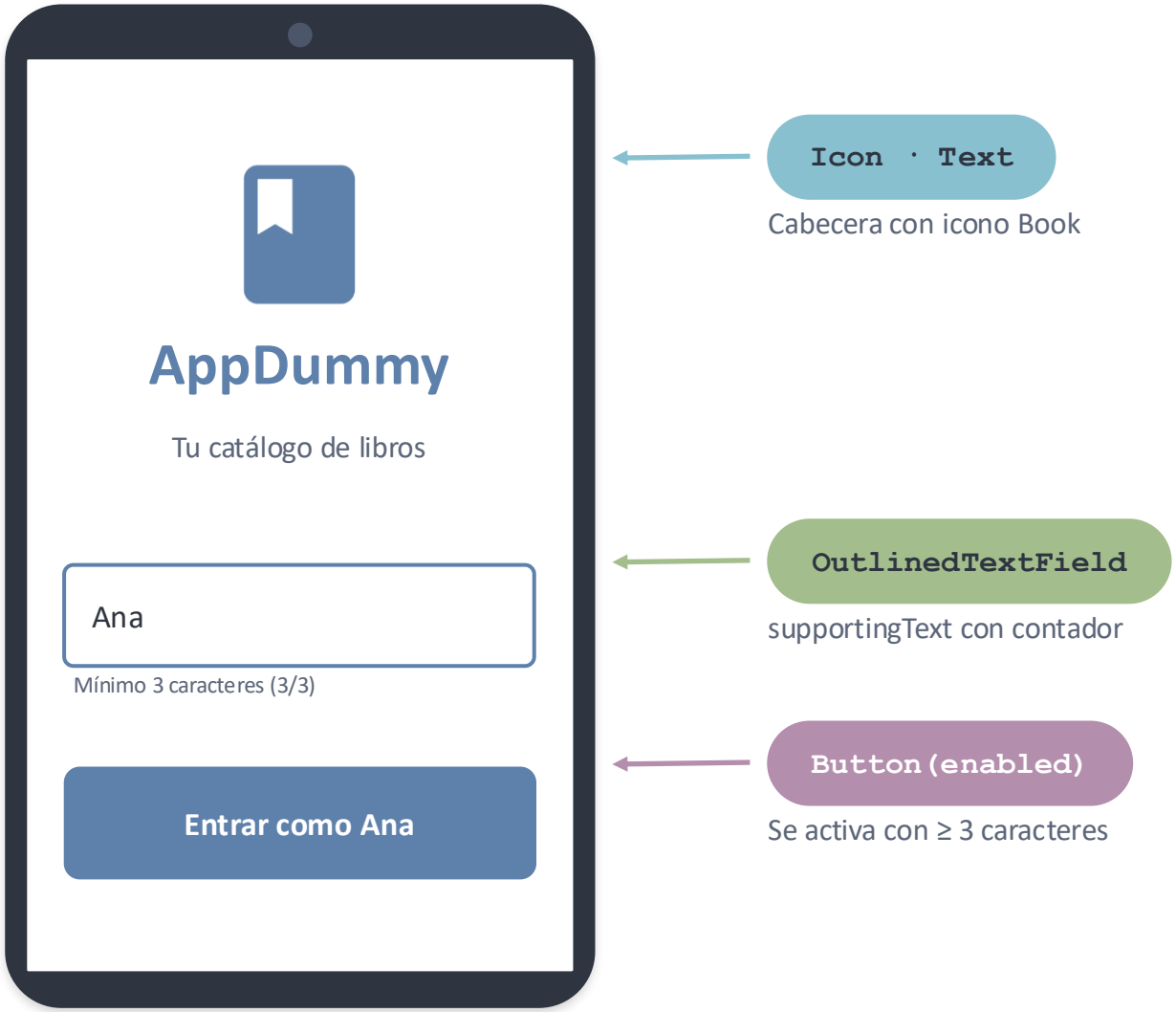



```
device = "spec:width=411dp,
          height=891dp"
```

Varias @Preview en el mismo archivo · envuelve el contenido en el tema de la app · no sustituyen al emulador

Desarrollo guiado: pantalla de bienvenida

AppDummy v1



PantallaBienvenida.kt

```
@Composable
fun PantallaBienvenida(
    onEntrar: () -> Unit) {
  var nombreUsuario by remember {
      mutableStateOf("") }
  val botonHabilitado =
      nombreUsuario.trim()
          .length >= 3
  Column(…) { … }
}
```

Dependencia

```
androidx.compose.material:
material-icons-extended:1.7.8
```

Para Icons.Default.Book

Ideas clave

UI = f(Estado)

Se describe qué mostrar; Compose lo actualiza.

@Composable

Sin efectos secundarios e idempotentes.

Column · Row · Box

Se anidan para cualquier distribución.

Modifier en orden

Cada modificador envuelve al siguiente.

remember + State

Observable y conservado entre recomposiciones.

Lazy y Scaffold

Listas eficientes y estructura Material.

Referencias

Fuentes

Documentación oficial de Jetpack Compose
developer.android.com/develop/ui/compose/documentation

Compose Tutorial — Android Developers
developer.android.com/jetpack/compose/tutorial

Estado en Jetpack Compose
developer.android.com/develop/ui/compose/state

Layouts en Compose
developer.android.com/develop/ui/compose/layouts/basics

Material 3 en Compose
developer.android.com/develop/ui/compose/designsystems/material3

Compose compiler (plugin de Kotlin)
developer.android.com/develop/ui/compose/compiler

Tema 1B. Introducción a Jetpack Compose (apuntes)
documentation.javiercarrasco.es/pmdm/B1/B1-T1b_JetpackCompose