

# TECNOLÓGICO NACIONAL DE MÉXICO

## INSTITUTO TECNOLÓGICO DEL VALLE DE OAXACA



TECNOLÓGICO  
NACIONAL DE MEXICO

ASIGNATURA:  
APLICACIONES MOBILES

NOMBRE DEL ALUMNO:  
VÍCTOR ROMÁN PÉREZ LÓPEZ

NO. CONTROL: 22920259  
GRUPO: A. SEMESTRE: 40



CARRERA: INGENIERÍA EN TECNOLOGÍAS DE LA INFORMACIÓN Y COMUNICACIONES.

NOMBRE DEL DOCENTE: CARDOSO JIMENEZ AMBROSIO

LUGAR Y FECHA: NAZARENO, SANTA CRUZ  
XOXOCOTLÁN, OAXACA 01 DE MAYO DEL 2024

Curso de jetpack compose android

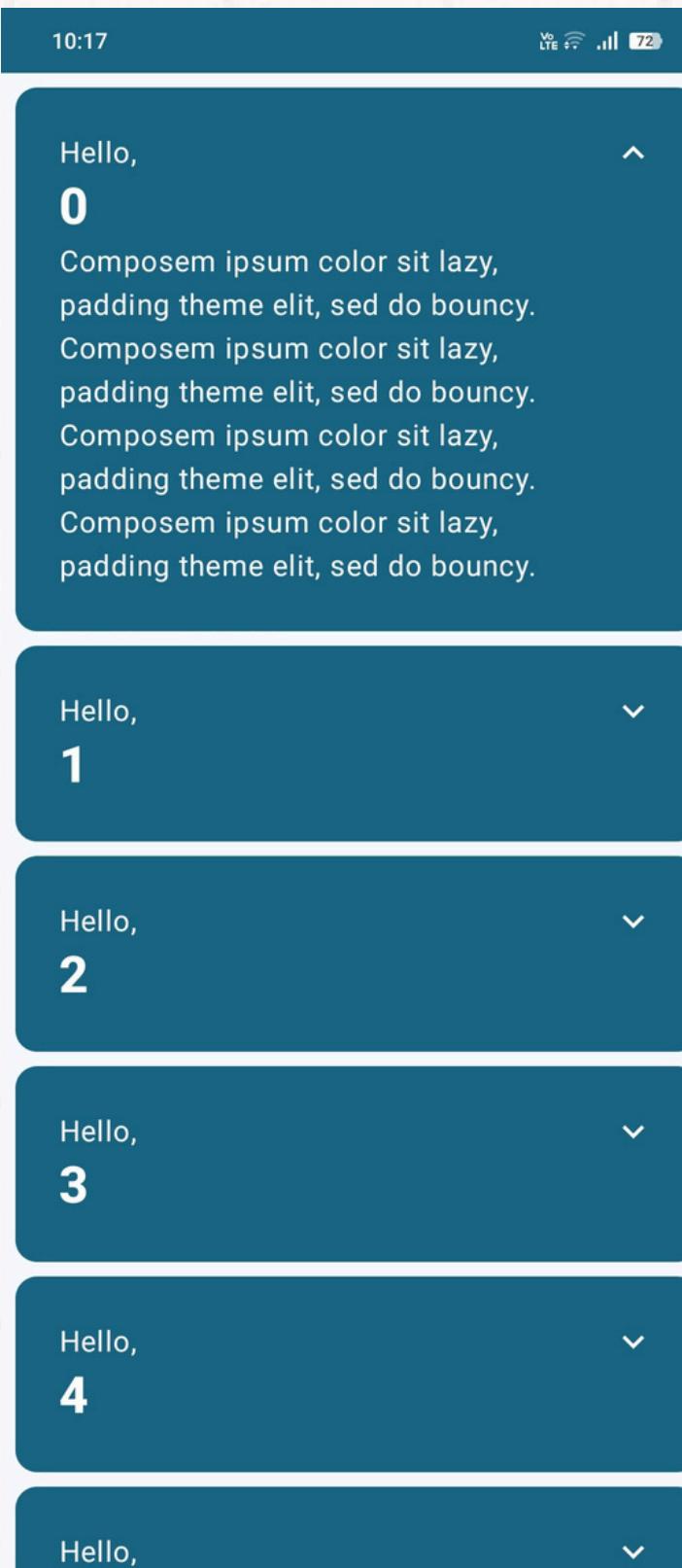
# ASPECTOS BÁSICOS DE JETPACK COMPOSE

10:17



Welcome to the Basics Codelab!

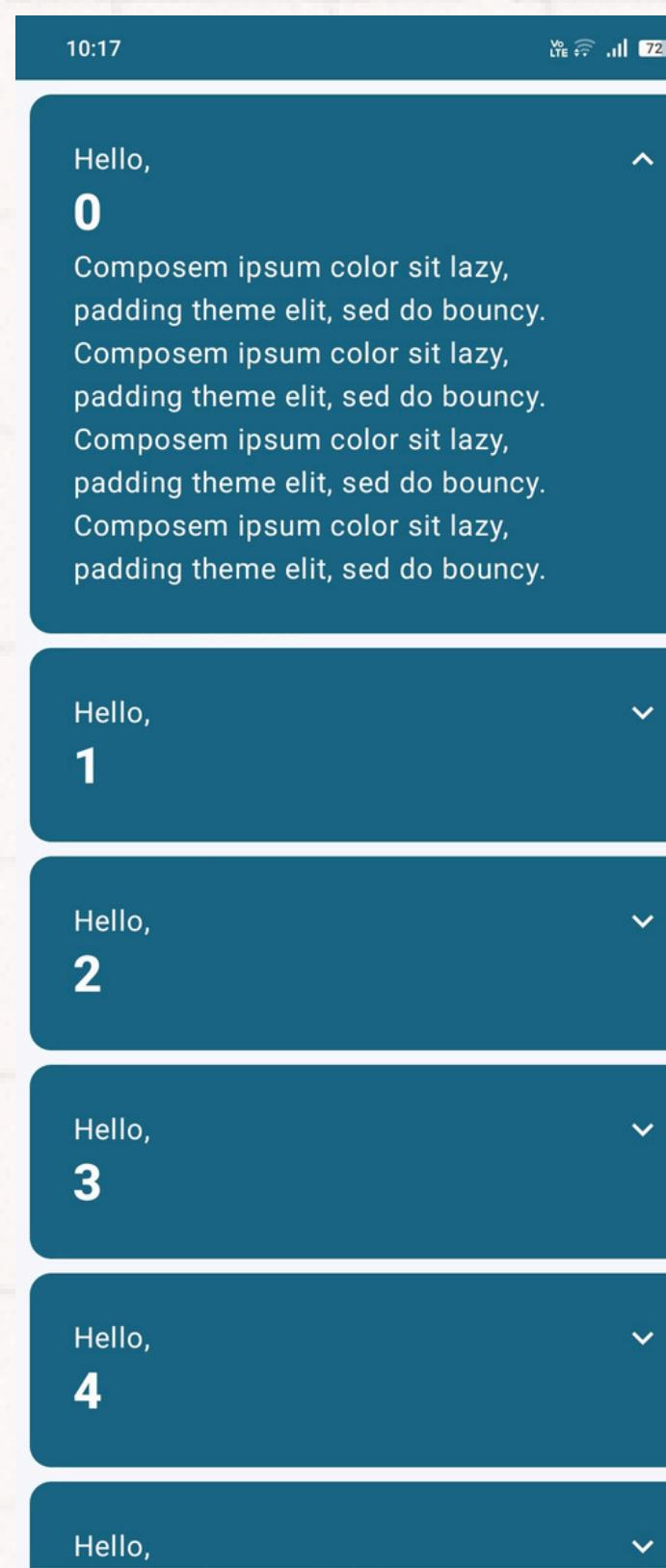
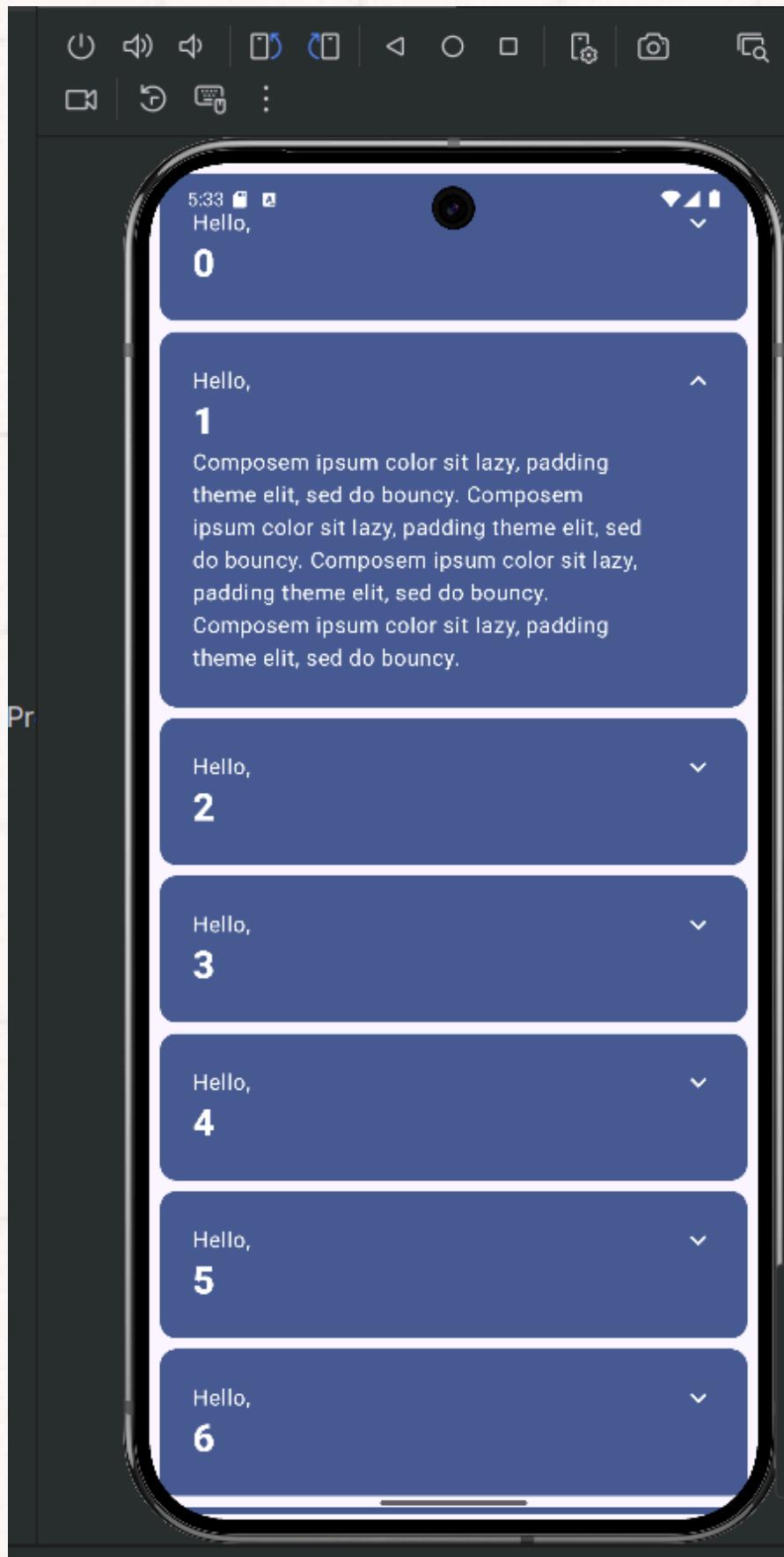
Continue



En este proyecto se comprendió:

- Qué es Compose
- Cómo compilar IUs con Compose
- Cómo administrar el estado en funciones de componibilidad
- Cómo crear una lista de rendimiento
- Cómo agregar animaciones
- Cómo aplicarle estilos y temas a una app

Dando como resultado una app con una pantalla de integración y una lista de elementos animados expandidos

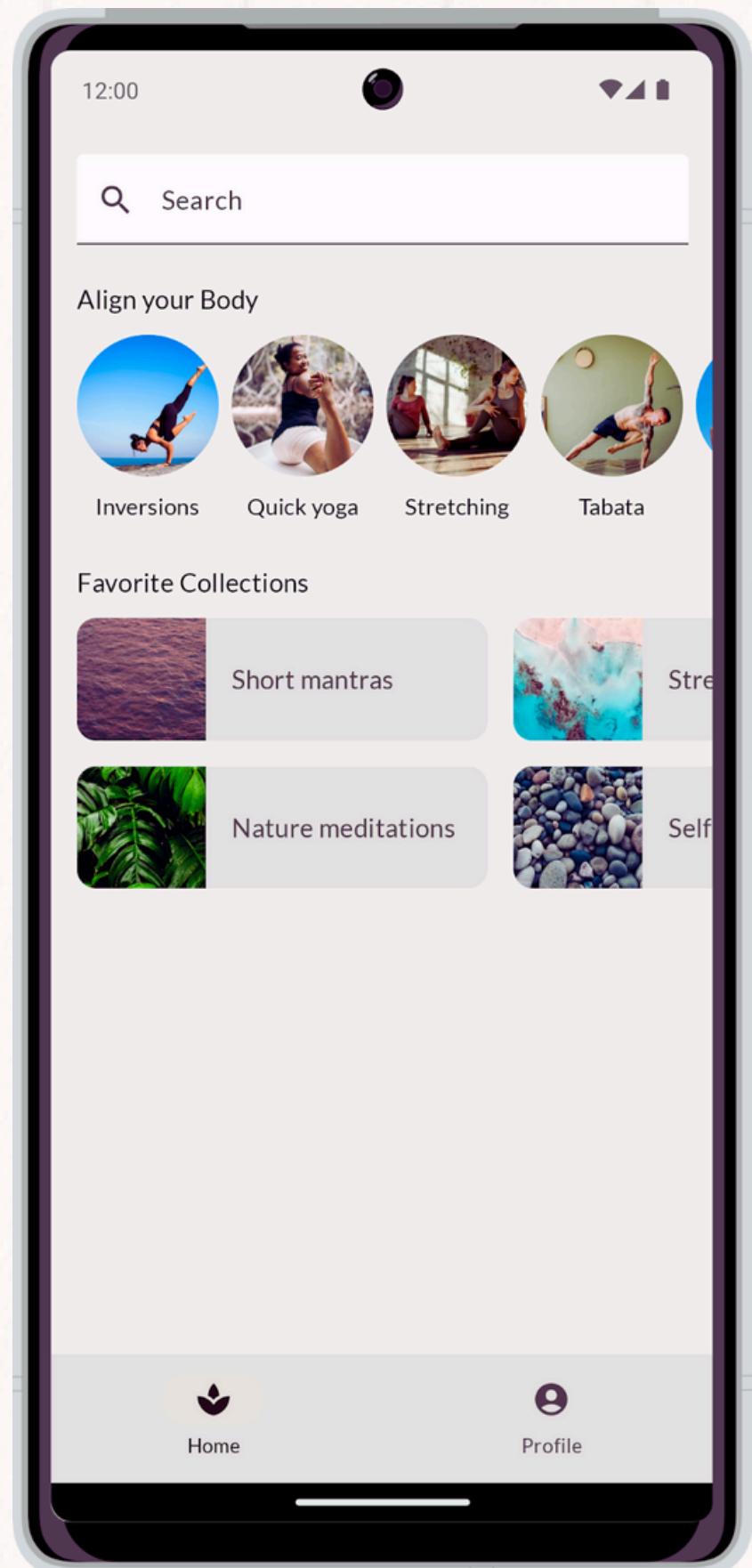


Este proyecto introduce los fundamentos de Jetpack Compose mediante la creación de una aplicación móvil básica.

- Uso de componentes básicos (@Composable)
- Implementación de padding y temas
- Creación de listas o elementos repetitivos

Curso de jetpack compose android

# DISEÑOS BÁSICOS EN COMPOSE



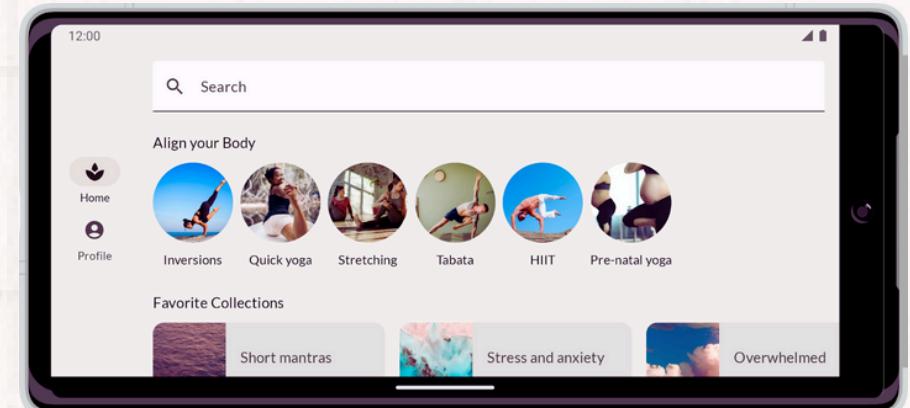
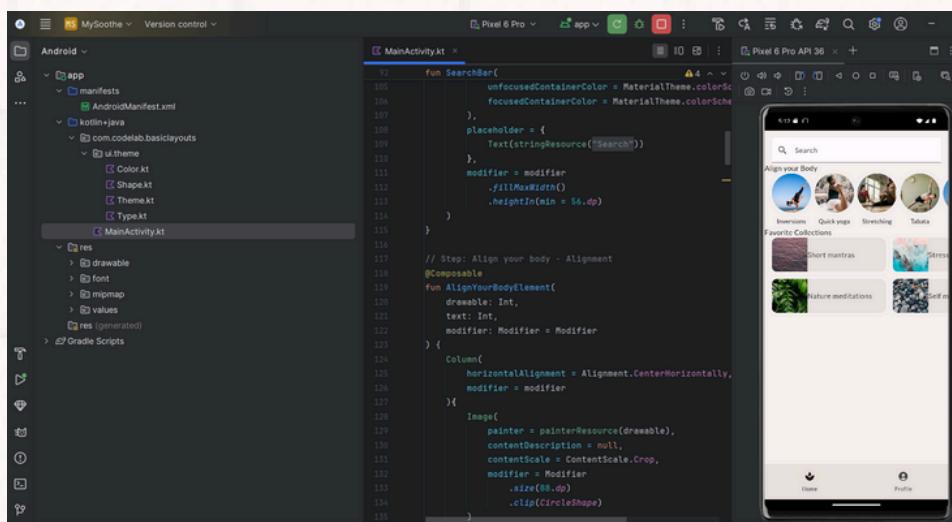
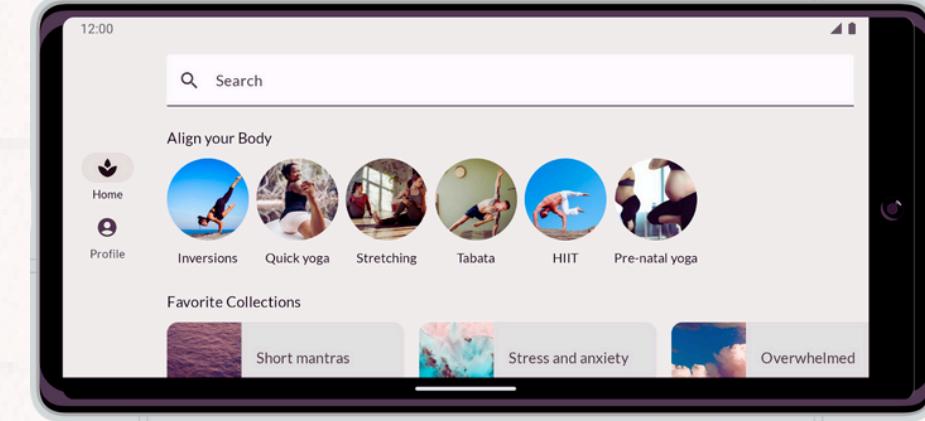
En este Se vieron los fundamentos del diseño de interfaces en Jetpack Compose.

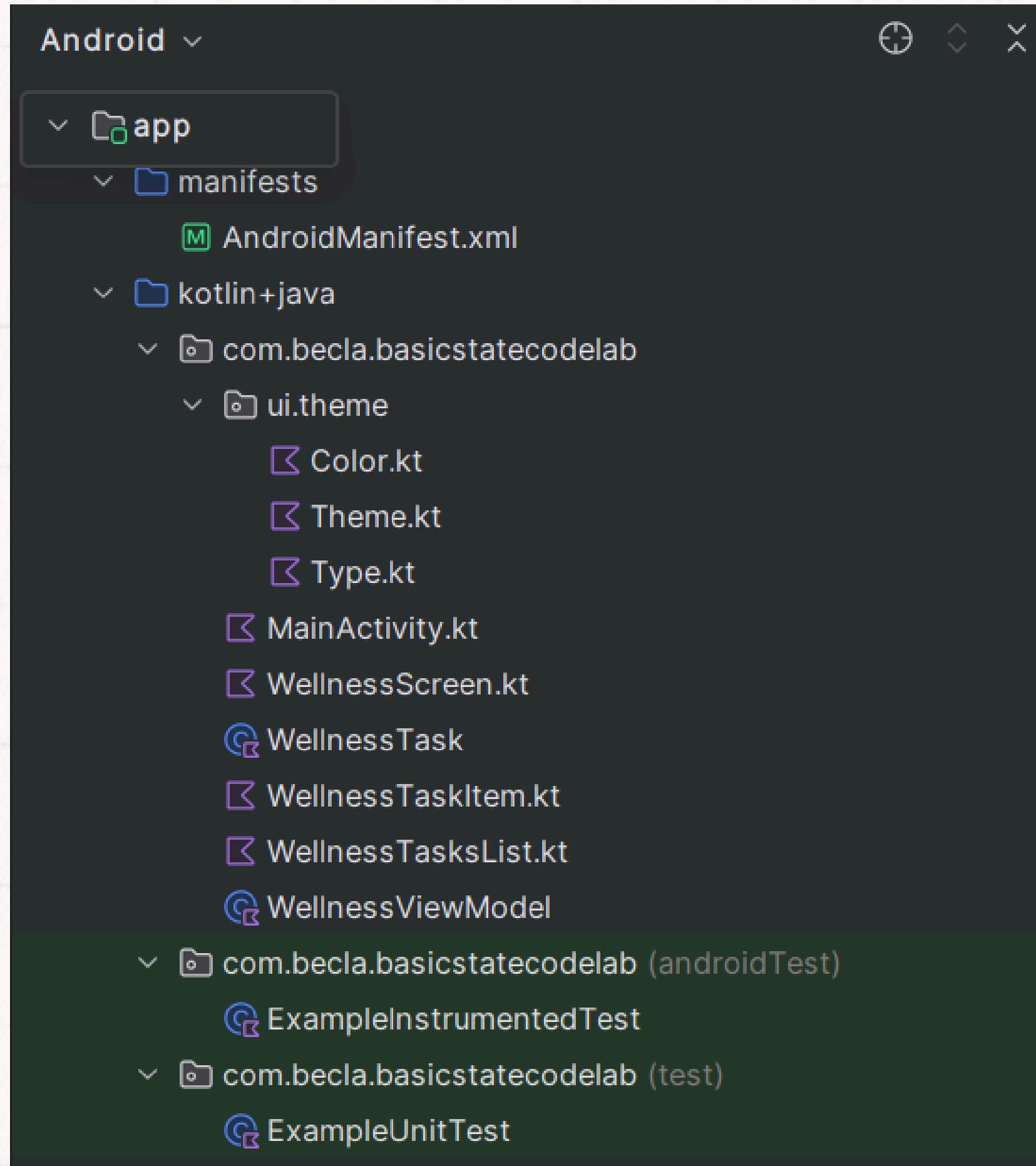
También cómo funcionan los componentes de diseño estándar, como Column y LazyRow, para posicionar y organizar elementos secundarios.

Además, de los alineamientos y arreglos influyen en la distribución de los elementos dentro de sus contenedores.

Fue el Codelab con el que más me tardé ya que las versiones me causaban conflicto. Así como el programa me empezó a pedir permiso y a ser bloqueado con el Antivirus (Este problema lo resolví más adelante).

Pero en este codelab se comprendieron también temas de versiones.



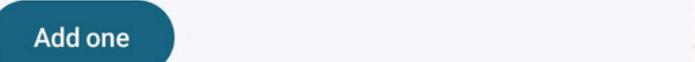
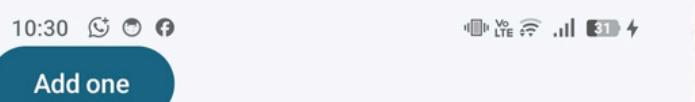


Este proyecto introduce los fundamentos de Jetpack Compose mediante la creación de una aplicación móvil básica.

- Uso de componentes básicos (`@Composable`)
- Implementación de padding y temas
- Creación de listas o elementos repetitivos

Curso de jetpack compose android

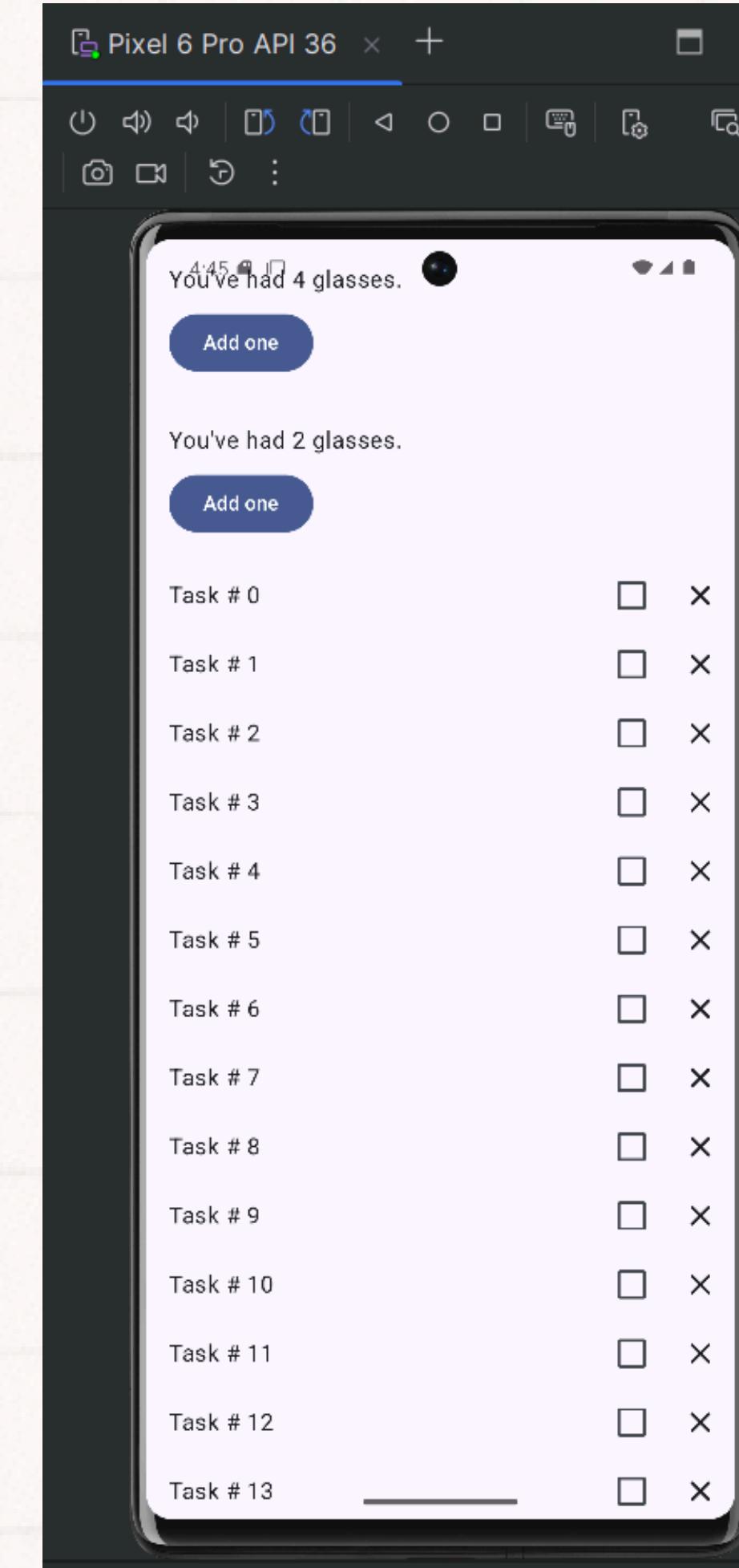
ESTADO EN  
JETPACK COMPOSE



Task # 0	<input type="checkbox"/> X
Task # 1	<input type="checkbox"/> X
Task # 2	<input type="checkbox"/> X
Task # 3	<input type="checkbox"/> X
Task # 4	<input type="checkbox"/> X
Task # 5	<input type="checkbox"/> X
Task # 6	<input type="checkbox"/> X
Task # 7	<input type="checkbox"/> X
Task # 8	<input type="checkbox"/> X
Task # 9	<input type="checkbox"/> X
Task # 10	<input type="checkbox"/> X
Task # 11	<input type="checkbox"/> X
Task # 12	<input type="checkbox"/> X
Task # 13	<input type="checkbox"/> X
...	...

Este proyecto se enfoca en el manejo de estado en Jetpack Compose. La aplicación implementa dos funciones principales:

- Manejo de Estado: Cómo el estado determina lo que se muestra en la UI
- Actualización de UI: Cómo Compose actualiza la interfaz cuando cambia el estado
- Estructura de funciones componibles: Optimización de componentes
- Integración con ViewModel: Uso en un entorno Compose



The screenshot shows the Android Studio interface. On the left, the code editor displays `MainActivity.kt` with the following code:

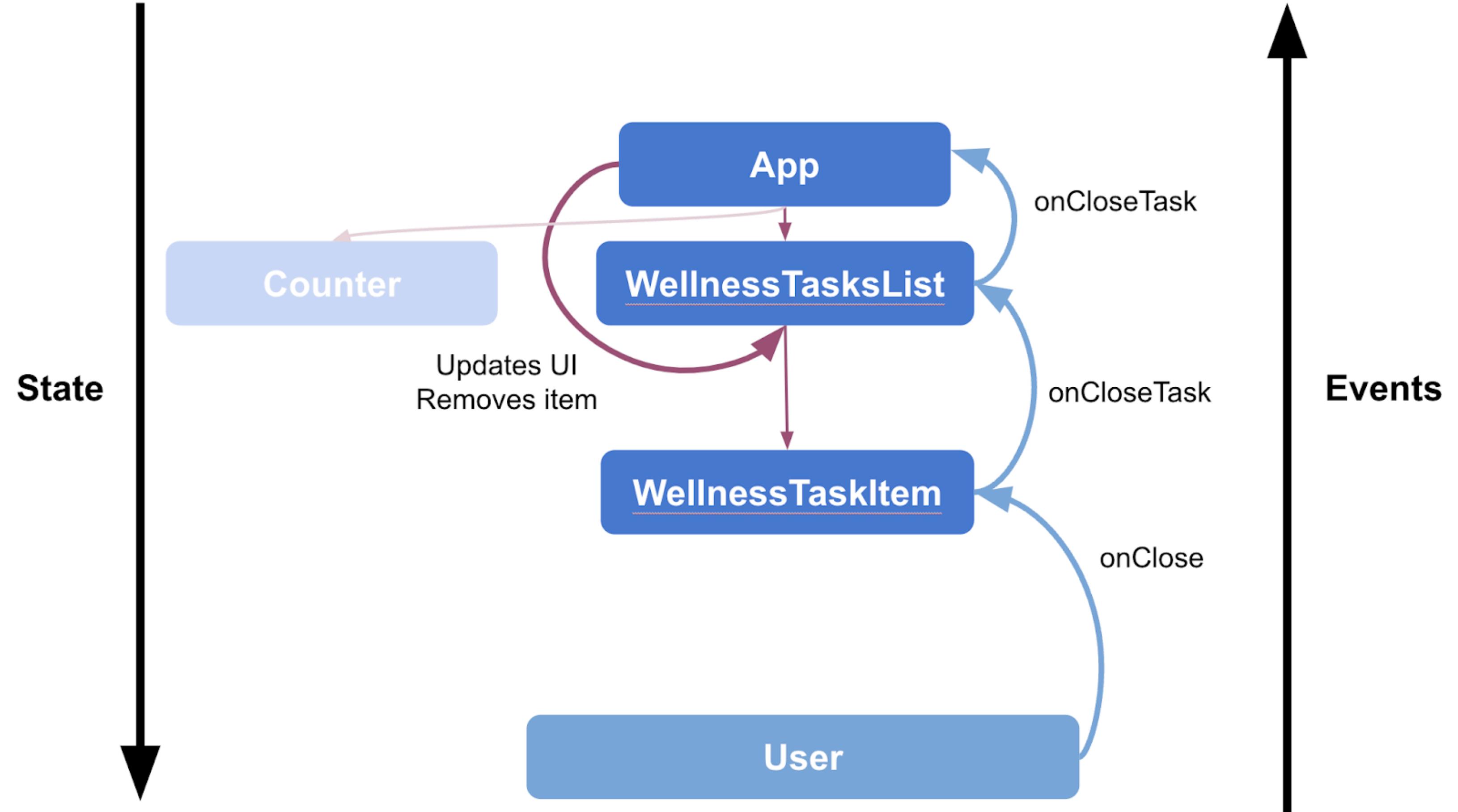
```
45     fun StatelessCounter(count: Int, onIncrement: () -> Unit, modifier: Modifier = Modifier) {  
46         }  
47  
48     @Composable  
49     fun WaterCounter(modifier: Modifier = Modifier) {  
50         Column(modifier = modifier.padding(16.dp)) {  
51             var count by rememberSaveable { mutableStateOf(0) }  
52             if (count > 0) {  
53                 Text("You've had $count glasses.")  
54             }  
55             Button(onClick = { count++ }, Modifier.padding(top = 8.dp), enabled = count < 10) {  
56                 Text("Add one")  
57             }  
58         }  
59     }  
60  
61     @Composable  
62     fun StatefulCounter() {  
63         var waterCount by remember { mutableStateOf(0) }  
64         var juiceCount by remember { mutableStateOf(0) }  
65  
66         StatelessCounter(waterCount, { waterCount++ })  
67         StatelessCounter(juiceCount, { juiceCount++ })  
68     }  
69 }  
70  
71 @Composable  
72 fun StatelessCounter(count: Int, onIncrement: () -> Unit, modifier: Modifier = Modifier) {  
73     var waterCount by remember { mutableStateOf(0) }  
74     var juiceCount by remember { mutableStateOf(0) }  
75  
76     StatelessCounter(waterCount, { waterCount++ })  
77     StatelessCounter(juiceCount, { juiceCount++ })  
78 }  
79 }
```

The right side of the interface shows the project structure under the 'Android' tab:

- app
  - manifests
    - AndroidManifest.xml
  - kotlin+java
    - com.becla.basicstatecodelab
      - ui.theme
        - Color.kt
        - Theme.kt
        - Type.kt
      - MainActivity.kt
      - WellnessScreen.kt
      - WellnessTask
      - WellnessTaskItem.kt
      - WellnessTasksList.kt
      - WellnessViewModel
    - com.becla.basicstatecodelab (androidTest)
      - ExampleInstrumentedTest
    - com.becla.basicstatecodelab (test)
      - ExampleUnitTest

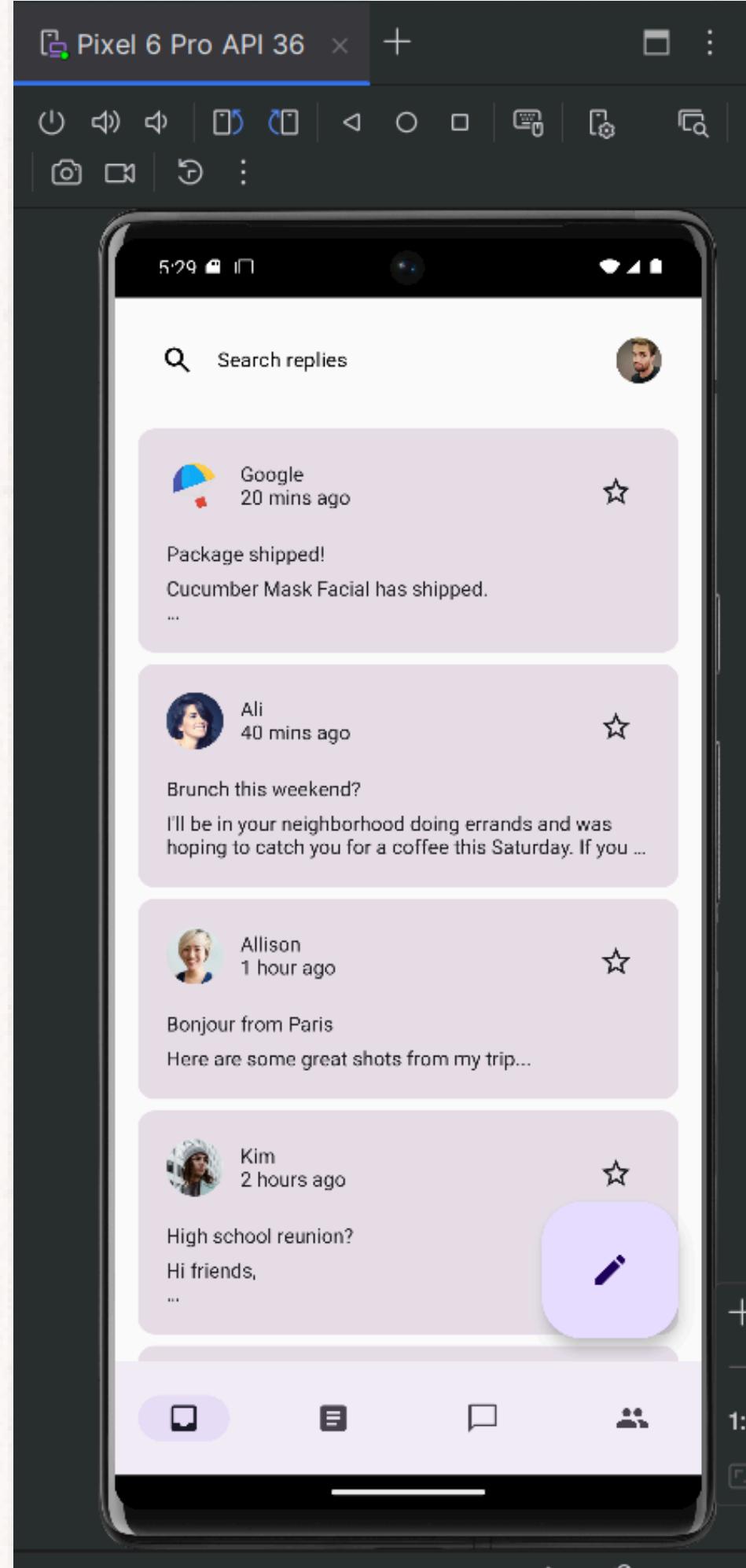
Este proyecto demuestra el comportamiento de Jetpack Compose para manejar estado:

1. `mutableStateOf` + `remember` para estado local (contador)
2. ViewModel para estado compartido (lista de tareas)
3. Componentes (`StatelessCounter`) que reciben estado por parámetros
4. Recomposición automática al actualizar el estado



Curso de jetpack compose android

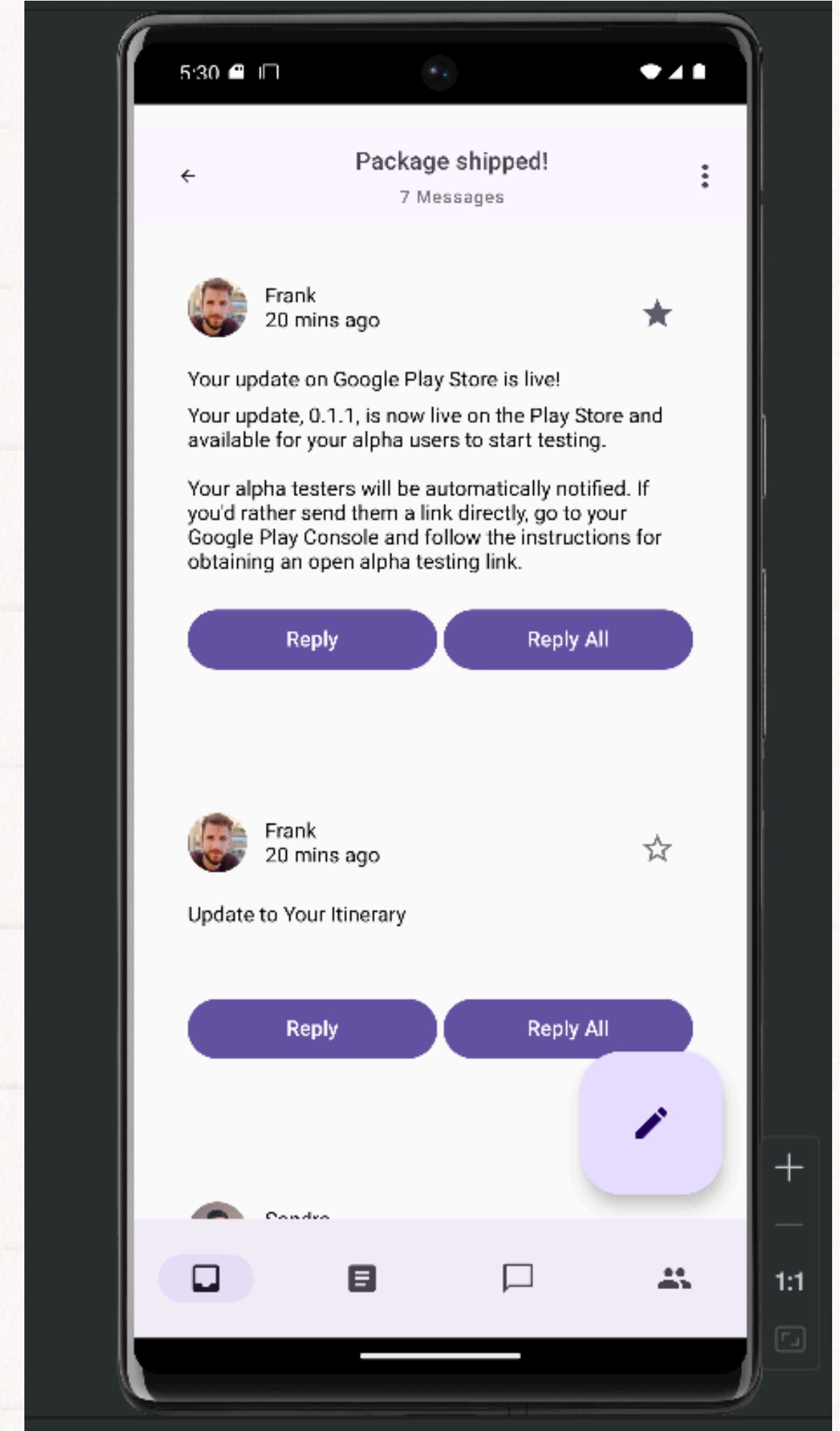
# TEMAS EN COMPOSE CON MATERIAL 3



En este codelab se aplicaron temas en Jetpack Compose usando Material Design 3 (M3), tomando como ejemplo la app de correo "Reply".

El proyecto demostró cómo los temas unificados transforman completamente la experiencia de usuario, como se ve en los ejemplos de pantallas de correo electrónico (listado y detalle).

- Implementar esquemas de color personalizados
- Configurar tipografía y formas coherentes
- Adicionar soporte para temas claro, oscuro y dinámico
- Personalizar componentes de Material 3



The screenshot shows an Android Studio interface. On the left is a code editor with Kotlin code for a Compose application. The code includes functions like `ReplyApp` and `ReplyAppContent`, and a `val selectedDestination` declaration. On the right is a file browser titled "Android" showing the project structure:

- `app`:
  - `manifests`
  - `kotlin+java`:
    - `com.example.reply`:
      - `data`:
        - `Account`
        - `Email.kt`
        - `LocalAccountsDataProvider`
        - `LocalEmailsDataProvider`
      - `ui`:
        - `components`:
          - `ReplyAppBars.kt`
          - `ReplyEmailListItem.kt`
          - `ReplyEmailThreadItem.kt`
          - `ReplyProfileImage.kt`
        - `theme`:
          - `Color.kt`
          - `Theme.kt`
          - `Type.kt`
          - `EmptyComingSoon.kt`
          - `MainActivity.kt`
          - `ReplyApp.kt` (highlighted with a blue selection bar)
          - `ReplyHomeViewModel.kt`
          - `ReplyListContent.kt`
    - `res`
    - `res (generated)`

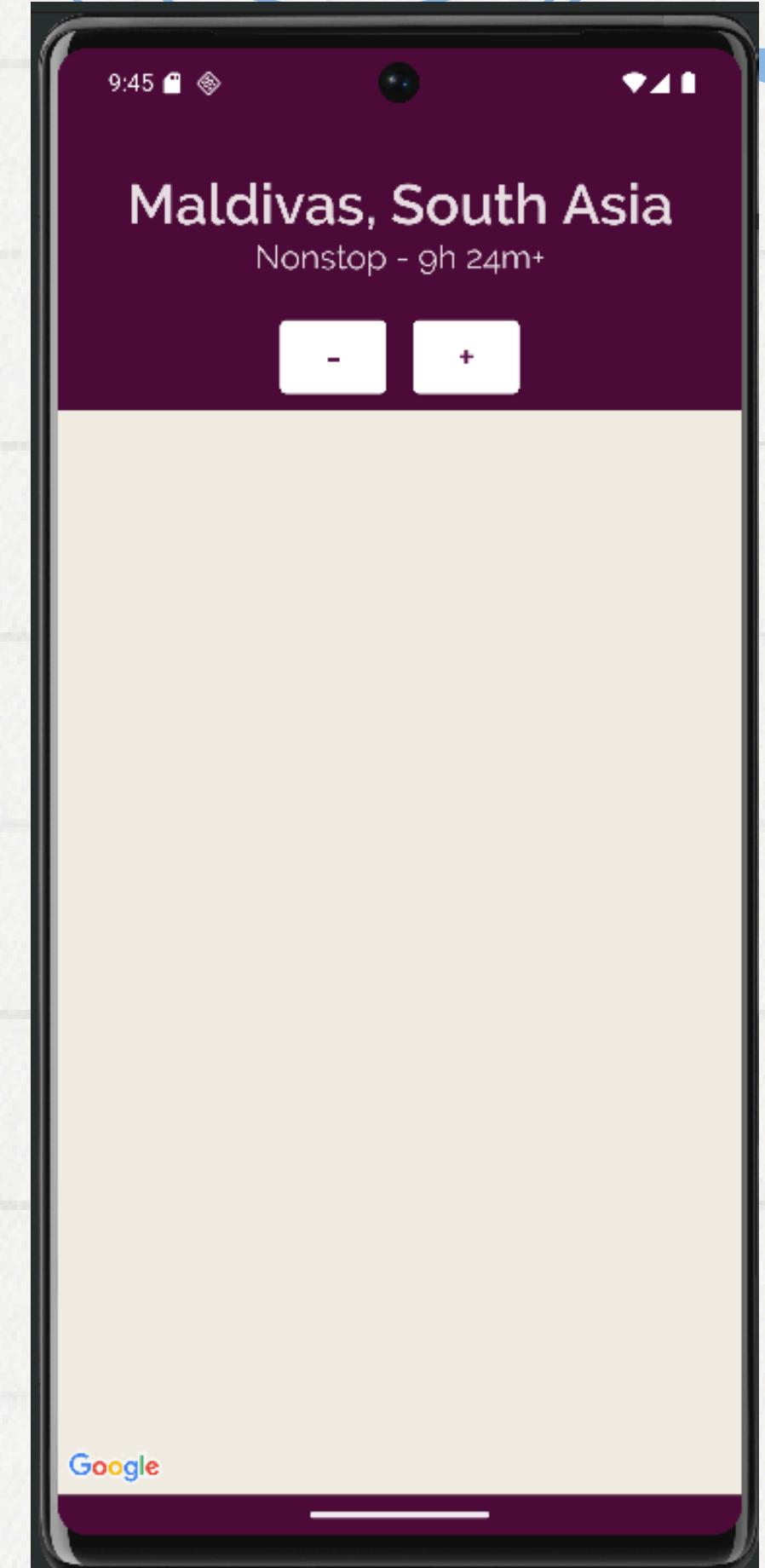
La implementación de temas en Compose nos permitió:

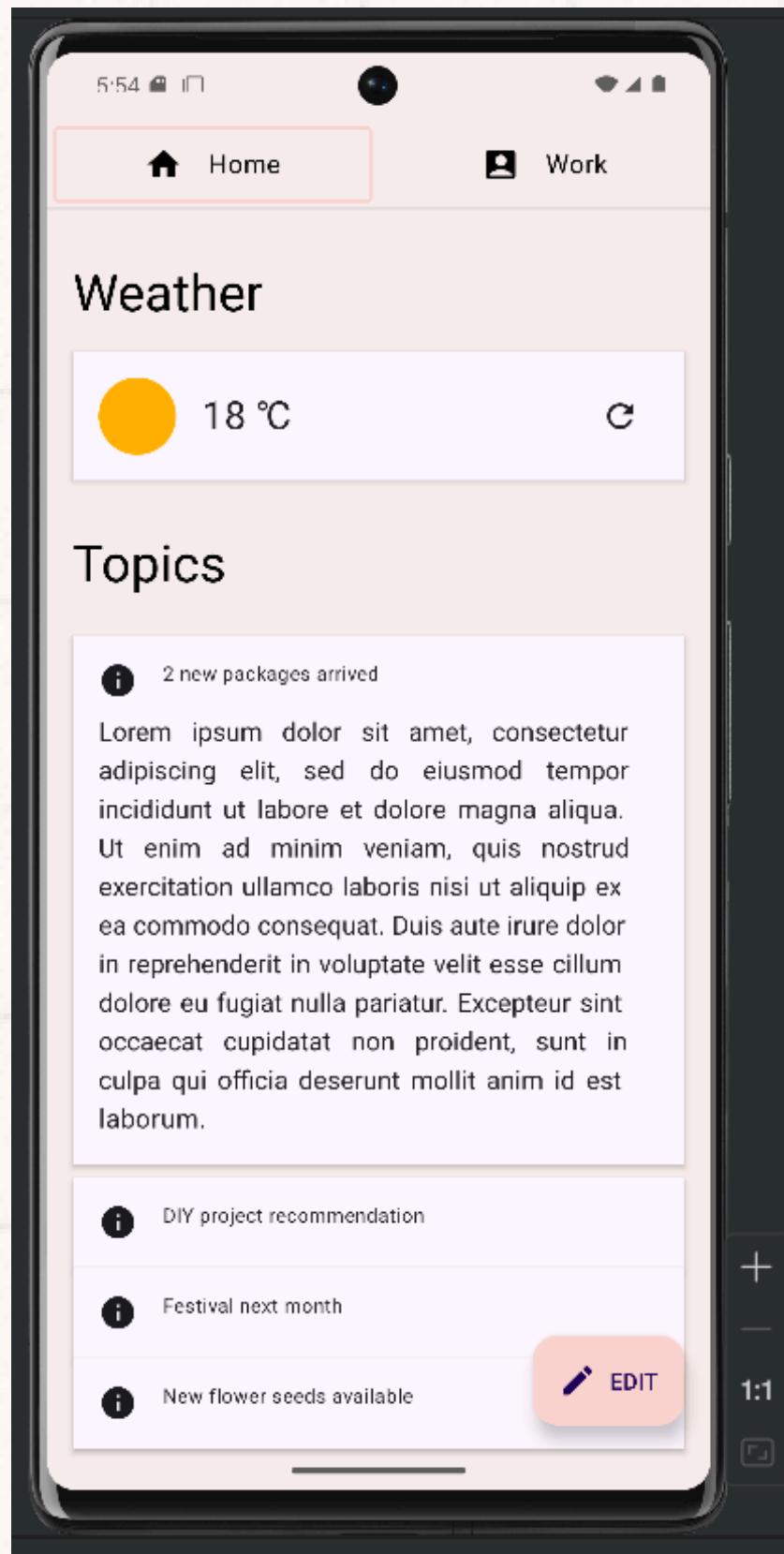
- Una buena presentación visual: Usando esquemas de color personalizados
- Mejorar accesibilidad: Con contrastes adecuados en ambos modos (claro/oscuro)
- Implementar temas dinámicos: Que responden al sistema operativo
- Mantener consistencia: A través de componentes estandarizados

Este enfoque simplifica enormemente el diseño de interfaces profesionales y accesibles, demostrando que con Compose podemos lograr apps visualmente atractivas con menos código que en el sistema tradicional de vistas.

Curso de jetpack compose android

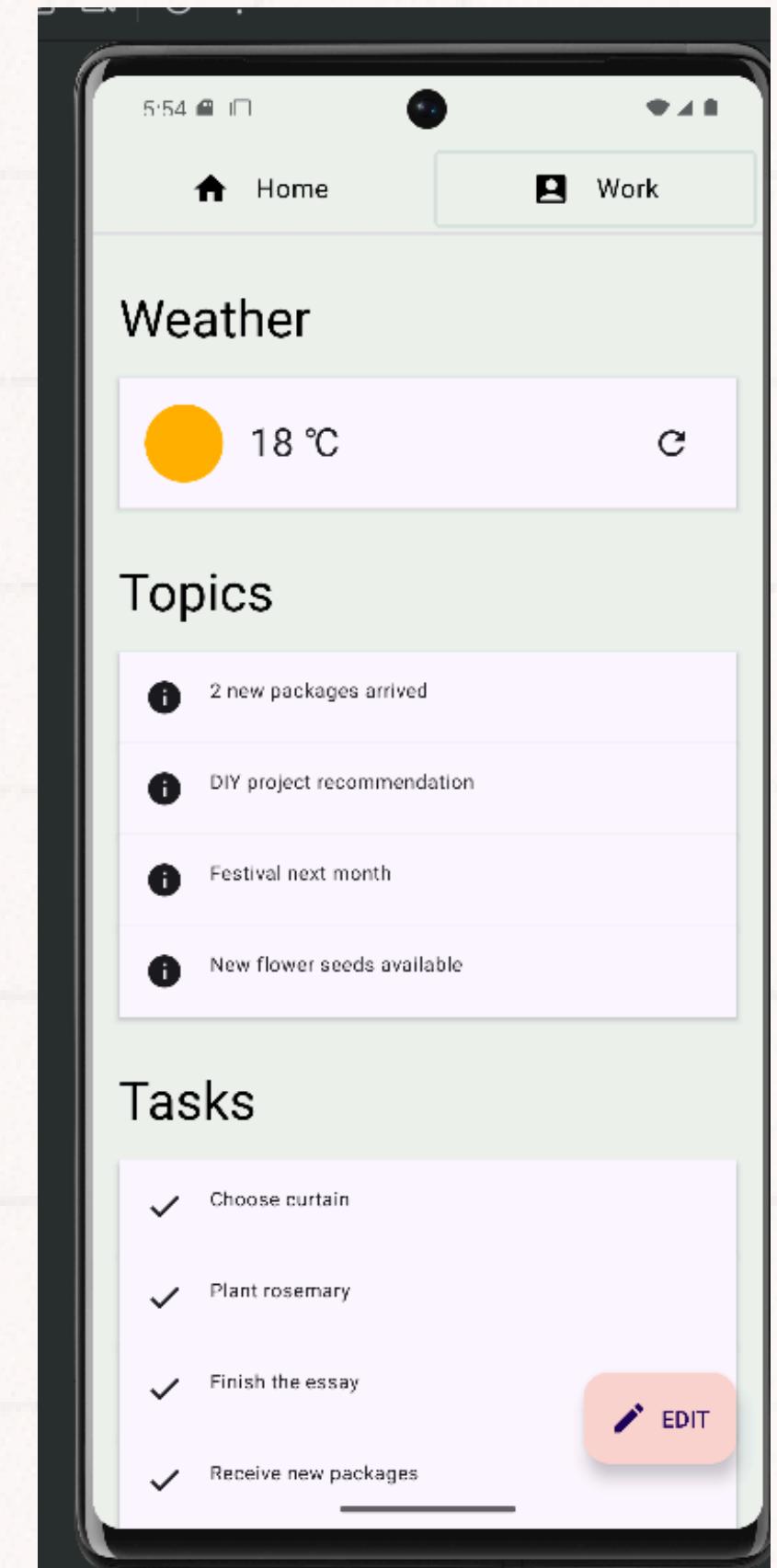
# CÓMO ANIMAR ELEMENTOS EN JETPACK COMPOSE





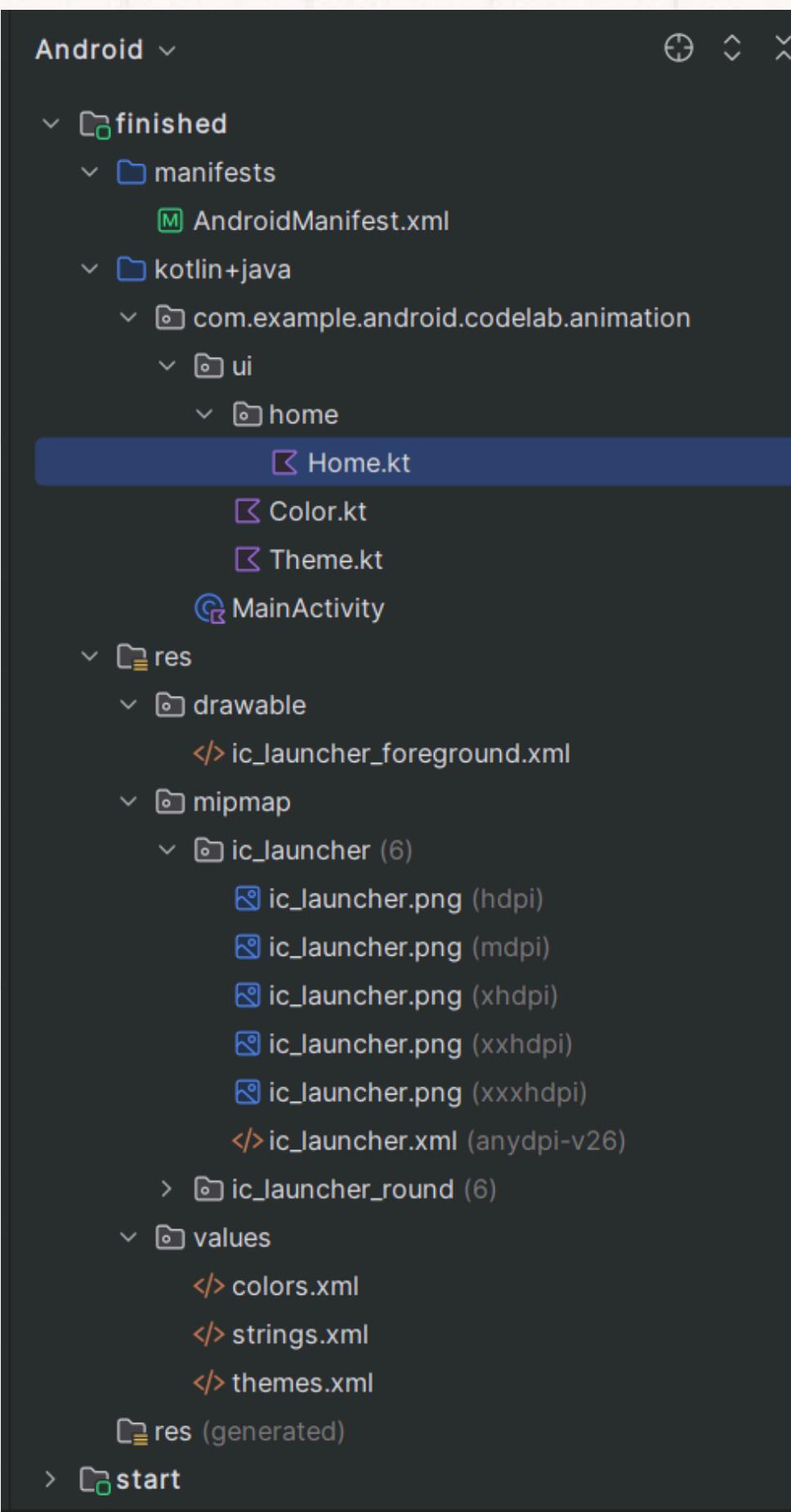
Este proyecto demuestra el poder del sistema de animaciones en Jetpack Compose, implementando múltiples patrones de animación en una app de gestión de tareas y clima. A través del código, vemos cómo Compose simplifica la creación de:

- 1. Animaciones de estado:** Transiciones suaves entre pantallas (Home/Work)
- 2. Microinteracciones:** Efectos al completar tareas (swipe to dismiss)
- 3. Indicadores visuales:** Carga animada del clima
- 4. Transiciones complejas:** Animación del FAB (Floating Action Button)



The screenshot shows the Android Studio interface with two tabs open: Home.kt and MainActivity.kt. The Home.kt file is the active tab, displaying Kotlin code for a Jetpack Compose application. The code includes components like Box, LazyColumn, and FloatingActionButton, along with state management using remember and mutableStateOf. The MainActivity.kt file is also visible in the background.

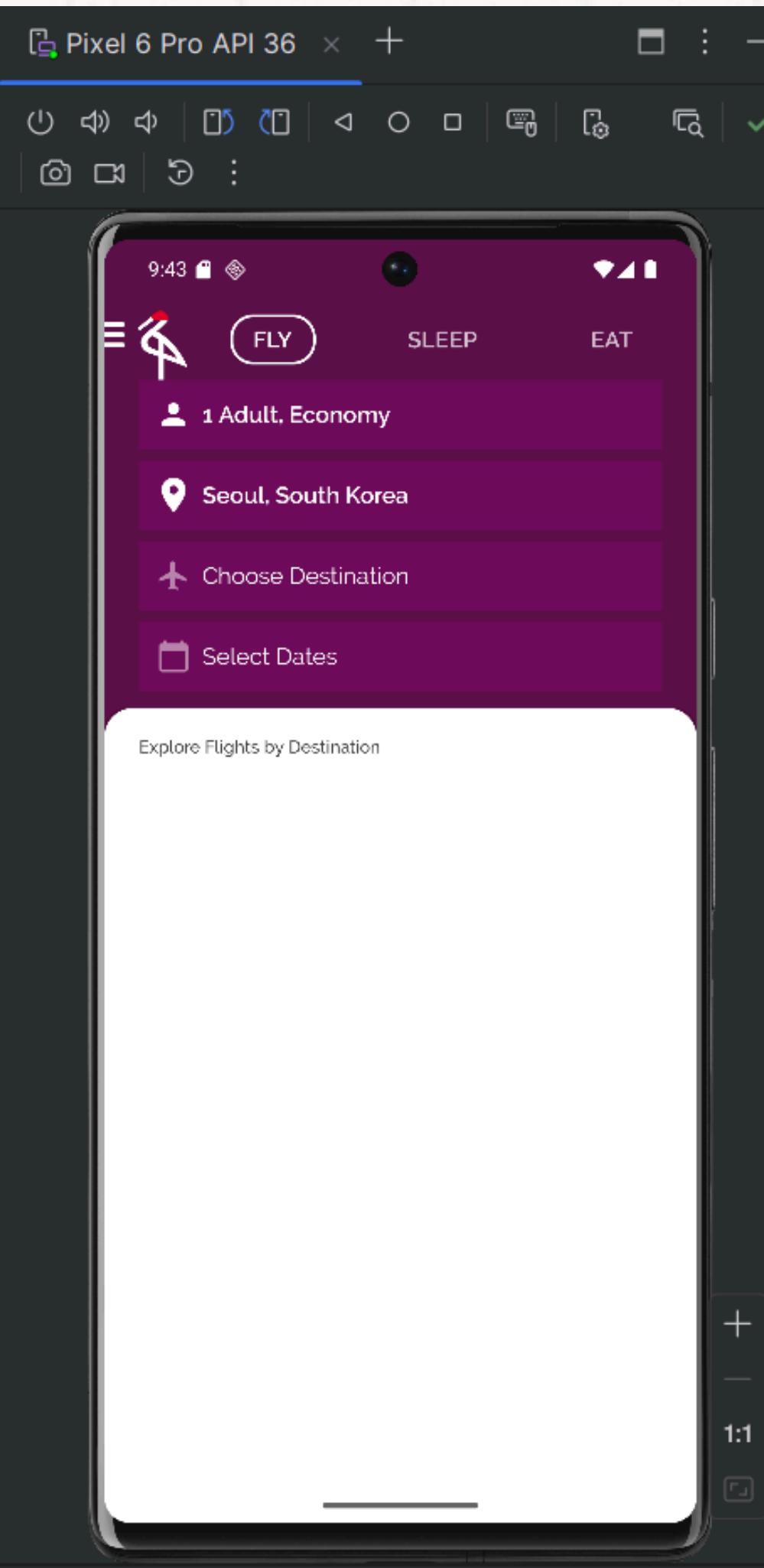
```
145 fun Home() {
146     topBar = {
147         backgroundColor = backgroundColor,
148         tabPage = tabPage,
149         onTabSelected = { tabPage = it }
150     }
151     containerColor = backgroundColor,
152     floatingActionButton = {
153         HomeFloatingActionButton(
154             extended = lazyListState.isScrollingUp(),
155             onClick = {
156                 coroutineScope.launch {
157                     showEditMessage()
158                 }
159             }
160         )
161     }
162 } { padding ->
163     Box(modifier = Modifier.padding(
164         top = padding.calculateTopPadding(),
165         start = padding.calculateLeftPadding(LayoutDirection.Ltr),
166         end = padding.calculateEndPadding(LayoutDirection.Ltr)
167     )) {
168         LazyColumn(
169             contentPadding = WindowInsets(16.dp, 32.dp, 16.dp,
170                 padding.calculateBottomPadding() + 32.dp
171             ).asPaddingValues(),
172             state = lazyListState
173         ) {
174             // Weather
175             item { Header(title = stringResource("Weather")) }
176         }
177     }
178 }
```



Este proyecto demuestra cómo Jetpack Compose actúa con las animaciones en Android mediante un enfoque declarativo que integra fluidamente transiciones de estado (como el cambio entre pestañas Home/Work), microinteracciones (swipe-to-dismiss), efectos visuales (carga del clima) y componentes dinámicos (FAB), utilizando APIs concisas como animate\*AsState para animaciones básicas y updateTransition para flujos complejos, todo optimizado para ejecutarse eficientemente en el hilo UI sin bloqueos gracias al sistema de recomposición, reduciendo significativamente el código frente al enfoque tradicional de Views mientras se mantiene un rendimiento excelente y una curva de aprendizaje intuitiva basada en reactividad y composición modular.

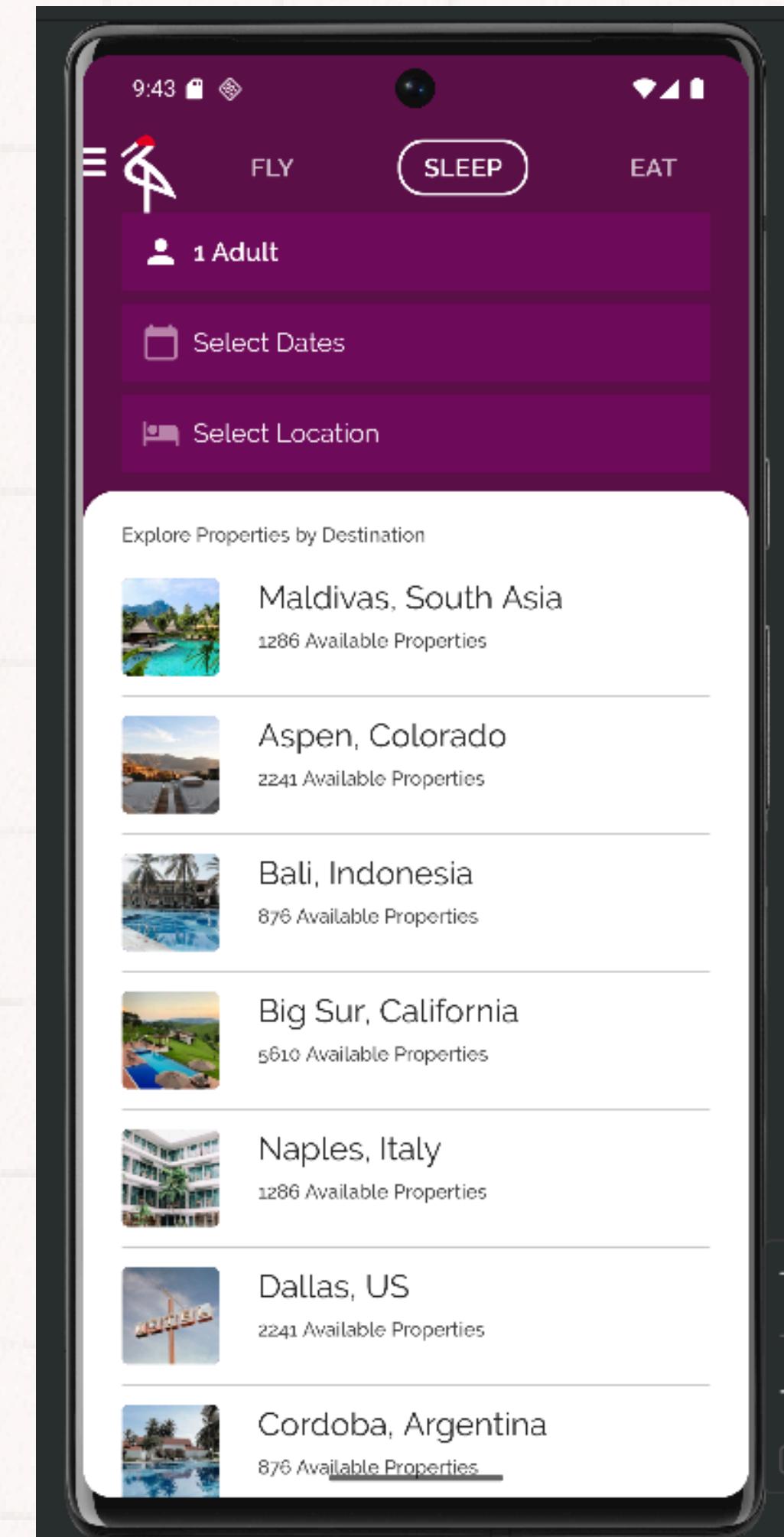
Curso de jetpack compose android

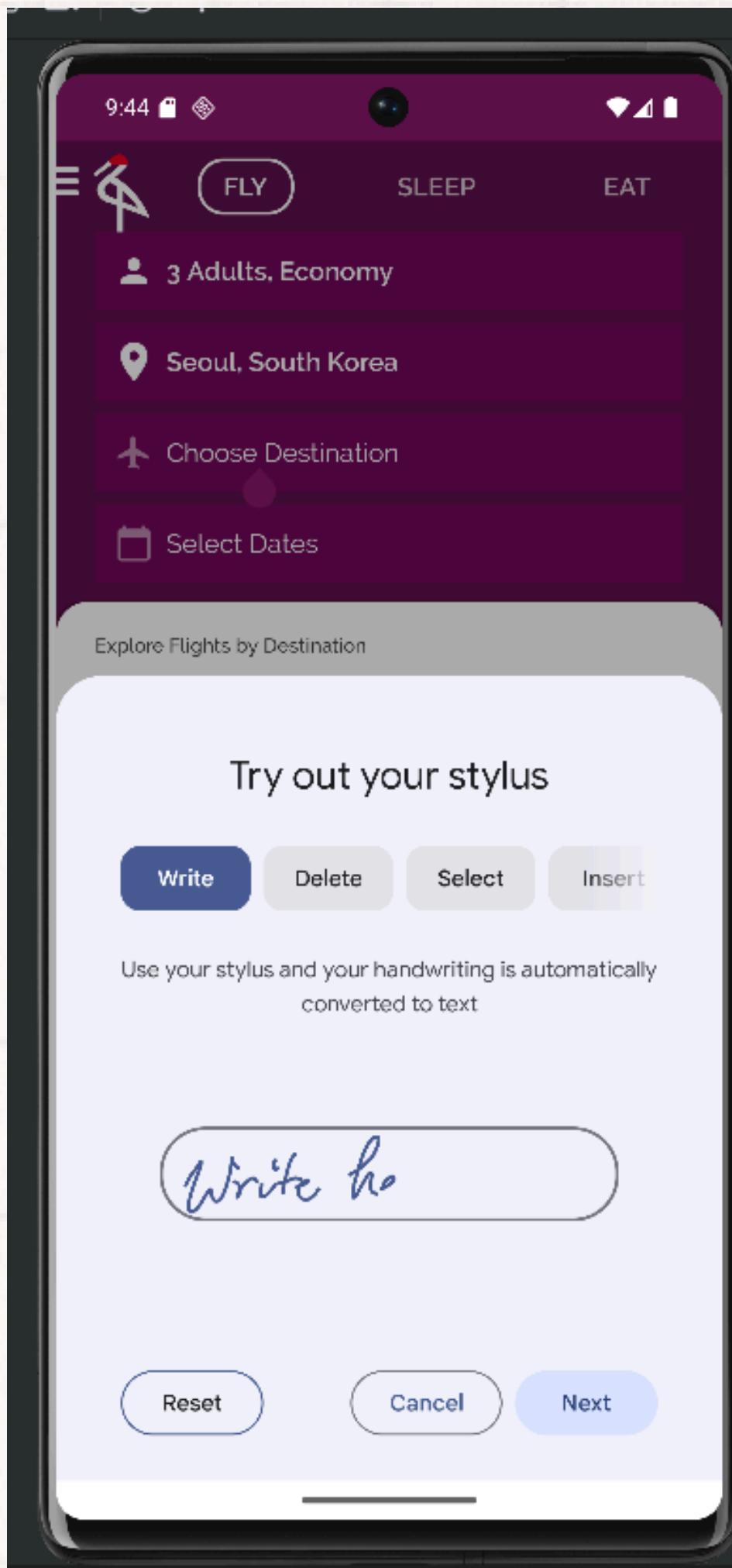
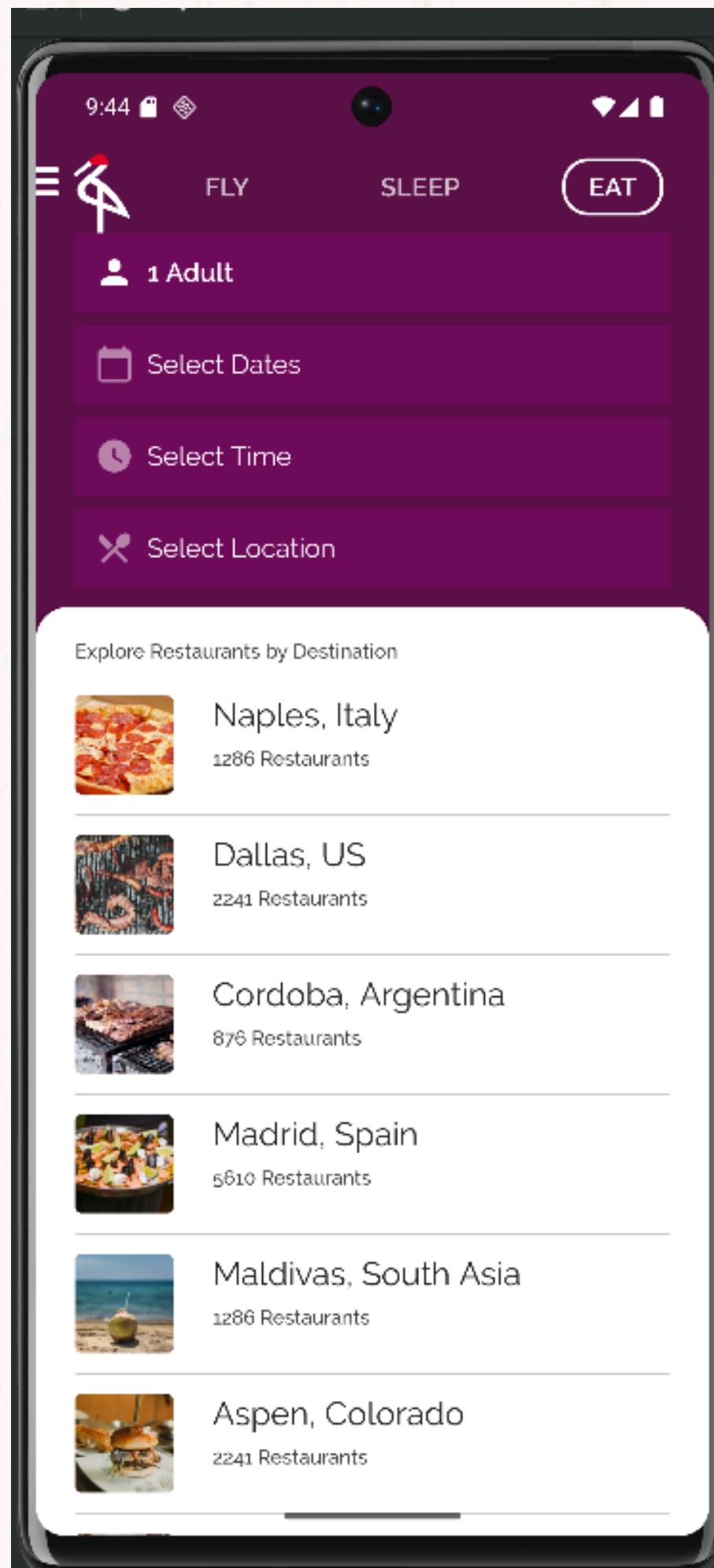
# **EFFECTOS SECUNDARIOS Y ESTADOS AVANZADOS EN JETPACK COMPOSE**



En este proyecto, muestra el manejo avanzado de estado y efectos secundarios en Jetpack Compose, aplicando conceptos clave para mejorar la app Crane (un estudio de materiales). Aprendimos a:

- Observar flujos de datos (Flow) para actualizar la UI reactivamente.
- Crear contenedores de estado para lógica compleja en componibles.
- Usar efectos secundarios como LaunchedEffect, DisposableEffect, produceState y derivedStateOf para manejar ciclos de vida y operaciones asíncronas.
- Lanzar corrutinas con rememberCoroutineScope para tareas suspendidas.

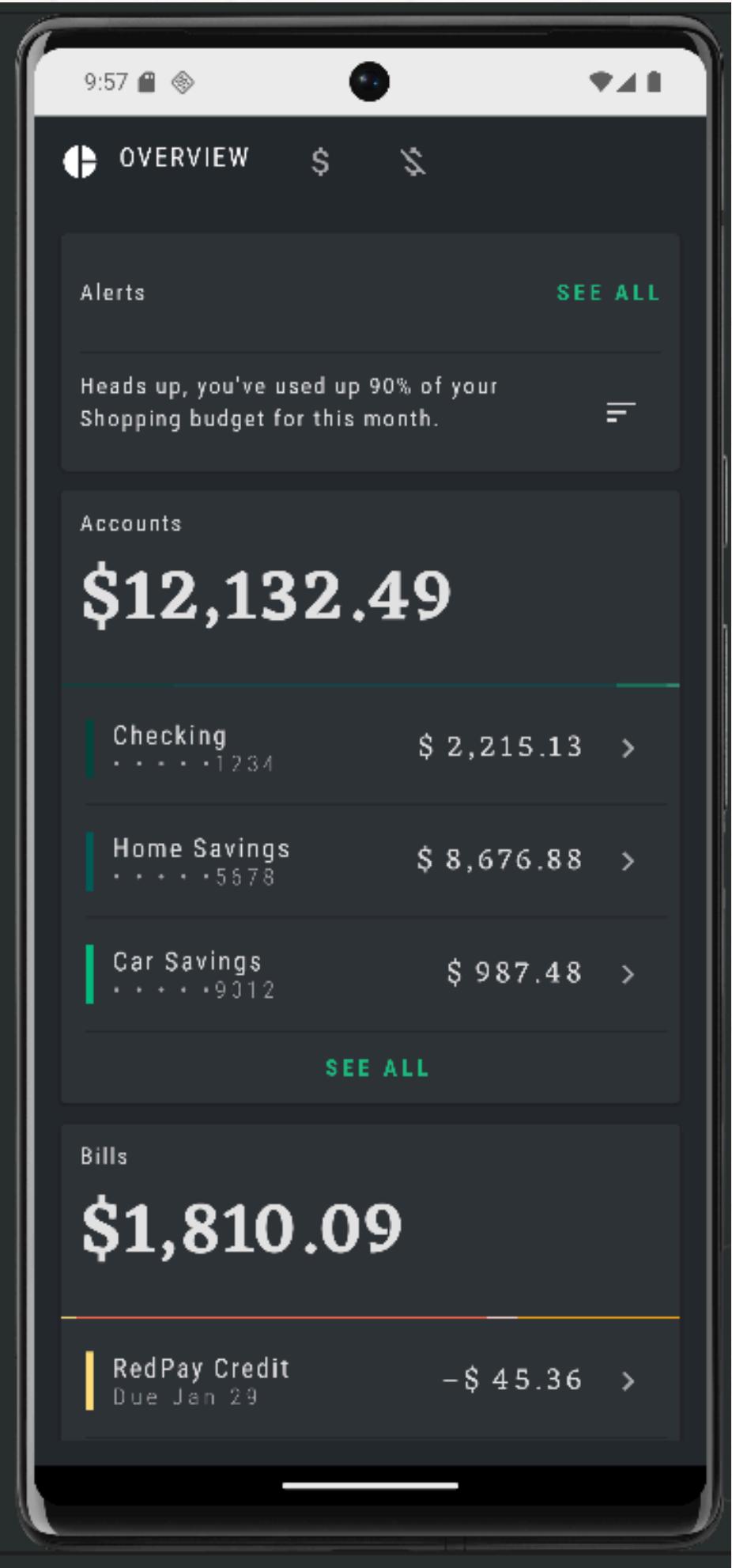




Este proyecto demostró el poder de las APIs avanzadas de Compose para manejar estados y efectos secundarios de manera declarativa. A través de técnicas como LaunchedEffect (para corrutinas en la UI), DisposableEffect (para limpieza de recursos), y produceState (para convertir flujos externos en estado), logramos una arquitectura escalable y mantenible. Además, el uso de derivedStateOf optimizó renders innecesarios, mientras que las corrutinas (rememberCoroutineScope) permitieron operaciones asíncronas sin romper el flujo reactivo.

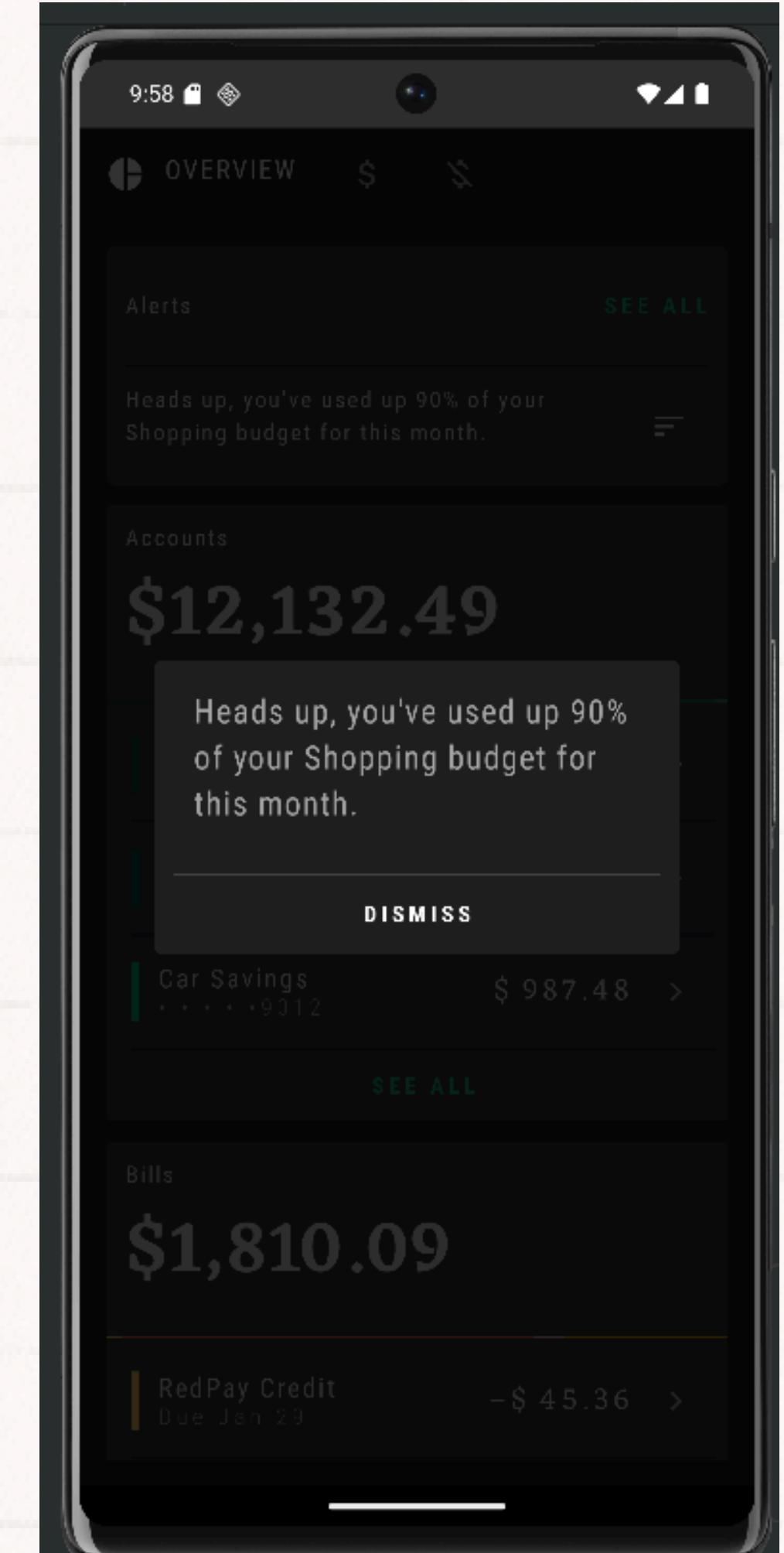
Curso de jetpack compose android

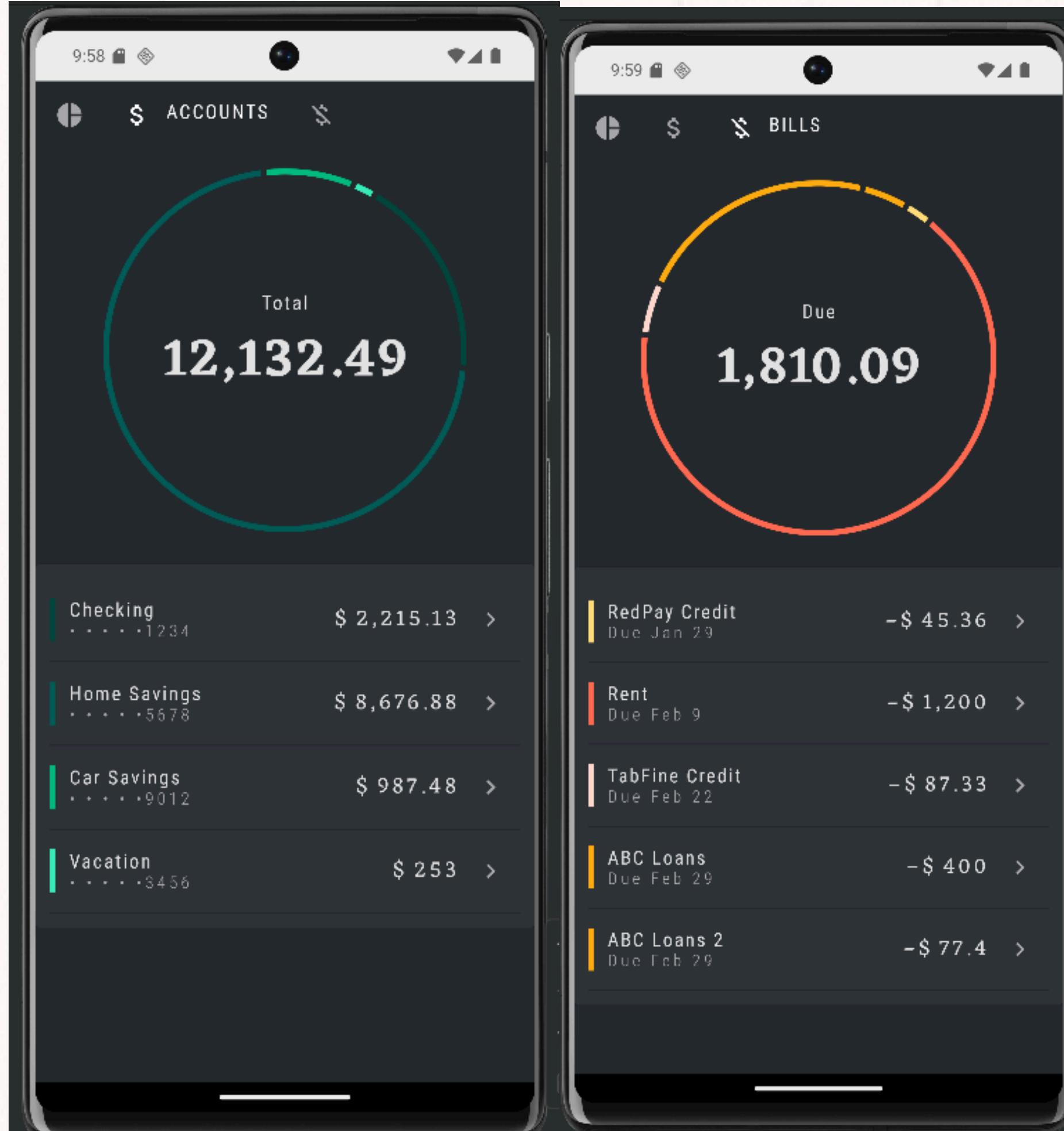
# NAVIGATION DE JETPACK COMPOSE



En este codelab, vimos la biblioteca de Navigation de Jetpack adaptada para Jetpack Compose, implementando un sistema de navegación. Aprendimos a:

- Configurar un gráfico de navegación con destinos componibles.
- Navegar entre pantallas usando rutas y acciones de navegación.
- Pasar argumentos entre destinos (como parámetros o datos complejos).
- Integrar una barra de pestañas personalizada en la jerarquía de navegación.
- Manejar vínculos directos (deep links) para acceder a pantallas específicas.
- Probar la navegación para garantizar su correcto funcionamiento.





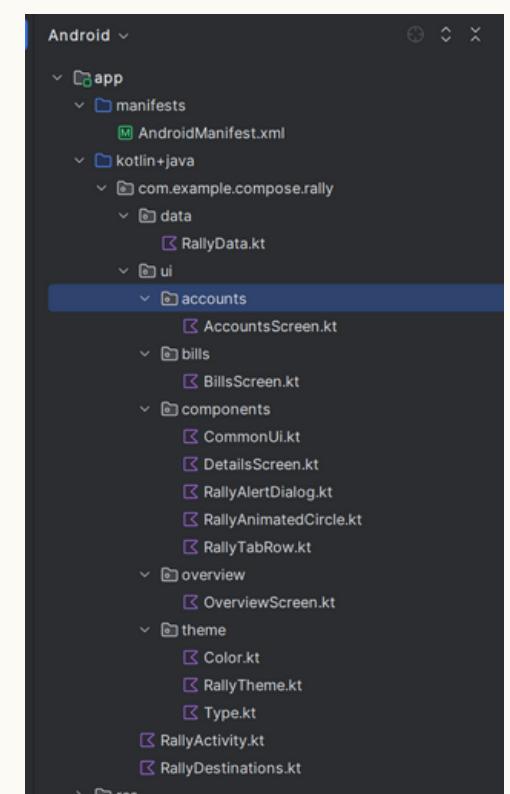
Este proyecto demostró cómo Navigation Compose simplifica la creación de flujos de pantallas complejos con un enfoque declarativo y tipo-safe. Mediante el uso de:

- NavController para gestionar la pila de navegación.
- NavHost como contenedor de destinos componibles.
- Argumentos tipados (parcelables, serializables o rutas con parámetros).
- Deep links para integrarse con otras apps o notificaciones.

```

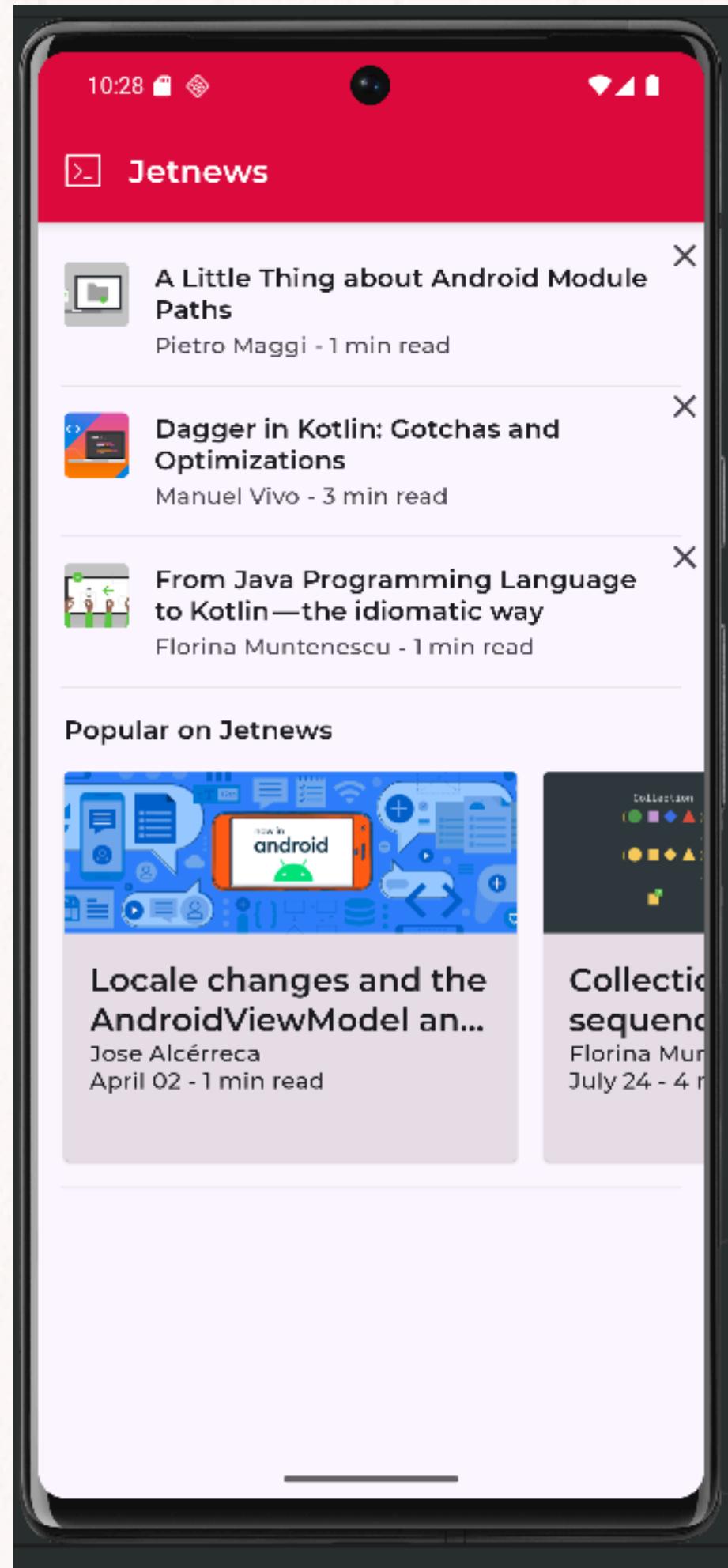
34 ① interface RallyDestination {
35 ②     val icon: ImageVector
36 ③     val route: String
37 ④     val screen: @Composable () -> Unit
38 ⑤ }
39
40 /**
41 * Rally app navigation destinations
42 */
43 object Overview : RallyDestination {
44     override val icon = Icons.Filled.PieChart
45     override val route = "overview"
46     override val screen: @Composable () -> Unit = { OverviewScreen() }
47 }
48
49 object Accounts : RallyDestination {
50     override val icon = Icons.Filled.AttachMoney
51     override val route = "accounts"
52     override val screen: @Composable () -> Unit = { AccountsScreen() }
53 }
54
55 object Bills : RallyDestination {
56     override val icon = Icons.Filled.MoneyOff
57     override val route = "bills"
58     override val screen: @Composable () -> Unit = { BillsScreen() }
59 }
60
61 object SingleAccount : RallyDestination {
62     // Added for simplicity, this icon will not in fact be used, as SingleAccount isn't
63     // part of the RallyTabRow selection
64     override val icon = Icons.Filled.Money
65     override val route = "single_account"
}

```



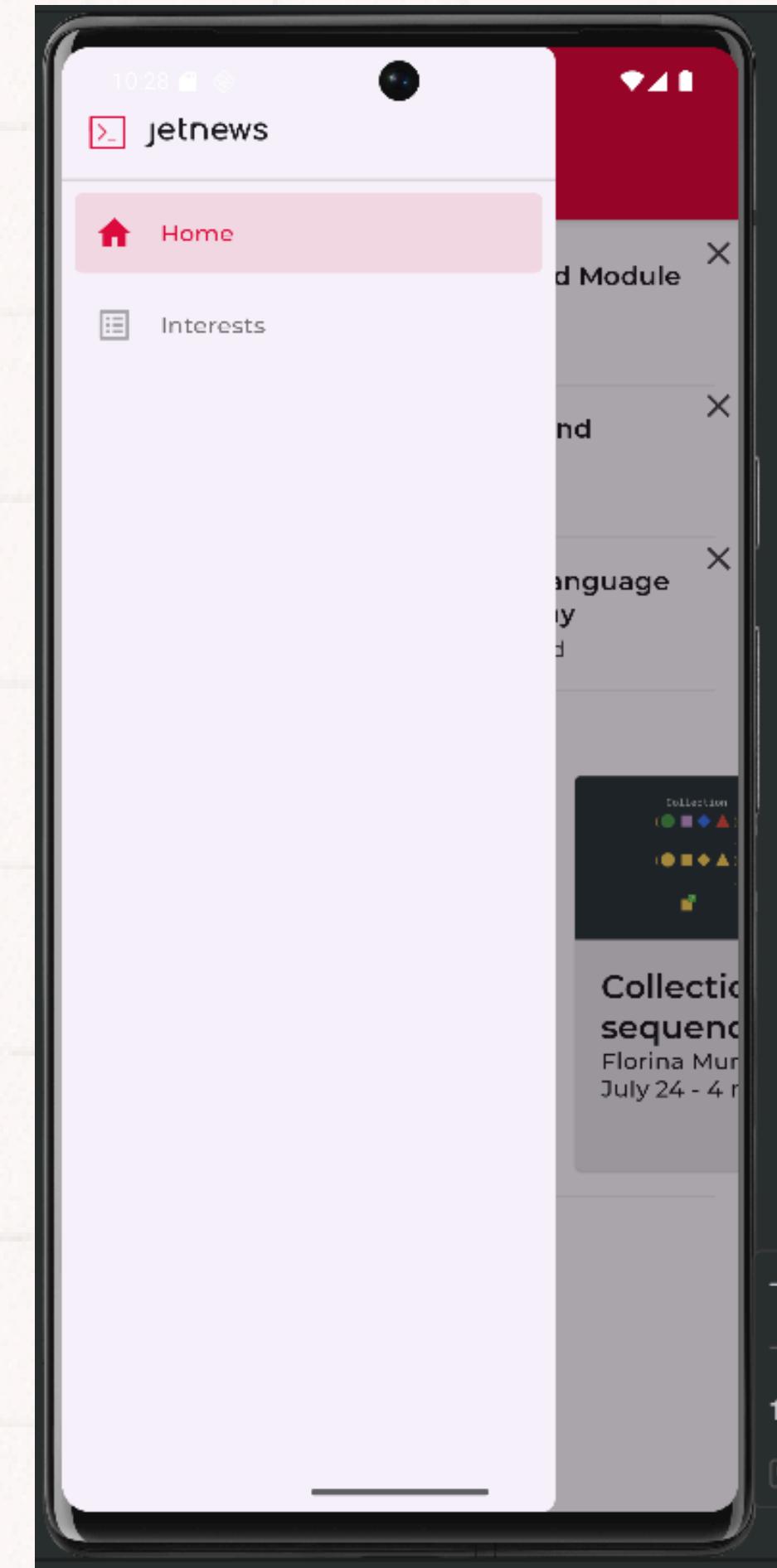
Curso de jetpack compose android

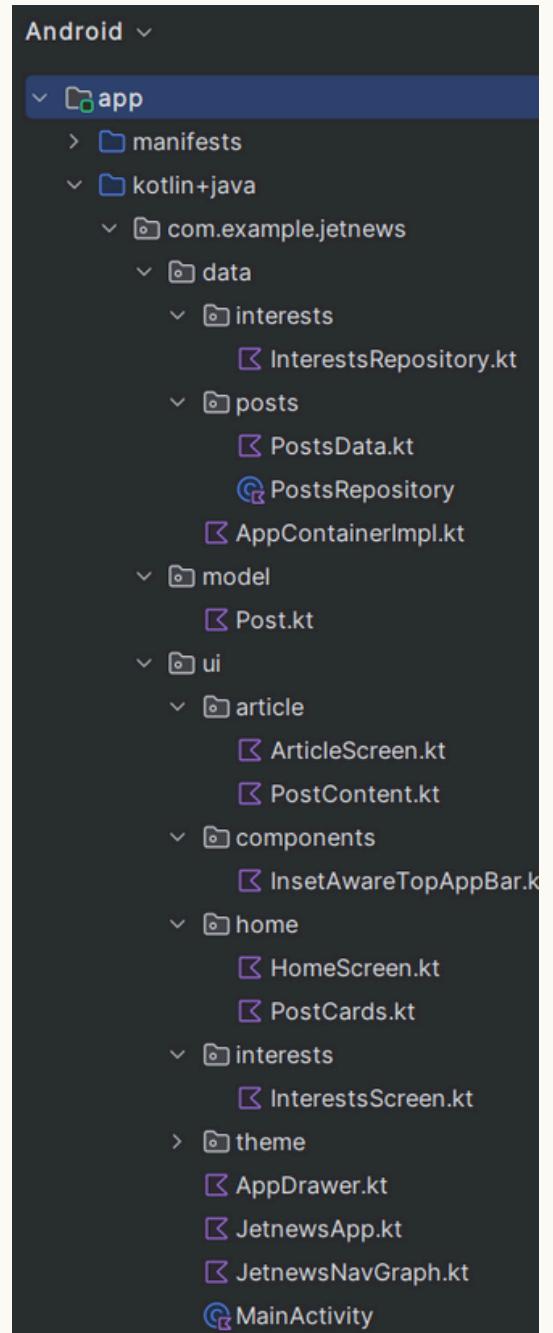
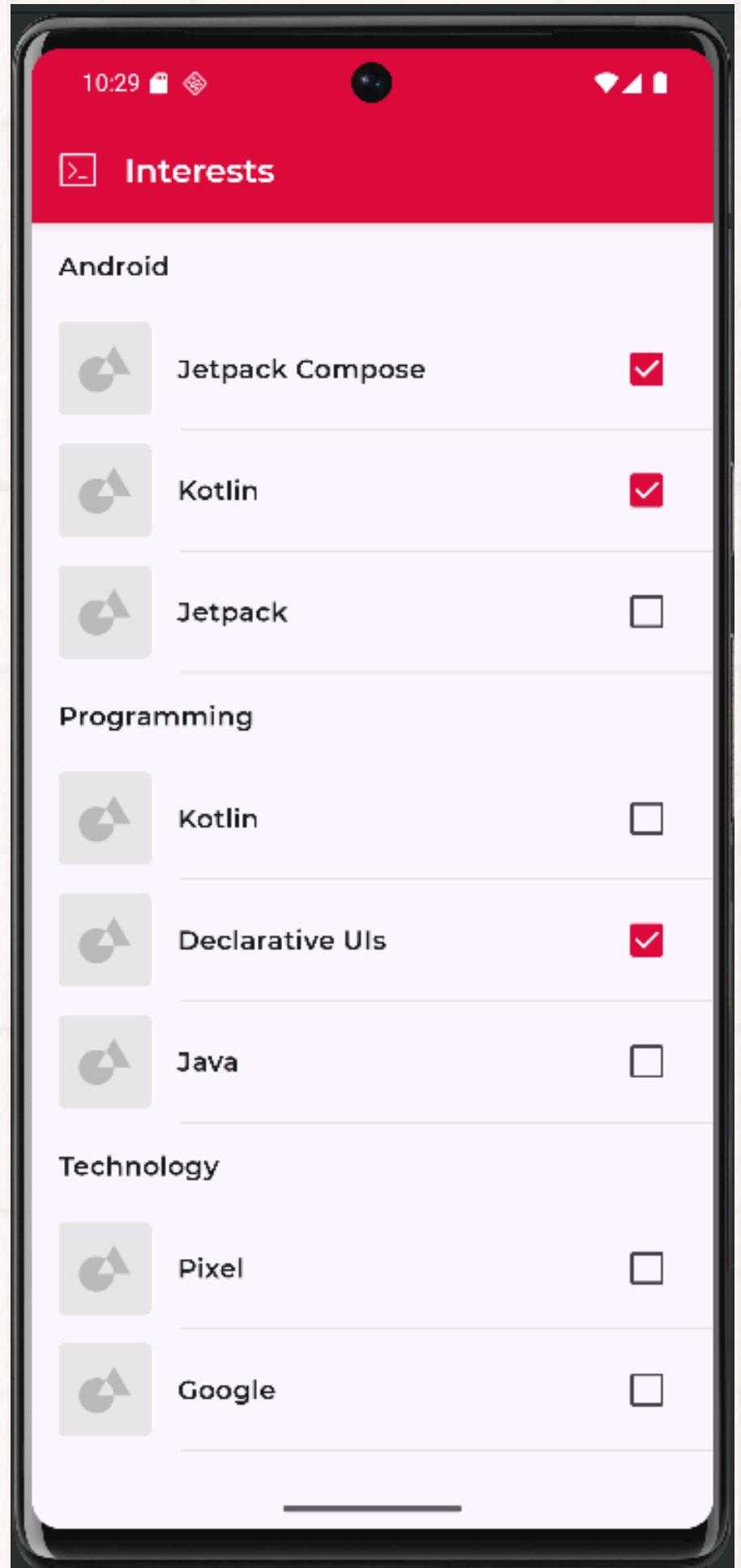
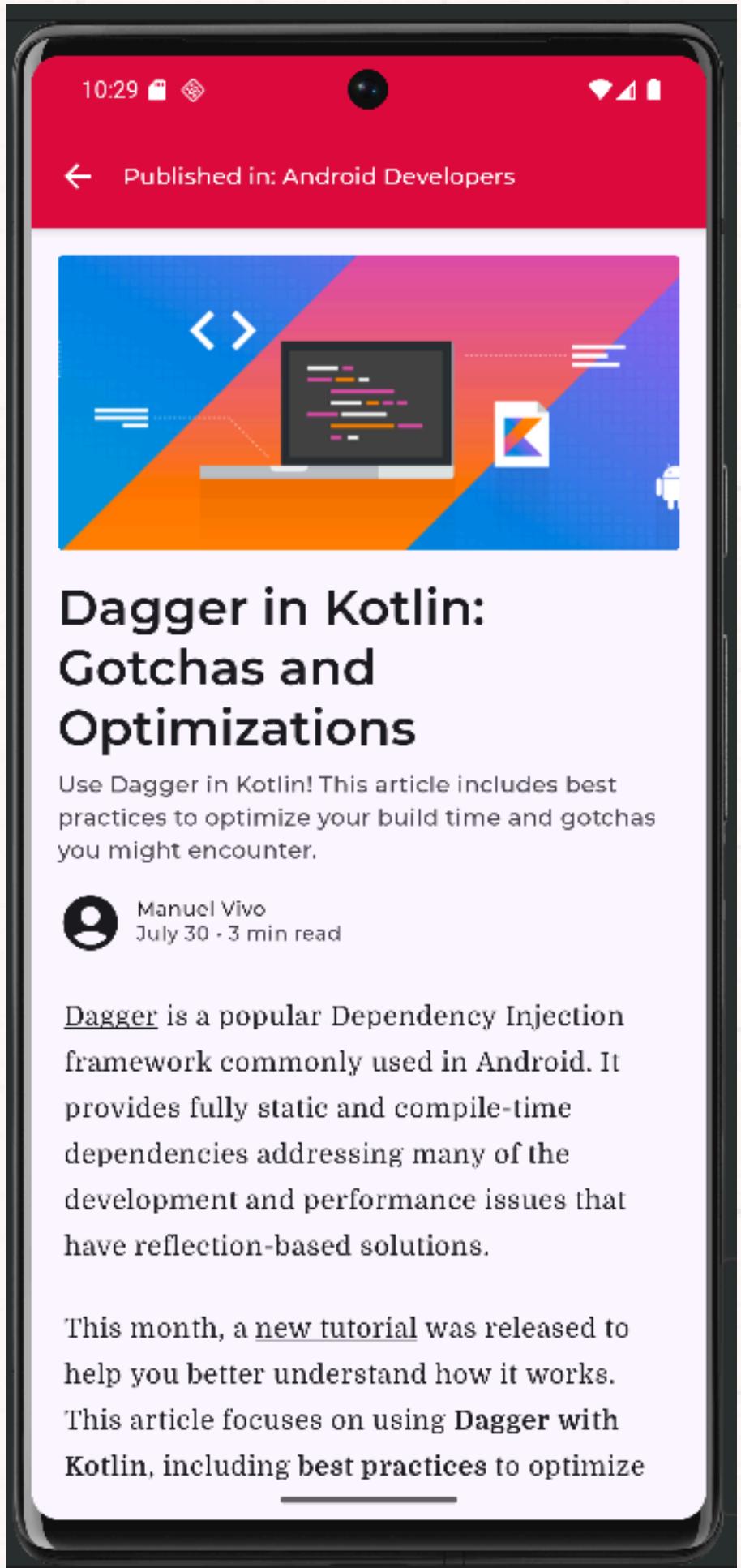
# ACCESIBILIDAD EN JETPACK COMPOSE



En este codelab, exploramos cómo diseñar aplicaciones, utilizando como ejemplo la app Jetnews (un lector de noticias). Aprendimos técnicas clave para mejorar la accesibilidad, como:

- Aumentar objetivos táctiles para usuarios con trastornos de motricidad.
- Propiedades semánticas: `contentDescription`, `clickLabel` y `stateDescription` para lectores de pantalla como TalkBack.
- Encabezados y agrupaciones lógicas para navegación estructurada.
- Acciones personalizadas y manejo de componentes interactivos (interruptores, checkboxes).
- Pruebas manuales con TalkBack y herramientas de diagnóstico.

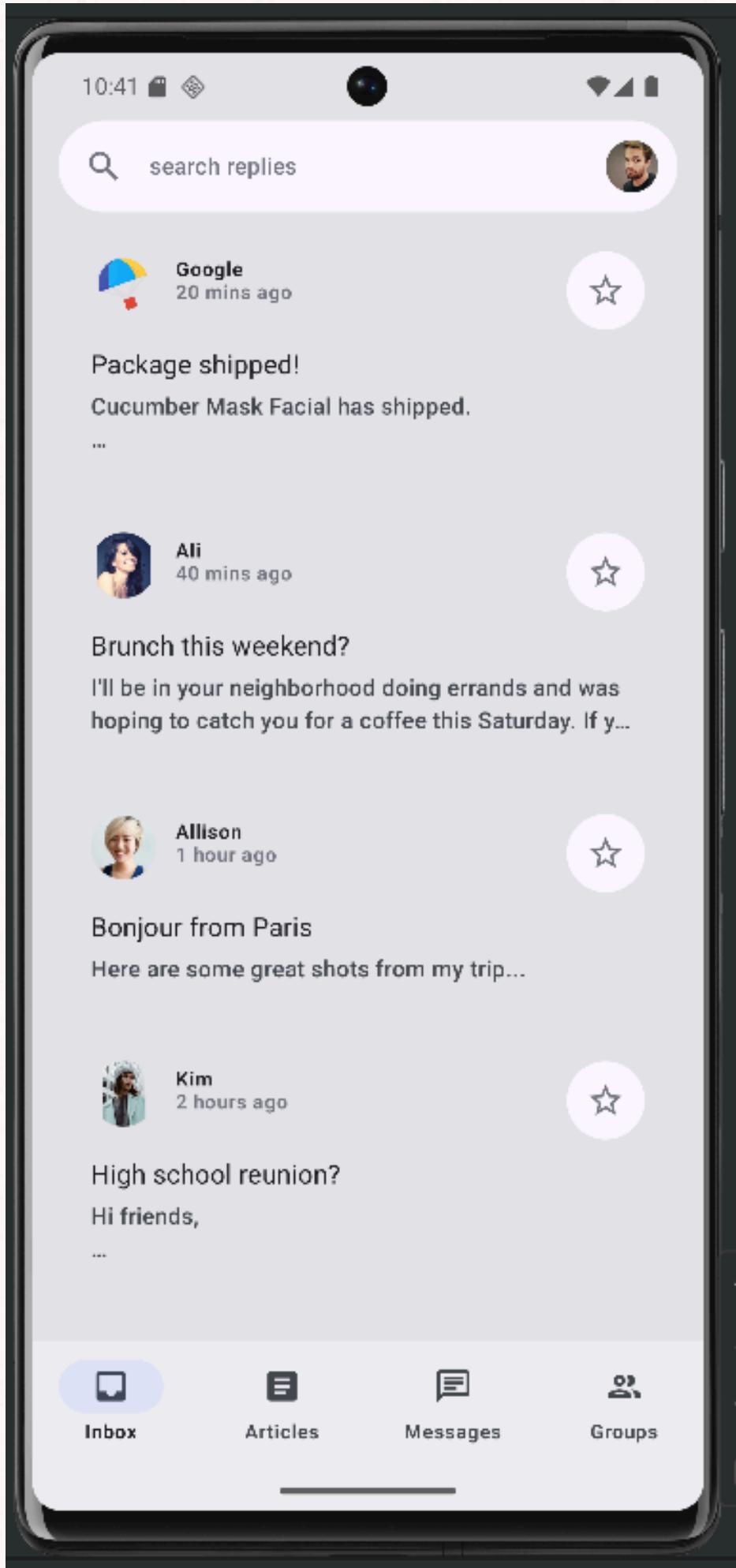




- De esta practica puedo destacar:
1. TalkBack es esencial: Las descripciones semánticas (contentDescription) permiten a usuarios con discapacidad visual entender la interfaz.
  2. Tamaños accesibles: Los componentes deben cumplir con un mínimo de 48dp x 48dp para tacto preciso.
  3. Jerarquía clara: Los encabezados (heading) y grupos semánticos facilitan la navegación.
  4. Pruebas continuas: Combinar herramientas automáticas (como Accessibility Scanner) con pruebas manuales garantiza cobertura total.

Curso de jetpack compose android

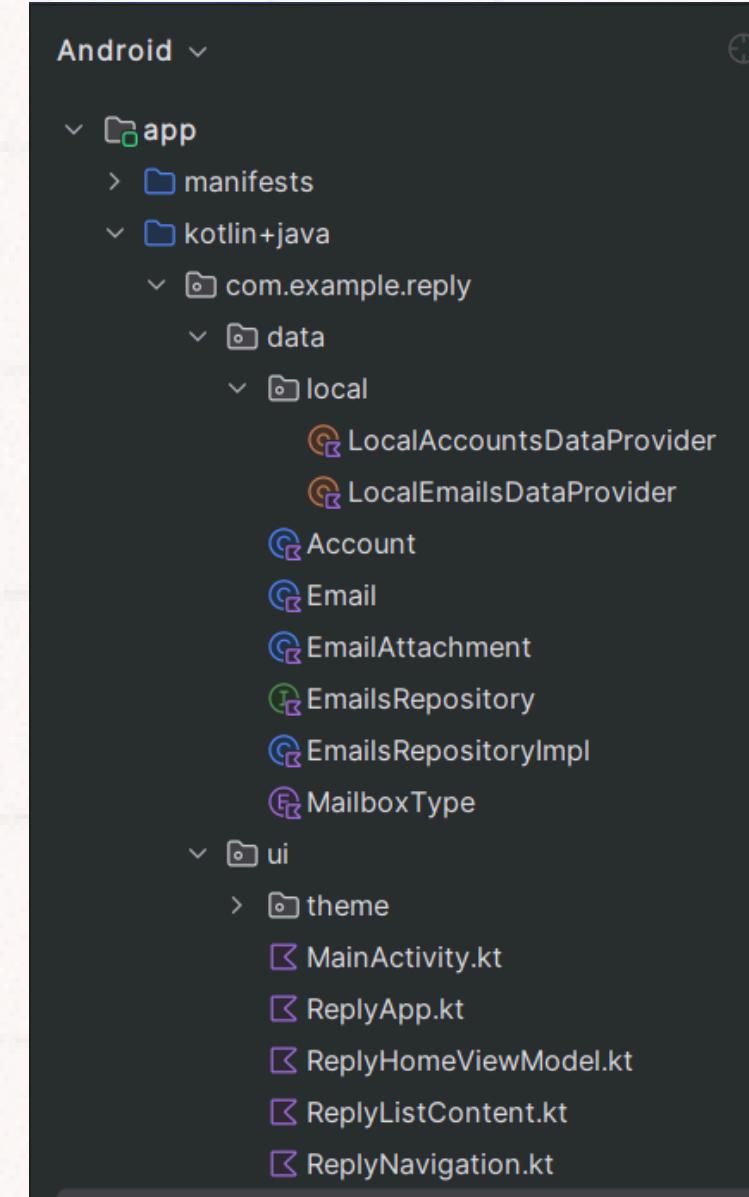
# CÓMO COMPILAR APPS ADAPTABLES CON JETPACK COMPOSE



En este codelab, exploramos cómo crear aplicaciones responsivas para múltiples dispositivos (teléfonos, tablets y plegables) usando Jetpack Compose y Material 3, centrándonos en la app Reply (cliente de correo).

- Adaptar layouts dinámicamente usando WindowSizeClass para detectar cambios de pantalla.
- Implementar patrones de navegación flexibles según el espacio disponible.
- Optimizar componentes de Material 3 para experiencias maestro-detalle.
- Integrar accesibilidad en diseños adaptables.

El objetivo fue demostrar cómo una única base de código puede ofrecer experiencias óptimas en cualquier dispositivo, desde móviles hasta pantallas grandes.

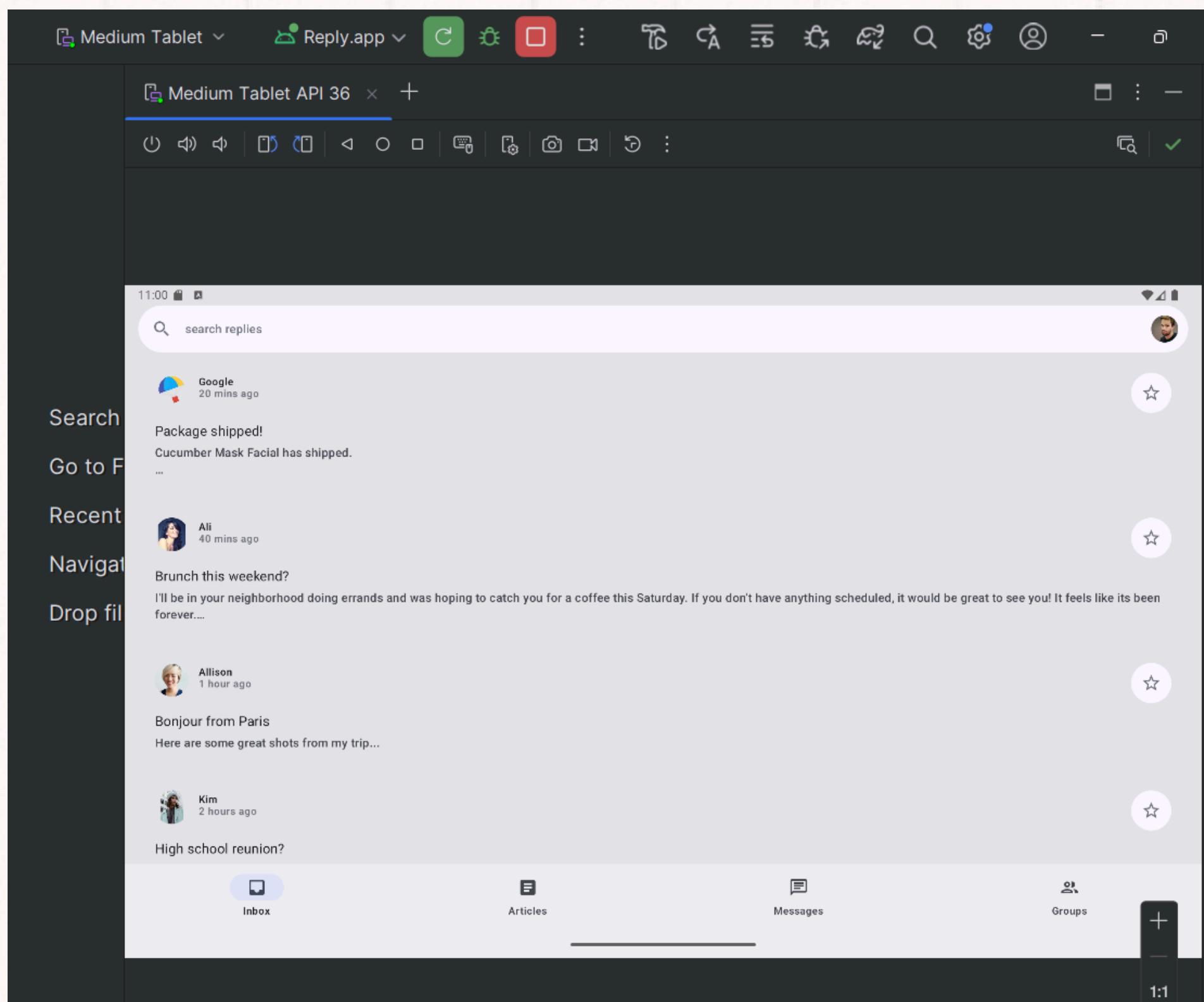


```
private fun ReplyNavigationWrapperUI(
    content: @Composable () -> Unit = {}
) {
    var selectedDestination: ReplyDestination by remember {
        mutableStateOf(ReplyDestination.Inbox)
    }

    // You will implement adaptive navigation here.
    Column(
        modifier = Modifier
            .fillMaxSize()
            .background(MaterialTheme.colorScheme.inverseOnSurface)
    ) {
        Box(modifier = Modifier.weight(1f)) {
            content()
        }

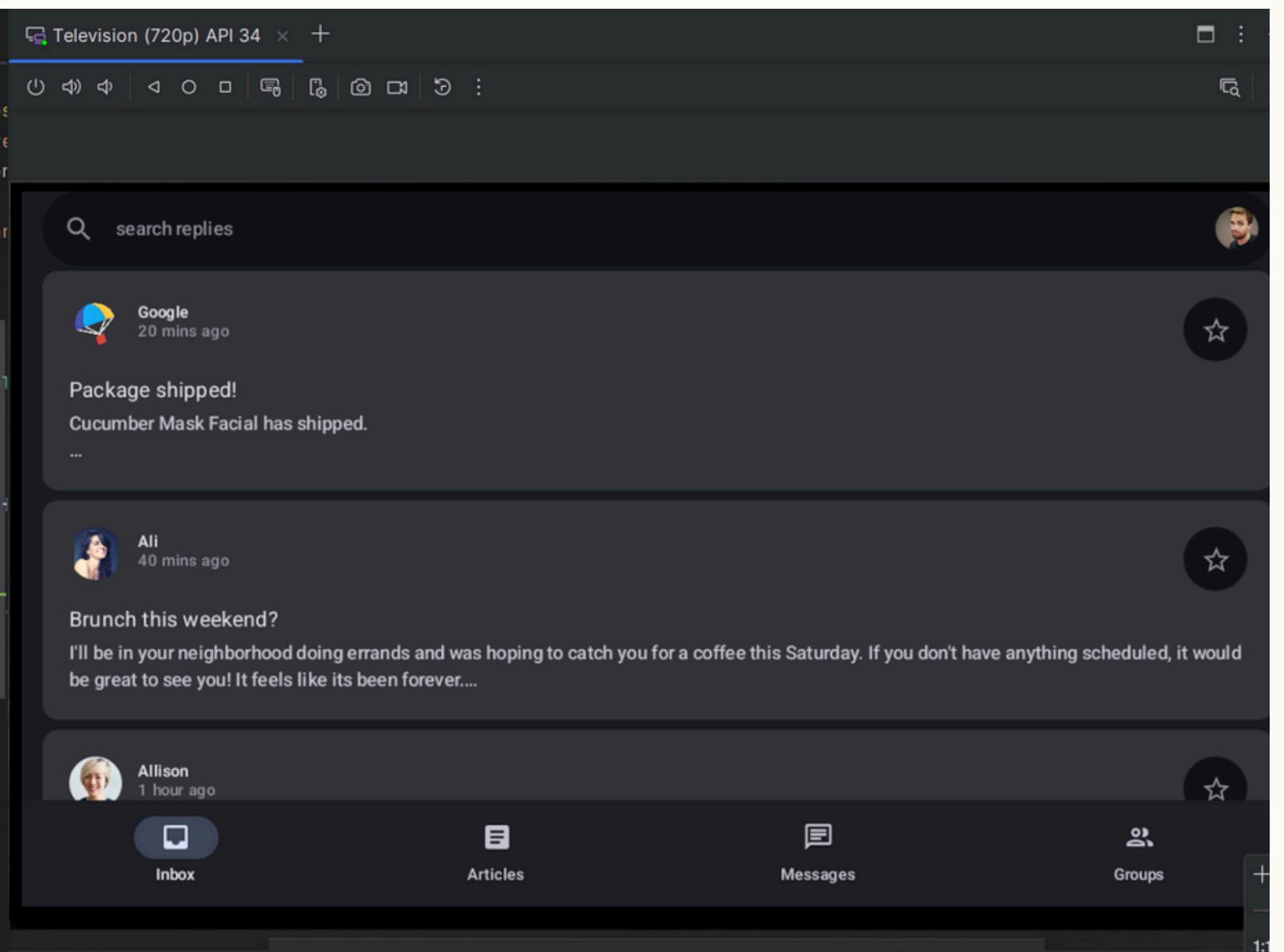
        NavigationBar(modifier = Modifier.fillMaxWidth())
            .padding(16.dp)
            .clip(RoundedCornerShape(16.dp))
            .background(MaterialTheme.colorScheme.surface)

            ReplyDestination.entries.forEach {
                NavigationBarItem(
                    selected = it == selectedDestination,
                    onClick = { /*TODO update selection*/ },
                    icon = {
                        Icon(
                            imageVector = it.icon,
                            contentDescription = stringResource(it.labelRes)
                        )
                    },
                    label = {
                        Text(it.label)
                    }
                )
            }
    }
}
```



Al implementar Reply, comprobamos que:

1. Jetpack Compose simplifica la adaptabilidad con APIs como `adaptivePaneSize` y `WindowSizeClass`.
2. Material 3 proporciona componentes listos para pantallas grandes (ej: `NavigationRail`).
3. La accesibilidad mejora naturalmente cuando los diseños son flexibles y bien estructurados.



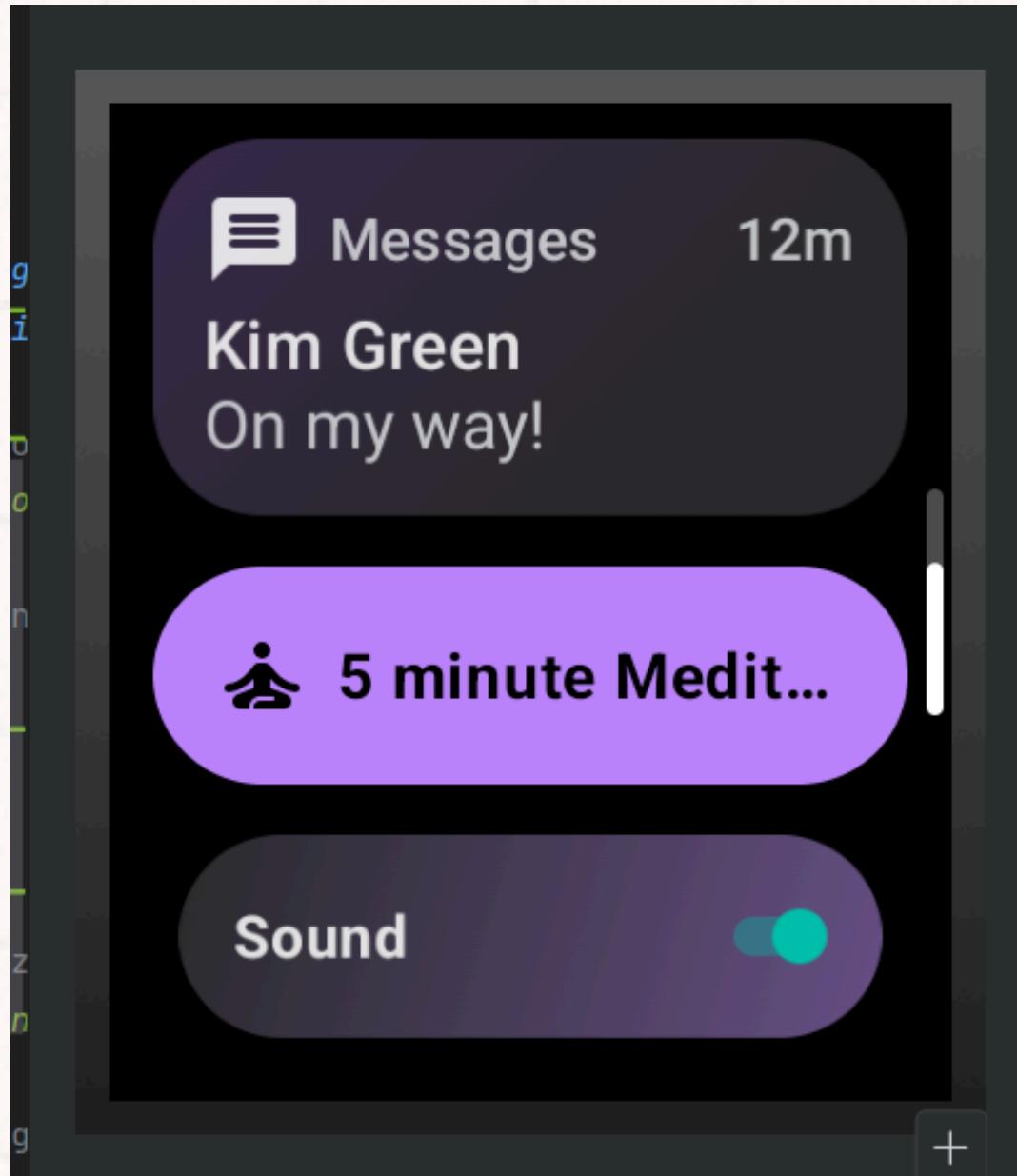
Curso de jetpack compose android

# Codelab de compose para Wear OS

The screenshot shows the Android Studio interface with the project 'ComposeForWearOSCodeLab' open. The left sidebar displays the project structure, including 'finished' and 'start' modules. The main area shows two code files: 'ReusableComponents.kt' and 'MainActivity.kt'. The code in 'ReusableComponents.kt' includes imports for 'WearApp', 'WearAppTheme', and 'AppScaffold'. It defines a 'WearApp()' function that sets the theme and scaffold. The 'AppScaffold' block contains logic for a 'ScalingLazyColumn' with specific padding and item types ('SingleButton' and 'Chip'). The code also includes comments about 'Modifiers' and 'ScreenScaffold'. The 'MainActivity.kt' file is partially visible at the bottom. To the right, a preview window shows a circular Wear OS interface with a phone icon, the text 'Square Device', and a message card for 'Kim Green' with the text 'On my way!'. The bottom status bar of the preview shows '12:04'.

En este codelab, exploramos cómo desarrollar apps para Wear OS usando Jetpack Compose, aprovechando el conocimiento previo de Compose móvil pero adaptado a las particularidades de los wearables.

- Similitudes y diferencias entre Compose para móvil y Wear OS.
- Componentes exclusivos de Wear OS como `ScalingLazyColumn` (para listas optimizadas) y `Vignette`.
- Material Design adaptado a pantallas circulares y cuadradas.
- Uso de Horologist, librería de código abierto que extiende las capacidades de Compose para wearables.



- Compose unifica el desarrollo: El mismo paradigma declarativo funciona en móviles y wearables, con componentes específicos para Wear OS.
- Experiencia de usuario crítica: En pantallas diminutas, cada pixel cuenta
- Ecosistema robusto: Librerías como Horologist aceleran el desarrollo de funciones complejas(cronómetros, gestos).