Android
Unidad 7. IU avanzadas

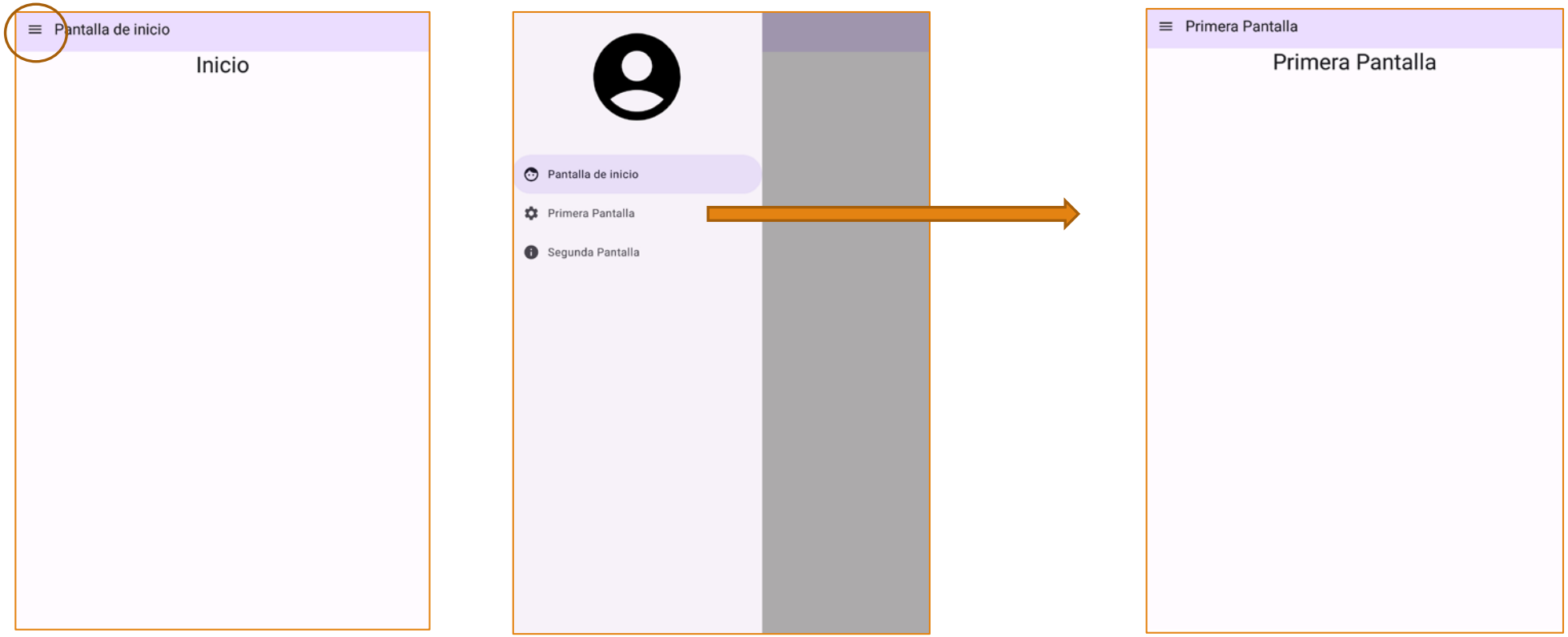
PROGRAMACIÓN MULTIMEDIA Y DE DISPOSITIVOS MÓVILES

2º DAM

Índice

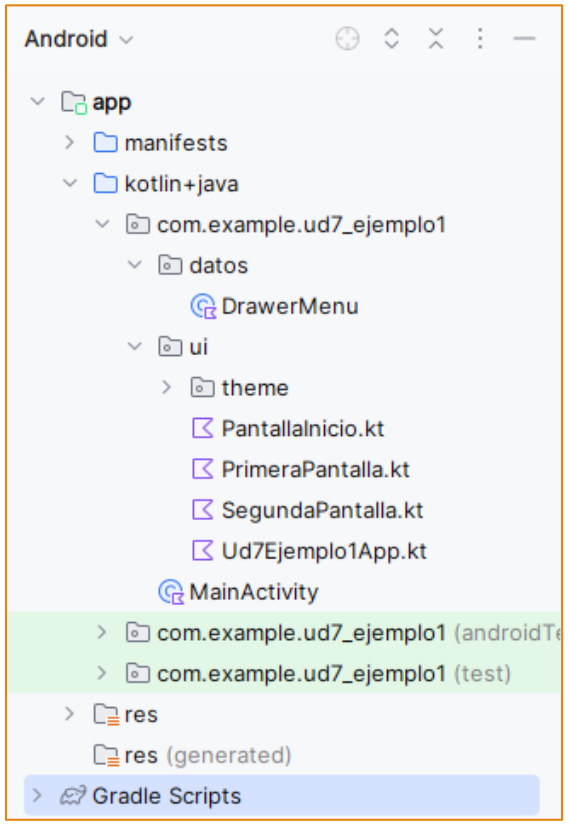
1. *NavigationDrawer*

El funcionamiento de la aplicación será el siguiente:



1. *NavigationDrawer*

La estructura del proyecto será la siguiente:



1. *NavigationDrawer*

MainActivity.kt

```kotlin
package com.example.ud7_ejemplo1

import ...

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            Ud7_Ejemplo1Theme {
                Ud7Ejemplo1App()
            }
        }
    }
}
```

DrawerMenu.kt

```kotlin
package com.example.ud7_ejemplo1.datos

import ...

data class DrawerMenu(
    val icono: ImageVector,
    @StringRes val titulo: Int,
    val ruta: String
)
```

Creamos una clase con los atributos que tendrá cada elemento del menú del *NavigationDrawer*: icono, título y ruta a la que navega.

1. *NavigationDrawer*

Ud7Ejemplo1App.kt

```kotlin
package com.example.ud7_ejemplo1.ui

import ...

enum class Pantallas(@StringRes val titulo: Int) {
    Inicio(titulo = "Pantalla de inicio"),
    Pantalla1(titulo = "Primera Pantalla"),
    Pantalla2(titulo = "Segunda Pantalla")
}

val menu = arrayOf(
    DrawerMenu(Icons.Filled.Face, Pantallas.Inicio.titulo, Pantallas.Inicio.name),
    DrawerMenu(Icons.Filled.Settings, Pantallas.Pantalla1.titulo, Pantallas.Pantalla1.name),
    DrawerMenu(Icons.Filled.Info, Pantallas.Pantalla2.titulo, Pantallas.Pantalla2.name)
)
```

Creamos un *array* con los elementos que tendrá el menú del *NavigationDrawer*.

1. *NavigationDrawer*

Ud7Ejemplo1App.kt

```
62
63  @Composable
64  fun Ud7Ejemplo1App(
65      navController: NavHostController = rememberNavController(),
66      coroutineScope: CoroutineScope = rememberCoroutineScope(),
67      drawerState: DrawerState = rememberDrawerState(initialValue = DrawerValue.Closed),
68  ){
69      val pilaRetroceso by navController.currentBackStackEntryAsState()
70
71      val pantallaActual = Pantallas.valueOf(
72          value: pilaRetroceso?.destination?.route ?: Pantallas.Inicio.name
73      )
74
75      ModalNavigationDrawer(
76          drawerState = drawerState,
77          drawerContent = {
78              ModalDrawerSheet {
79                  DrawerContent(
80                      menu = menu,
81                      pantallaActual = pantallaActual
82                  ) { ruta ->
83                      coroutineScope.launch {
84                          drawerState.close()
85                      }
86
87                      navController.navigate(ruta)
88                  }
89              }
90          },
91      ) {
```

Como argumentos de la función deberemos tener:

- *navController* para realizar la navegación
- *coroutineScope* para crear la corutina y así poder abrir y cerrar el *NavigationDrawer*.
- *drawerState* para pasarle el estado actual (abierto o cerrado) al *NavigationDrawer*.

ModelNavigationDrawer es el elemento que crea el *NavigationDrawer*. Le pasamos por parámetros *drawerState* con su estado actual y *drawerContent* con su contenido, para ello utilizamos el elemento *ModalDrawerSheet* y llamamos a nuestra función *DrawerContent*.

1. *NavigationDrawer*

```
92          Scaffold(
93              topBar = {
94                  AppTopBar(
95                      pantallaActual = pantallaActual,
96                      drawerState = drawerState
97                  )
98              }
99          ) { innerPadding ->
100
101             NavHost(
102                 navController = navController,
103                 startDestination = Pantallas.Inicio.name,
104                 modifier = Modifier.padding(innerPadding)
105             ) {
106                 // Grafo de las rutas
107                 composable(route = Pantallas.Inicio.name) {
108                     PantallaInicio(
109                         modifier = Modifier
110                             .fillMaxSize()
111                     )
112                 }
113                 composable(route = Pantallas.Pantalla1.name) {
114                     PrimeraPantalla(
115                         modifier = Modifier
116                             .fillMaxSize()
117                     )
118                 }
119                 composable(route = Pantallas.Pantalla2.name) {
120                     SegundaPantalla(
121                         modifier = Modifier
122                             .fillMaxSize()
123                     )
124                 }
125
126             }
127         }
128     }
129 }
```

1. *NavigationDrawer*

```
@Composable
private fun DrawerContent(
    menu: Array<DrawerMenu>,
    pantallaActual: Pantallas,
    onMenuClick: (String) -> Unit
) {
    Column(
        modifier = Modifier.fillMaxSize()
    ) {
        Box(
            modifier = Modifier
                .fillMaxWidth()
                .height(200.dp),
            contentAlignment = Alignment.Center
        ) {
            Image(
                modifier = Modifier.size(150.dp),
                imageVector = Icons.Filled.AccountCircle,
                contentScale = ContentScale.Crop,
                contentDescription = null
            )
        }
        Spacer(modifier = Modifier.height(12.dp))
        menu.forEach {
            NavigationDrawerItem(
                label = { Text(text = stringResource(id = it.titulo)) },
                icon = { Icon(imageVector = it.icono, contentDescription = null) },
                selected = it.titulo == pantallaActual.titulo,
                onClick = {
                    onMenuClick(it.ruta)
                }
            )
        }
    }
}
```

Por cada elemento del menú pasado por argumento a la función creamos un *NavigationDrawerItem.*

1. *NavigationDrawer*

PantallaInicio.kt

```kotlin
package com.example.ud7_ejemplo1.ui

import ...

@Composable
fun PantallaInicio(
    modifier: Modifier = Modifier
){
    Column (modifier = modifier,
        horizontalAlignment = Alignment.CenterHorizontally) {
        Text(
            text = stringResource(R.string.inicio),
            style = MaterialTheme.typography.headlineLarge
        )
    }
}
```

PrimeraPantalla.kt

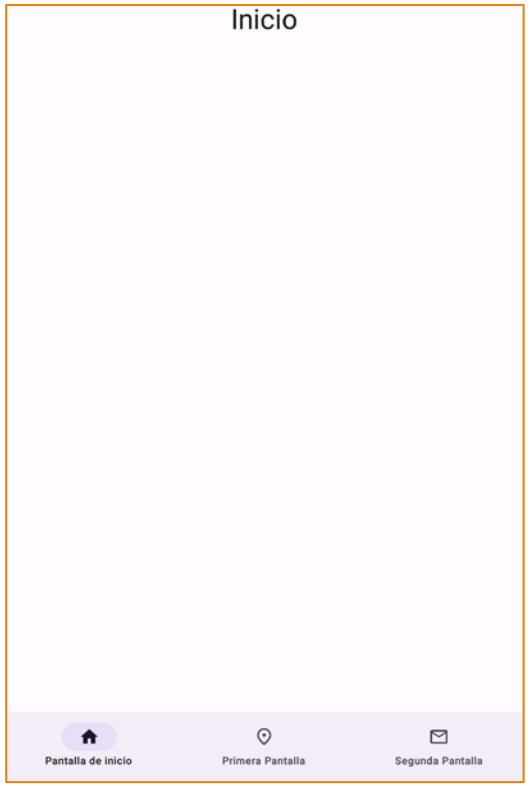
```kotlin
package com.example.ud7_ejemplo1.ui

import ...

@Composable
fun PrimeraPantalla(
    modifier: Modifier = Modifier
){
    Column (modifier = modifier,
        horizontalAlignment = Alignment.CenterHorizontally) {
        Text(
            text = stringResource(id = "Primera Pantalla"),
            style = MaterialTheme.typography.headlineLarge
        )
    }
}
```

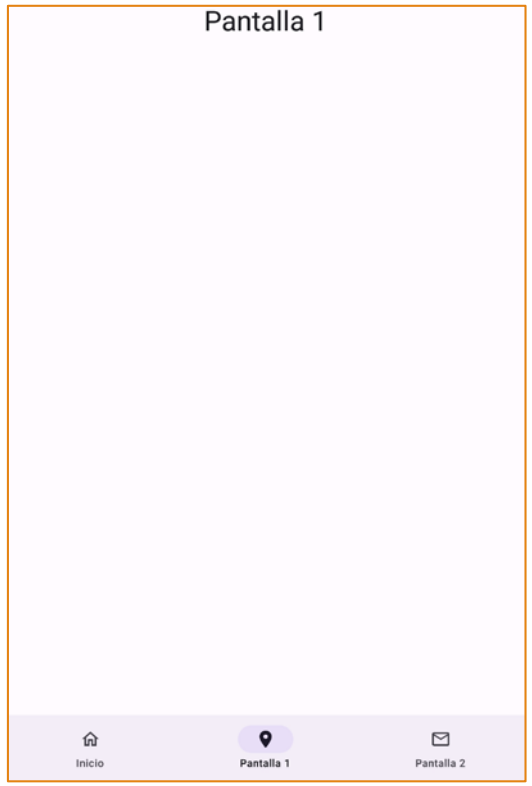
SegundaPantalla.kt

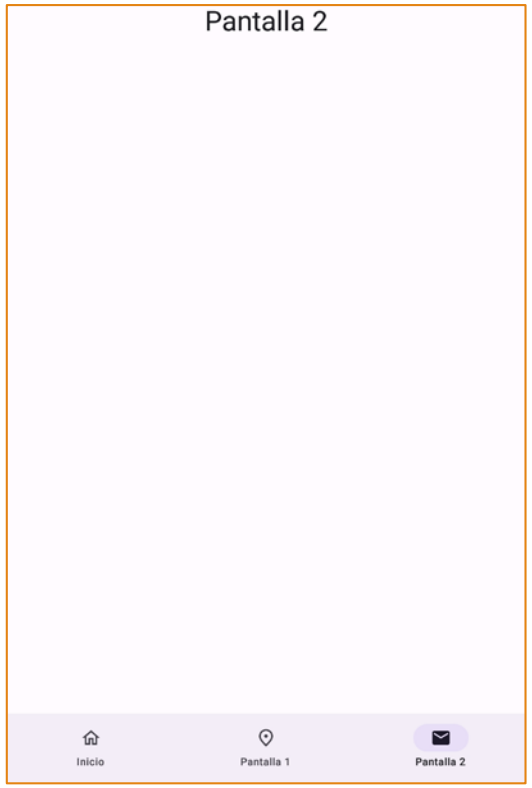
```kotlin
package com.example.ud7_ejemplo1.ui

import ...

@Composable
fun SegundaPantalla(
    modifier: Modifier = Modifier
){
    Column (modifier = modifier,
        horizontalAlignment = Alignment.CenterHorizontally) {
        Text(
            text = stringResource(R.string.segunda_pantalla),
            style = MaterialTheme.typography.headlineLarge
        )
    }
}
```

2. *Navigation bar*

El funcionamiento de la aplicación será el siguiente:

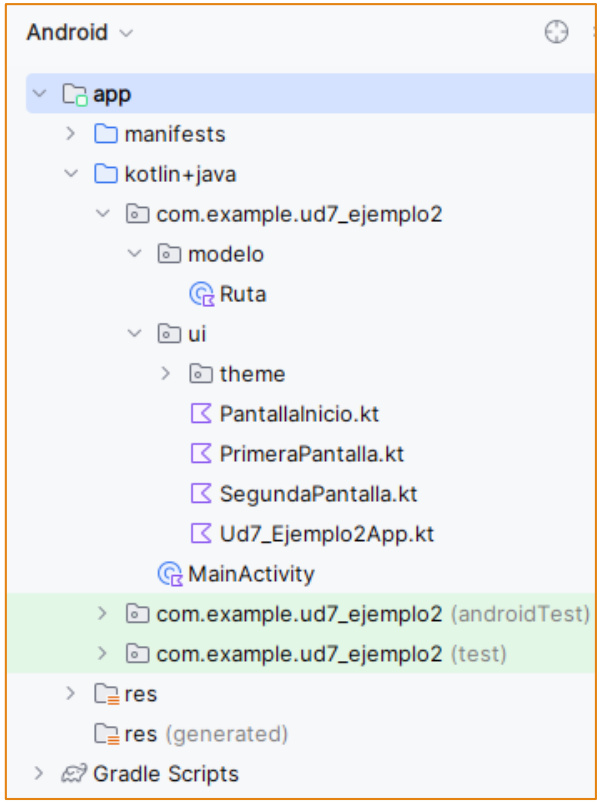






2. *Navigation bar*

La estructura del proyecto será la siguiente:



2. *Navigation bar*

MainActivity.kt

```kotlin
package com.example.ud7_ejemplo2

import ...

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            Ud7_Ejemplo2Theme {
                Ud7_Ejemplo2App()
            }
        }
    }
}
```

Ruta.kt

```kotlin
package com.example.ud7_ejemplo2.modelo

import ...

data class Ruta<T : Any>(
    @StringRes val nombre: Int,
    val ruta: T,
    val iconoLleno: ImageVector,
    val iconoVacio: ImageVector
)
```

Creamos una clase con los atributos que tendrá cada elemento del *Navigation bar*: nombre, ruta a la que navega y los iconos (lleno si está seleccionado y vacío si no lo está).

2. *Navigation bar*

Ud7_Ejemplo2App.kt

```kotlin
package com.example.ud7_ejemplo2.ui

import ...

enum class Pantallas(@StringRes val titulo: Int) {
    Inicio(titulo = R.string.pantalla_inicio),
    Pantalla1(titulo = R.string.pantalla1),
    Pantalla2(titulo = R.string.pantalla2)
}

val listaRutas = listOf(
    Ruta(Pantallas.Inicio.titulo, Pantallas.Inicio.name, Icons.Filled.Home, Icons.Outlined.Home),
    Ruta(Pantallas.Pantalla1.titulo, Pantallas.Pantalla1.name,  Icons.Filled.Place, Icons.Outlined.Place),
    Ruta(Pantallas.Pantalla2.titulo, Pantallas.Pantalla2.name, Icons.Filled.Email, Icons.Outlined.Email)
)
```

Creamos un *array* con los elementos que tendrá el *Navigation bar*.

2. *Navigation bar*

```
@Composable
fun Ud7_Ejemplo2App(
    navController: NavHostController = rememberNavController()
){
    var selectedItem by remember { mutableIntStateOf( value: 0) }

    Scaffold(
        bottomBar = {
            NavigationBar {
                listaRutas.forEachIndexed { indice, ruta ->
                    NavigationBarItem(
                        icon = {
                            if(selectedItem == indice)
                                Icon(
                                    imageVector = ruta.iconoLleno,
                                    contentDescription = stringResource(id = ruta.nombre)
                                )
                            else
                                Icon(
                                    imageVector = ruta.iconoVacio,
                                    contentDescription = stringResource(id = ruta.nombre)
                                )
                        },
                        label = { Text(stringResource(id = ruta.nombre)) },
                        selected = selectedItem == indice,
                        onClick = {
                            selectedItem = indice
                            navController.navigate(ruta.ruta)
                        }
                    )
                }
            }
        },
        modifier = Modifier.fillMaxSize()
```

Hacemos uso del argumento *bottomBar* del elemento *Scaffold*. En él utilizamos un elemento *NavigationBar* donde recorremos la lista de rutas mediante un *forEachIndexed* para no solo tener la ruta sino también su índice y para cada elemento dibujamos un *NavigationBarItem*.

2. *Navigation bar*

```
78      ) { innerPadding ->
79          NavHost(
80              navController = navController,
81              startDestination = Pantallas.Inicio.name,
82              modifier = Modifier.padding(innerPadding)
83          ) {
84              // Grafo de las rutas
85              composable(route = Pantallas.Inicio.name) {
86                  PantallaInicio(
87                      modifier = Modifier
88                          .fillMaxSize()
89                  )
90              }
91              composable(route = Pantallas.Pantalla1.name) {
92                  PrimeraPantalla(
93                      modifier = Modifier
94                          .fillMaxSize()
95                  )
96              }
97               composable(route = Pantallas.Pantalla2.name) {
98                      SegundaPantalla(
99                          modifier = Modifier
100                             .fillMaxSize()
101                     )
102              }
103
104         }
105     }
106 }
```

2. *Navigation bar*

PantallaInicio.kt

```kotlin
package com.example.ud7_ejemplo2.ui

import ...

@Composable
fun PantallaInicio(
    modifier: Modifier = Modifier
){
    Column (modifier = modifier,
        horizontalAlignment = Alignment.CenterHorizontally) {
        Text(
            text = stringResource(R.string.pantalla_inicio),
            style = MaterialTheme.typography.headlineLarge
        )
    }
}
```

PrimeraPantalla.kt

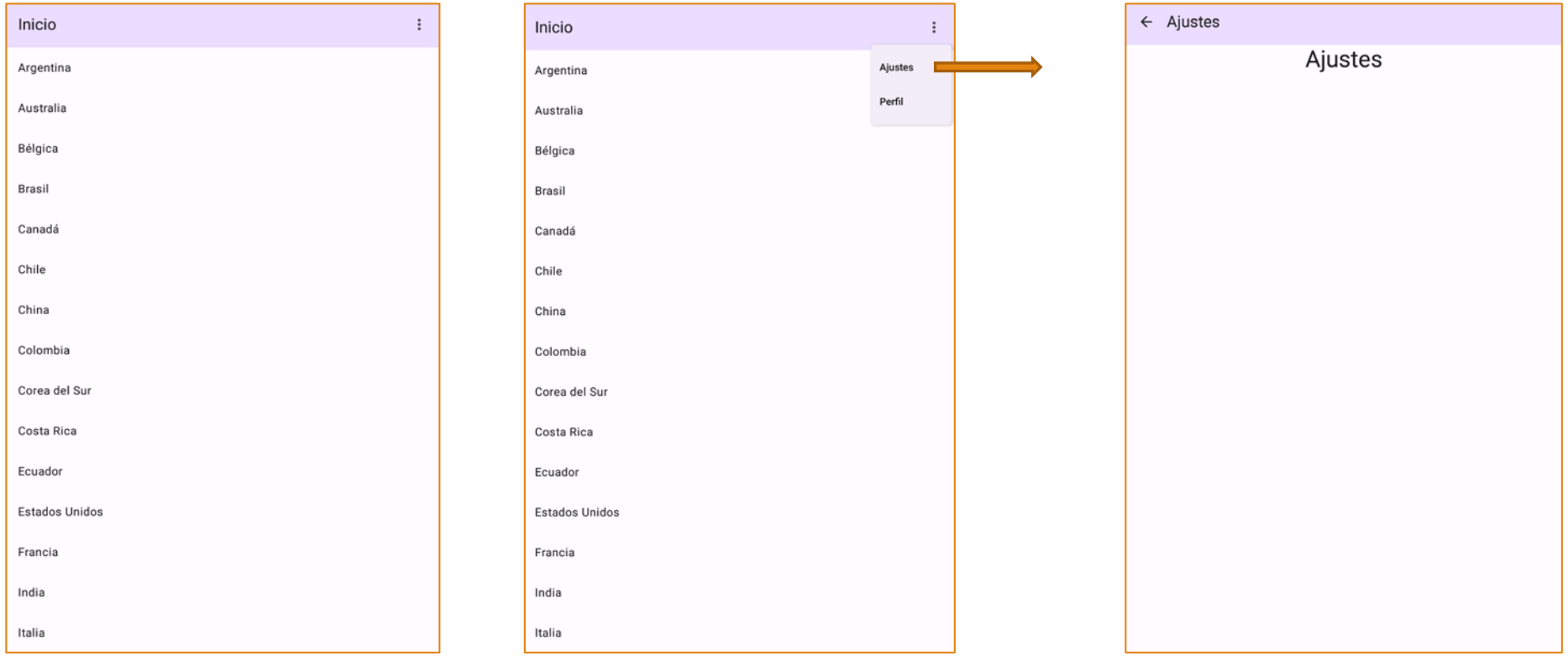
```kotlin
package com.example.ud7_ejemplo2.ui

import ...

@Composable
fun PrimeraPantalla(
    modifier: Modifier = Modifier
){
    Column (modifier = modifier,
        horizontalAlignment = Alignment.CenterHorizontally) {
        Text(
            text = stringResource(id = R.string.pantalla1),
            style = MaterialTheme.typography.headlineLarge
        )
    }
}
```

SegundaPantalla.kt

```kotlin
package com.example.ud7_ejemplo2.ui

import ...

@Composable
fun SegundaPantalla(
    modifier: Modifier = Modifier
){
    Column (modifier = modifier,
        horizontalAlignment = Alignment.CenterHorizontally) {
        Text(
            text = stringResource(R.string.pantalla2),
            style = MaterialTheme.typography.headlineLarge
        )
    }
}
```

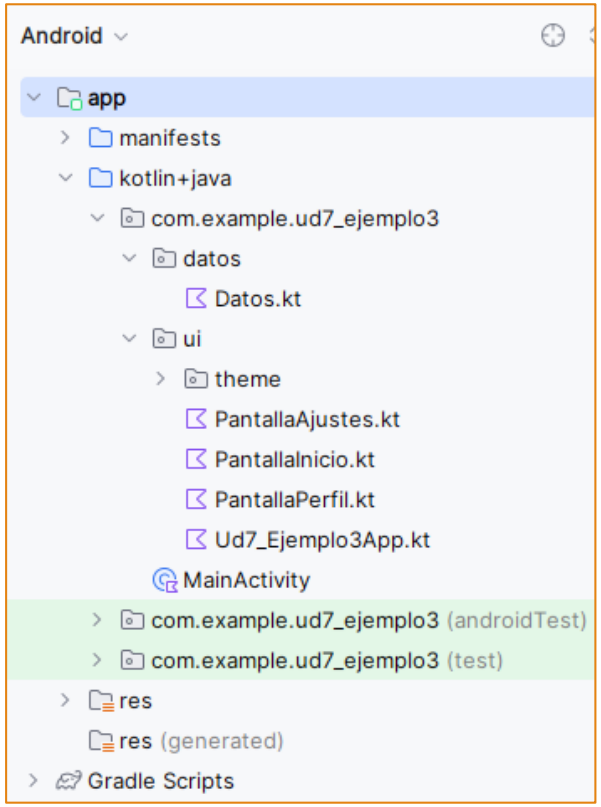
3. Menú

El funcionamiento de la aplicación será el siguiente:



3. Menú

La estructura del proyecto será la siguiente:



3. Menú

MainActivity.kt

```kotlin
package com.example.ud7_ejemplo3

import ...

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            Ud7_Ejemplo3Theme {
                Ud7_Ejemplo3App()
            }
        }
    }
}
```

Datos.kt

```kotlin
package com.example.ud7_ejemplo3.datos

val lista: List<String> = listOf(
    "Argentina",
    "Australia",
    "Bélgica",
    "Brasil",
    "Canadá",
    "Chile",
    "China",
    "Colombia",
    "Corea del Sur",
    "Costa Rica",
    "Ecuador",
    "Estados Unidos",
    "Francia",
    "India",
    "Italia",
    "Jamaica",
    "Japón",
    "Polonia",
    "Portugal",
    "Reino Unido",
    "Rusia",
    "Sudáfrica",
    "Suiza"
)
```

3. Menú

Ud7_Ejemplo3App.kt

```kotlin
package com.example.ud7_ejemplo3.ui

import ...

enum class Pantallas(@StringRes val titulo: Int) {
    Inicio(titulo = R.string.inicio),
    Ajustes(titulo = R.string.ajustes),
    Perfil(titulo = R.string.perfil)
}

@OptIn(ExperimentalMaterial3Api::class)
@Composable
fun Ud7_Ejemplo3App(
    navController: NavHostController = rememberNavController()
){
    val pilaRetroceso by navController.currentBackStackEntryAsState()

    val pantallaActual = Pantallas.valueOf(
        value: pilaRetroceso?.destination?.route ?: Pantallas.Inicio.name
    )

    val scrollBehavior = TopAppBarDefaults.enterAlwaysScrollBehavior(rememberTopAppBarState())

    Scaffold(
        topBar = {
            AppTopBar(
                pantallaActual = pantallaActual,
                navController = navController,
                scrollBehavior = scrollBehavior
            )
        },
        modifier = Modifier.nestedScroll(scrollBehavior.nestedScrollConnection)
    ) { innerPadding ->
```

Creamos una variable *scrollBehavior* para controlar el comportamiento del *topBar* cuando hay un desplazamiento por la pantalla. En este caso mediante la función *enterAlwaysScrollBehavior* conseguimos que el *topBar* desaparezca cuando deslizamos hacía abajo la lista de elementos.

En este ejemplo pasamos por argumento de la función *AppTopBar* directamente la variable *navController* ya que será ella la que gestione todo el funcionamiento de la *TopBar*.

3. Menú

```
66          NavHost(
67              navController = navController,
68              startDestination = Pantallas.Inicio.name,
69              modifier = Modifier.padding(innerPadding)
70          ){
71              // Grafo de las rutas
72              composable(route = Pantallas.Inicio.name) {
73                  PantallaInicio(
74                      modifier = Modifier
75                          .fillMaxSize()
76                  )
77              }
78              composable(route = Pantallas.Ajustes.name) {
79                  PantallaAjustes(
80                      modifier = Modifier
81                          .fillMaxSize()
82                  )
83              }
84              composable(route = Pantallas.Perfil.name) {
85                  PantallaPerfil(
86                      modifier = Modifier
87                          .fillMaxSize()
88                  )
89              }
90          }
91      }
92  }
```

3. Menú

```
94  @OptIn(ExperimentalMaterial3Api::class)
95  @Composable
96  fun AppTopBar(
97      pantallaActual: Pantallas,
98      navController: NavHostController,
99      scrollBehavior: TopAppBarScrollBehavior,
100     modifier: Modifier = Modifier
101 ){
102     var mostrarMenu by remember { mutableStateOf( value: false) }
103
104     TopAppBar(
105         title = { Text(text = stringResource(id = pantallaActual.titulo)) },
106         colors = TopAppBarDefaults.mediumTopAppBarColors(
107             containerColor = MaterialTheme.colorScheme.primaryContainer
108         ),
109         navigationIcon = {
110             if(navController.previousBackStackEntry != null) {
111                 IconButton(onClick = { navController.navigateUp() }) {
112                     Icon(
113                         imageVector = Icons.AutoMirrored.Filled.ArrowBack,
114                         contentDescription = stringResource("Atrás")
115                     )
116                 }
117             }
118         },
```

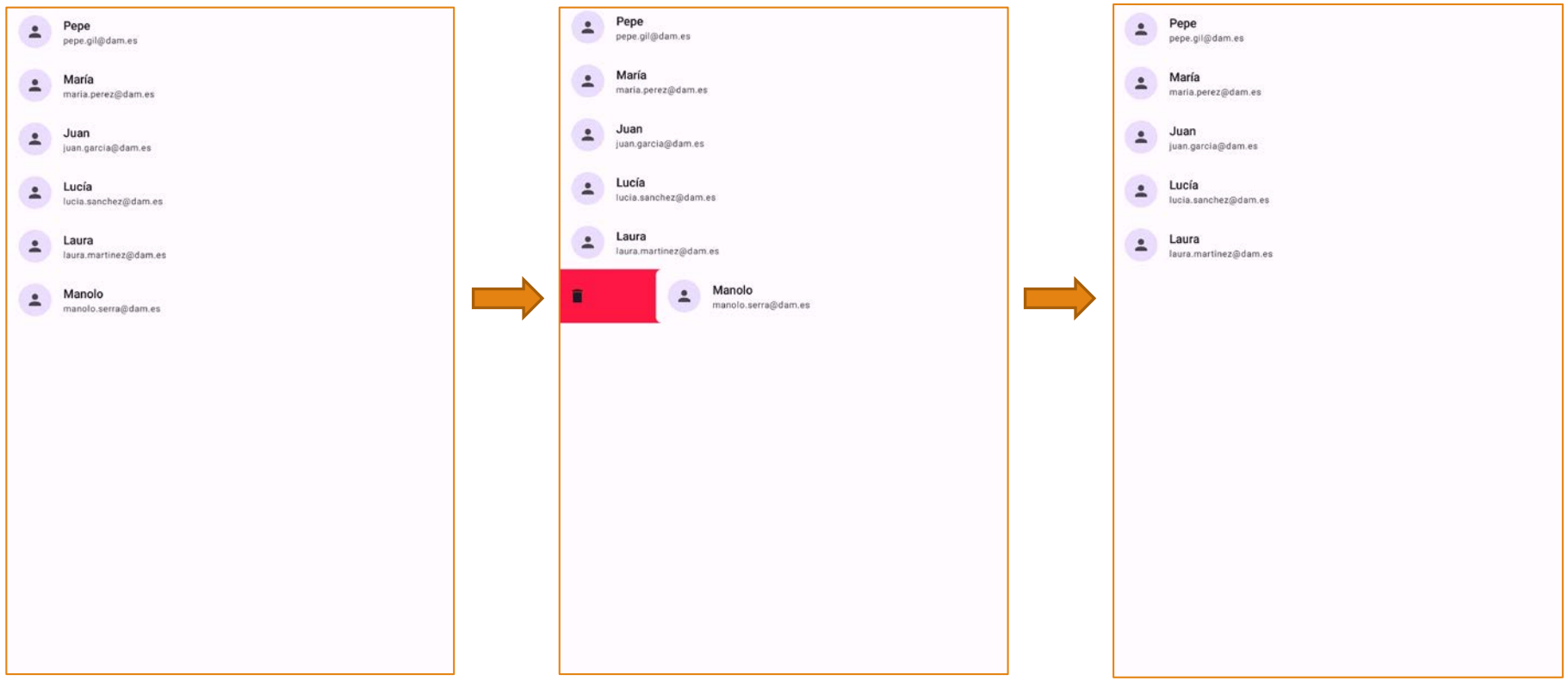
3. Menú

```
119         actions = {
120             if(pantallaActual == Pantallas.Inicio) {
121                 IconButton(onClick = { mostrarMenu = true }) {
122                     Icon(
123                         imageVector = Icons.Outlined.MoreVert,
124                         contentDescription = stringResource("Abrir menú")
125                     )
126                 }
127                 DropdownMenu(
128                     mostrarMenu, { mostrarMenu = false }
129                 ) {
130                     DropdownMenuItem(
131                         text = { Text(text = stringResource(id = "Ajustes")) },
132                         onClick = {
133                             mostrarMenu = false
134                             navController.navigate(Pantallas.Ajustes.name)
135                         }
136                     )
137                     DropdownMenuItem(
138                         text = { Text(text = stringResource(id = "Perfil")) },
139                         onClick = {
140                             mostrarMenu = false
141                             navController.navigate(Pantallas.Perfil.name)
142                         }
143                     )
144                 }
145             }
146         },
147         scrollBehavior = scrollBehavior,
148         modifier = modifier
149     )
150 }
```

Utilizamos el argumento *actions* del elemento *TopAppBar* para dibujar el menú mediante el elemento *DropdownMenu*. Para cada elemento del menú deberemos utilizar un elemento *DropdownMenuItem*.
El menú solo se mostrará cuando estemos en la pantalla de Inicio.

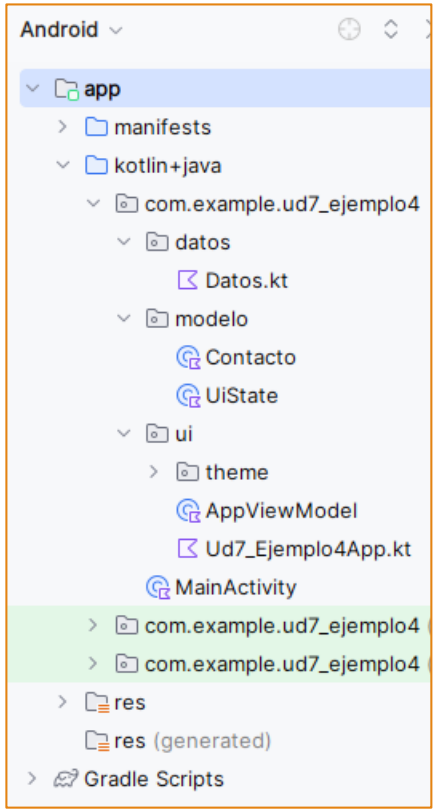
4. *SwipeToDismissBox*

El funcionamiento de la aplicación será el siguiente:



4. *SwipeToDismissBox*

La estructura del proyecto será la siguiente:



4. *SwipeToDismissBox*

MainActivity.kt

```kotlin
package com.example.ud7_ejemplo4

import ...

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            Ud7_Ejemplo4Theme {
                Ud7_Ejemplo4App()
            }
        }
    }
}
```

Datos.kt

```kotlin
package com.example.ud7_ejemplo4.datos

import com.example.ud7_ejemplo4.modelo.Contacto

val listaInicial: List<Contacto> = listOf(
    Contacto( nombre: "Pepe",  correo: "pepe.gil@dam.es"),
    Contacto( nombre: "María",  correo: "maria.perez@dam.es"),
    Contacto( nombre: "Juan",  correo: "juan.garcia@dam.es"),
    Contacto( nombre: "Lucía",  correo: "lucia.sanchez@dam.es"),
    Contacto( nombre: "Laura",  correo: "laura.martinez@dam.es"),
    Contacto( nombre: "Manolo",  correo: "manolo.serra@dam.es")
)
```

Contacto.kt

```kotlin
package com.example.ud7_ejemplo4.modelo

data class Contacto (
    val nombre: String = "",
    val correo: String = ""
)
```

4. *SwipeToDismissBox*

UiState.kt

```kotlin
package com.example.ud7_ejemplo4.modelo

data class UiState(
    val listaContactos: List<Contacto> = listOf()
)
```

AppViewModel.kt

```kotlin
package com.example.ud7_ejemplo4.ui

import ...

class AppViewModel: ViewModel() {
    private val _uiState = MutableStateFlow(UiState())
    val uiState: StateFlow<UiState> = _uiState.asStateFlow()

    init {
        _uiState.update { estado ->
            estado.copy(
                listaContactos = listaInicial.toMutableList()
            )
        }
    }

    fun eliminarContacto (contacto: Contacto) {
        _uiState.update { estado ->
            val listaActualizada = estado.listaContactos.toMutableList()
            listaActualizada.remove(contacto)

            estado.copy(
                listaContactos = listaActualizada
            )
        }
    }
}
```

4. *SwipeToDismissBox*

Ud7_Ejemplo4App.kt

```kotlin
package com.example.ud7_ejemplo4.ui

import ...

@Composable
fun Ud7_Ejemplo4App(
    appViewModel: AppViewModel = viewModel()
){
    val uiState by appViewModel.uiState.collectAsState()

    Scaffold(
        modifier = Modifier.fillMaxSize()
    ) { innerPadding ->

        LazyColumn(
            modifier = Modifier
                .padding(innerPadding)
                .fillMaxSize()
        ) {
            itemsIndexed(
                items = uiState.listaContactos,
                key = { _, item -> item.hashCode() }
            ) { _, contacto ->
                ContactoItem(
                    contacto = contacto,
                    onEliminar = { appViewModel.eliminarContacto(contacto) })
            }
        }
    }
}
```

Hacemos uso de la función *itemsIndexed* para controlar además del elemento el índice asociado.

4. *SwipeToDismissBox*

```
70  @OptIn(ExperimentalMaterial3Api::class)
71  @Composable
72  fun ContactoItem(
73      contacto: Contacto,
74      onEliminar: (Contacto) -> Unit,
75      modifier: Modifier = Modifier
76  ){
77      var puedeDismiss by remember {mutableStateOf( value: false)}
78
79      val context = LocalContext.current
80      val contatoActual by rememberUpdatedState(contacto)
81
82      val dismissState = rememberSwipeToDismissBoxState(
83          confirmValueChange = {
84              when(it) {
85                  SwipeToDismissBoxValue.StartToEnd -> {
86                      onEliminar(contatoActual)
87                      Toast.makeText(context, text: "Elemento eliminado.", Toast.LENGTH_SHORT).show()
88                      puedeDismiss = true
89                  }
90                  SwipeToDismissBoxValue.EndToStart -> { puedeDismiss = false }
91                  SwipeToDismissBoxValue.Settled -> { puedeDismiss = false }
92              }
93              return@rememberSwipeToDismissBoxState puedeDismiss
94          },
95          positionalThreshold = { it * .25f }
96      )
97      SwipeToDismissBox(
98          state = dismissState,
99          modifier = modifier,
100         backgroundContent = { DismissBackground(dismissState) },
101         content = { TarjetaContacto(contacto) }
102     )
103 }
```

Para utilizar el elemento *SwipeToDismissBox* debemos añadir la anotación *@OptIn*.

En el atributo de *confirmValueChange* tenemos que indicar mediante un booleano si se ejecuta la acción o no para cada movimiento.
Devolvemos el valor mediante una etiqueta para indicar que el *return* pertenece a la función *rememberSwipeToDismissBoxState*.

4. *SwipeToDismissBox*

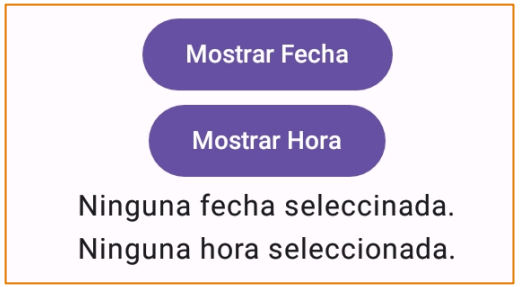
```
107  @OptIn(ExperimentalMaterial3Api::class)
108  @Composable
109  fun DismissBackground(dismissState: SwipeToDismissBoxState) {
110      val color = when (dismissState.dismissDirection) {
111          SwipeToDismissBoxValue.StartToEnd -> Color( color: 0xFFFF1744)
112          SwipeToDismissBoxValue.EndToStart -> Color( color: 0xFF1DE9B6)
113          SwipeToDismissBoxValue.Settled -> Color.Transparent
114      }
115
116      Row(
117          modifier = Modifier
118              .fillMaxSize()
119              .background(color)
120              .padding(12.dp, 8.dp),
121          verticalAlignment = Alignment.CenterVertically,
122          horizontalArrangement = Arrangement.SpaceBetween
123      ) {
124          Icon(
125              imageVector = Icons.Default.Delete,
126              contentDescription = stringResource(R.string.eliminar)
127          )
128          Spacer(modifier = Modifier)
129          Icon(
130              imageVector = Icons.Default.Delete,
131              contentDescription = stringResource(R.string.eliminar)
132          )
133      }
134  }
```

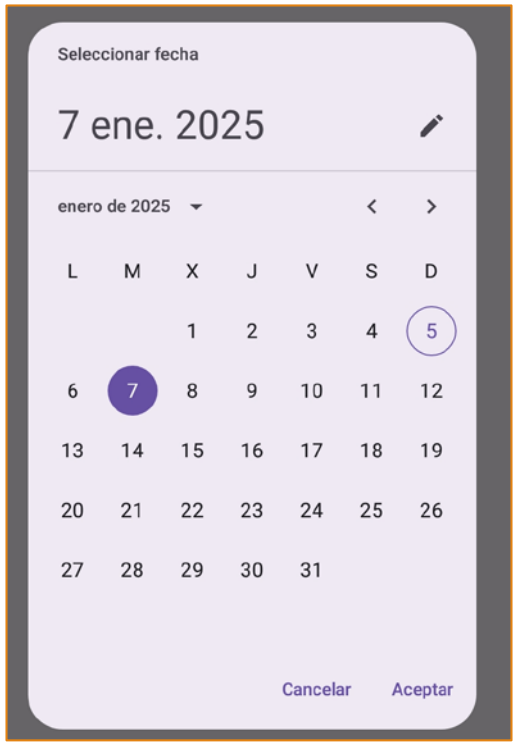
4. *SwipeToDismissBox*

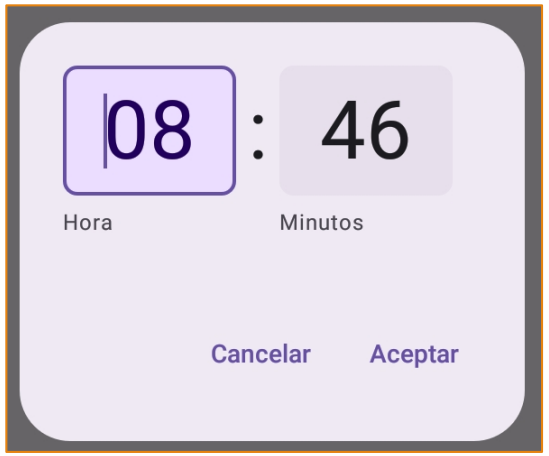
```
136  @Composable
137  fun TarjetaContacto(
138      contacto: Contacto
139  ){
140      ListItem(
141          modifier = Modifier.clip(MaterialTheme.shapes.small),
142          headlineContent = {
143              Text(
144                  text = contacto.nombre,
145                  style = MaterialTheme.typography.titleMedium
146              )
147          },
148          supportingContent = {
149              Text(
150                  text = contacto.correo,
151                  style = MaterialTheme.typography.bodySmall
152              )
153          },
154          leadingContent = {
155              Icon(
156                  Icons.Filled.Person,
157                  contentDescription = stringResource(R.string.icono_contacto),
158                  Modifier
159                      .clip(CircleShape)
160                      .background(MaterialTheme.colorScheme.primaryContainer)
161                      .padding(10.dp)
162              )
163          }
164      )
165  }
```

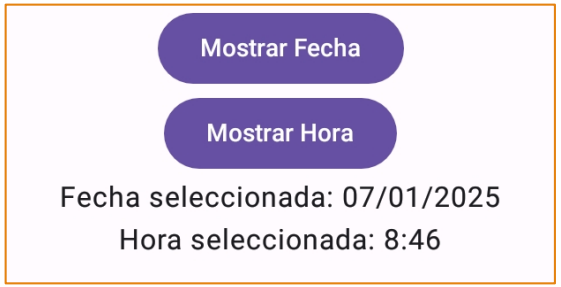
5. *DatePicker* y *TimePicker*

El funcionamiento de la aplicación será el siguiente:









5. *DatePicker* y *TimePicker*

strings.xml

```xml
<resources>
    <string name="app_name">Ud7_Ejemplo5</string>
    <string name="mostrar_fecha">Mostrar Fecha</string>
    <string name="mostrar_hora">Mostrar Hora</string>
    <string name="ninguna_fecha_seleccinada">Ninguna fecha seleccinada.</string>
    <string name="ninguna_hora_seleccionada">Ninguna hora seleccionada.</string>
    <string name="fecha_seleccionada">Fecha seleccionada: %1$s</string>
    <string name="hora_seleccionada">Hora seleccionada: %1$s:%2$s</string>
    <string name="aceptar">Aceptar</string>
    <string name="cancelar">Cancelar</string>
</resources>
```

MainActivity.kt

```kotlin
package com.example.ud7_ejemplo5

import ...

class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            Ud7_Ejemplo5Theme {
                Scaffold(modifier = Modifier.fillMaxSize()) { innerPadding ->
                    Ud7_Ejemplo5App(
                        modifier = Modifier.padding(innerPadding)
                    )
                }
            }
        }
    }
}
```

5. *DatePicker* y *TimePicker*

MainActivity.kt

```
51      }
52
53      @OptIn(ExperimentalMaterial3Api::class)
54      @Composable
55      fun Ud7_Ejemplo5App(
56          modifier: Modifier = Modifier
57      ){
58          var fechaElegida: Long? by remember { mutableStateOf( value: null) }
59          var horaElegida: TimePickerState? by remember { mutableStateOf( value: null) }
60
61          var botonFechaPulsado by remember { mutableStateOf( value: false) }
62          var botonHoraPulsado by remember { mutableStateOf( value: false) }
63
64          Column(
65              horizontalAlignment = Alignment.CenterHorizontally,
66              modifier = modifier.fillMaxWidth()
67          ){
68              Button(onClick = { botonFechaPulsado = true }) {
69                  Text(text = stringResource("Mostrar Fecha"))
70              }
71
72              Button(onClick = { botonHoraPulsado = true }) {
73                  Text(text = stringResource("Mostrar Hora"))
74              }
75
```

```
77              DatePickerMostrado(
78                  onConfirm = { fecha ->
79                      fechaElegida = fecha
80                      botonFechaPulsado = false
81                  },
82                  onDismiss = { botonFechaPulsado = false })
83          }
84
85          if(botonHoraPulsado){
86              TimePickerMostrado(
87                  onConfirm = { hora ->
88                      horaElegida = hora
89                      botonHoraPulsado = false
90                  },
91                  onDismiss = { botonHoraPulsado = false }
92              )
93          }
94
95          if (fechaElegida != null) {
96              val date = Date(fechaElegida!!)
97              val formattedDate = SimpleDateFormat( pattern: "dd/MM/yyyy", Locale.getDefault()).format(date)
98
99              Text(text = stringResource("Fecha seleccionada: %1$s", formattedDate))
100         }
101         else {
102             Text(text = stringResource("Ninguna fecha seleccinada."))
103         }
104
105         if (horaElegida != null) {
106             val cal = Calendar.getInstance()
107             cal.set(Calendar.HOUR_OF_DAY, horaElegida!!.hour)
108             cal.set(Calendar.MINUTE, horaElegida!!.minute)
109
110             Text(text = stringResource("Hora seleccionada: %1$s:%2$s", cal.get(Calendar.HOUR_OF_DAY), cal.get(Calendar.MINUTE)))
111         }
112         else {
113             Text(text = stringResource("Ninguna hora seleccionada."))
114         }
115     }
116
117 }
```

5. *DatePicker* y *TimePicker*

```
119     @OptIn(ExperimentalMaterial3Api::class)
120     @Composable
121     fun DatePickerMostrado(
122         onConfirm: (Long?) -> Unit,
123         onDismiss: () -> Unit
124     ) {
125         val datePickerState = rememberDatePickerState()
126
127         DatePickerDialog(
128             onDismissRequest = onDismiss,
129             confirmButton = {
130                 TextButton(onClick = {
131                     onConfirm(datePickerState.selectedDateMillis)
132                     onDismiss()
133                 }) {
134                     Text(stringResource("Aceptar"))
135                 }
136             },
137             dismissButton = {
138                 TextButton(onClick = onDismiss) {
139                     Text(stringResource("Cancelar"))
140                 }
141             }
142         ) {
143             DatePicker(state = datePickerState)
144         }
145     }
```

Para poder mostrar el *DatePicker* podemos hacer uso del elemento *DatePickerDialog* que lo muestra directamente dentro de un diálogo.
El contenido de éste será un elemento *DatePicker* al que le asamos el estado del *DatePicker*.

5. *DatePicker* y *TimePicker*

```
147  @OptIn(ExperimentalMaterial3Api::class)
148  @Composable
149  fun TimePickerMostrado(
150      onConfirm: (TimePickerState) -> Unit,
151      onDismiss: () -> Unit
152  ) {
153      val horaActual = Calendar.getInstance()
154
155      val timePickerState = rememberTimePickerState(
156          initialHour = horaActual.get(Calendar.HOUR_OF_DAY),
157          initialMinute = horaActual.get(Calendar.MINUTE),
158          is24Hour = true
159      )
160      var mostrarDialogo by remember { mutableStateOf( value: true) }
161
```

En el caso del *TimePicker* no tenemos disponible la opción de mostrarlo directamente en un diálogo con lo que lo tenemos que crear nosotros y pasarle como contenido el elemento *TimePicker* o *TimeInput*.

```
162      if (mostrarDialogo) {
163          AlertDialog(
164              text = {
165                  Column {
166                      TimeInput(state = timePickerState)
167                      //TimePicker(state = timePickerState)
168                  }
169              },
170              onDismissRequest = {
171                  onDismiss()
172                  mostrarDialogo = false
173              },
174              confirmButton = {
175                  TextButton(
176                      onClick = {
177                          onConfirm(timePickerState)
178                          mostrarDialogo = false
179                      }
180                  ) {
181                      Text(text = stringResource("Aceptar"))
182                  }
183              },
184              dismissButton = {
185                  TextButton(
186                      onClick = {
187                          onDismiss()
188                          mostrarDialogo = false
189                      }
190                  ) {
191                      Text(text = stringResource("Cancelar"))
192                  }
193              }
194          )
195      }
196  }
```