6

Building Lists with Jetpack Compose

Activity 6.01 – managing a list of items

Solution

The aim of this activity is to create an app with a `LazyColumn` composable that lists the titles of recipes, grouped by flavor. The `LazyColumn` composable will support user interaction. Each recipe will have a title, a description, and a flavor. Interactions will include clicks and swipes.

A click will present a user with a dialog showing the description of the recipe. A swipe will remove the swiped recipe from the app. Finally, with two `TextField` composable fields (see *Chapter 3, Developing the UI with Jetpack Compose*) and two `Button` composables, a user can add a new sweet or savory recipe, respectively, with the title and description set to the values set in the `TextField` fields.

The steps to complete this are as follows:

1. Create a new empty activity app named `My Recipes` with a package name of `com.example.myrecipes`.
2. Add a `LazyColumn` composable, two `TextField` composables (one for entering recipe titles and another for adding recipe descriptions), and two buttons (one to add a savory recipe and one to add a sweet one) to the main layout. Only allow one line of text for the title. Your composable should look like this:

```
@Composable
fun HomeScreen(modifier: Modifier = Modifier) {
    Column(modifier = modifier) {
        LazyColumn(
            modifier = Modifier
                .fillMaxWidth()
                .weight(1f)
        ) {}
        TextField(
            value = "",
            onValueChange = {},
            singleLine = true,
            label = { Text("Recipe Title") },
            modifier = Modifier
                .fillMaxWidth()
                .wrapContentHeight()
                .padding(8.dp)
        )
        TextField(
            value = "",
            onValueChange = {},
            label = { Text("Recipe Description") },
            modifier = Modifier
                .fillMaxWidth()
                .wrapContentHeight()
                .padding(8.dp)
        )
        Row(modifier = Modifier.fillMaxWidth()) {
            Button(
                onClick = {},
                modifier = Modifier
                    .weight(1f)
                    .padding(8.dp)
            ) {
                Text(text = "Add Savory")
            }
            Button(
                onClick = {},
                modifier = Modifier
                   .weight(1f)
                    .padding(8.dp)
            ) {
                Text(text = "Add Sweet")
            }
        }
    }
}
```

3. If you add a preview for the `HomeScreen` composable, it should look somewhat like *Figure 6.13*:

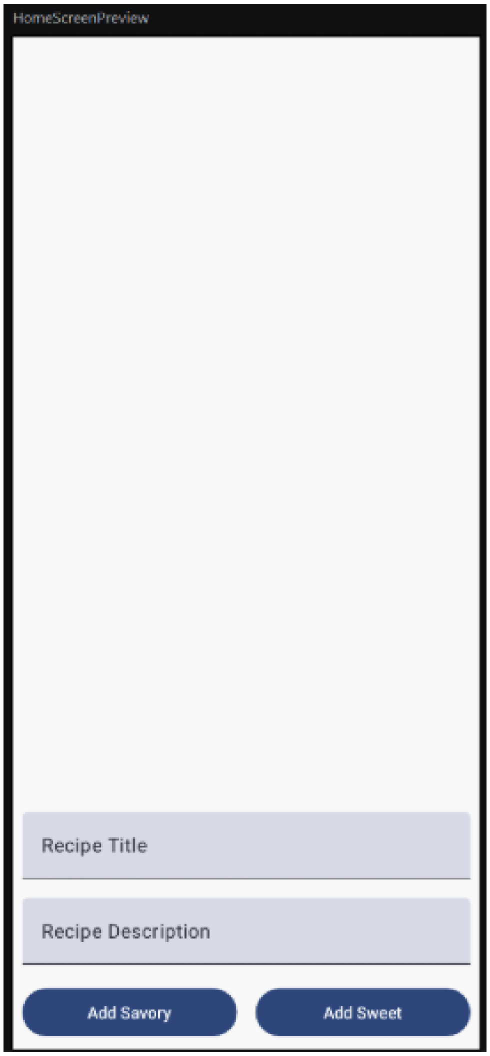

Figure 6.13 – The layout with a LazyColumn composable, two TextField composables, and two buttons

4. Add an enum for `Flavor` with two values: `SAVORY` and `SWEET`. Add a model to hold a recipe. Add a sealed class model for list items with two data types – one for **titles** and one for **recipes**:

```
enum class Flavor {
    SAVORY,
    SWEET
}
```

5. Declare a `RecipeUiModel` data class with a title and a description:

```
data class RecipeUiModel(
    val title: String,
    val description: String
)
```

6. Create a composable for a recipe, with a `RecipeUiModel` parameter. Include a title and the first line of its description:

```
@Composable
fun Recipe(recipe: RecipeUiModel) {
    Column(
        modifier = Modifier
            .fillMaxWidth()
            .padding(0.dp, 8.dp)
    ) {
        Text(
            text = recipe.title,
            fontWeight = FontWeight.Bold,
            modifier = Modifier.padding(8.dp, 0.dp)
        )
        Text(
            text = recipe.description,
            maxLines = 1,
            overflow = TextOverflow.Ellipsis,
            modifier = Modifier.padding(8.dp, 0.dp)
        )
    }
}
```

7. Update the `Recipe` composable so that it captures and reports clicks and swipes toward the end (the right of the screen when the layout is left to right):

```
@Composable
fun Recipe(
    recipe: RecipeUiModel,
    onClick: () -> Unit = {},
    onSwipe: () -> Unit = {}
) {
    val dragState = remember {
        AnchoredDraggableState(initialValue = DragAnchors.START)
    }

    LaunchedEffect(dragState.settledValue) {
        if (dragState.settledValue == DragAnchors.END) {
            onSwipe()
        }
    }

    Column(
        modifier = Modifier
            .fillMaxWidth()
            .padding(0.dp, 8.dp)
            .onSizeChanged { layoutSize ->
                dragState.updateAnchors(
                    DraggableAnchors {
                        DragAnchors.START at 0f
                        DragAnchors.END at
                            layoutSize.width.toFloat()
                    }
                )
            }
            .offset {
                IntOffset(
                    x = dragState
                        .requireOffset()
                        .roundToInt(),
                    y = 0
                )
            }
            .anchoredDraggable(
                state = dragState,
                orientation = Orientation.Horizontal
            )
            .clickable {
                onClick()
            }
    ) { ... }
}

private enum class DragAnchors {
    START,
    END,
}
```

8. Create a sealed class to represent list items – have a type for titles and another for recipes:

```
sealed class ListItemUiModel {
    data class Title(val title: String,
        val flavor: Flavor) : ListItemUiModel()

    data class Recipe(val recipe: RecipeUiModel) : ListItemUiModel()
}
```

9. Update your `HomeScreen` composable to let users type in a recipe title and description and trigger adding it to the list of recipes. Make sure the form is cleared after adding a recipe is triggered:

```
@Composable
fun HomeScreen(
    modifier: Modifier = Modifier,
    onAddRecipeClick: (Flavor, title: String,
        description: String) -> Unit =
            { _, _, _ -> }
) {
    var recipeTitle by remember { mutableStateOf("") }
    var recipeDescription by remember { mutableStateOf("") }

    Column(modifier = modifier) {
        LazyColumn(...) {}
        TextField(
            value = recipeTitle,
            singleLine = true,
            onValueChange = { recipeTitle = it },
            label = { Text("Recipe Title") },
            modifier = ...
        )
        TextField(
            value = recipeDescription,
            onValueChange = { recipeDescription = it },
            label = { Text("Recipe Description") },
            modifier = ...
        )
        Row(...) {
            Button(
                onClick = {
                    onAddRecipeClick(Flavor.SAVORY, recipeTitle,
                        recipeDescription)
                    recipeTitle = ""
                    recipeDescription = ""
                },
                modifier = ...
            ) { ... }
            Button(
                onClick = {
                    onAddRecipeClick(Flavor.SWEET, recipeTitle,
                        recipeDescription)
                    recipeTitle = ""
                    recipeDescription = ""
                },
                modifier = ...
            ) { ... }
        }
    }
}
```

10. Update your `HomeScreen` composable to include a `listItems` parameter of type `List<ListItemUiModel>`. Add composables for titles and recipes:

```
@Composable
fun HomeScreen(
    listItems: List<ListItemUiModel>,
    modifier: Modifier = Modifier,
    onAddRecipeClick: (Flavor, title: String,
        description: String) -> Unit = { _, _, _ -> }
) {
    ...

    Column(modifier = modifier) {
        LazyColumn(...) {
            items(listItems.size) { index ->
                when (val listItem = listItems[index]) {
                    is ListItemUiModel.Title -> {
                        Text(
                            text = listItem.title,
                            fontSize = 24.sp,
                            fontWeight = FontWeight.Bold,
                            modifier = Modifier.padding(8.dp)
                        )
                    }

                    is ListItemUiModel.Recipe -> {
                        Recipe(
                            recipe = listItem.recipe,
                            onClick = { onRecipeClick(index) },
                            onSwipe = { onRecipeSwipe(index) }
                        )
                    }
                }
            }
        }
        TextField(...)
        TextField(...)
        Row(...) {
            Button(...) { ... }
            Button(...) { ... }
        }
    }
}
```

11. Add a mutable list of recipes to the onCreate function of your `MainActivity` class. Populate it with a title for savory recipes and another title for sweet recipes. Pass the list to your `HomeScreen` composable:

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    enableEdgeToEdge()
    setContent {
        MyRecipesTheme {
            val listItems = remember {
                mutableStateListOf<ListItemUiModel>(
                    ListItemUiModel.Title("Savory Recipes",
                        Flavor.SAVORY),
                    ListItemUiModel.Title("Sweet Recipes",
                        Flavor.SWEET)
                )
            }
            Scaffold(modifier = Modifier.fillMaxSize()) {
                innerPadding ->
                HomeScreen(
                    listItems = listItems,
                    modifier = Modifier.padding(innerPadding)
                )
            }
        }
    }}
```

12. Update `MainActivity` to add a new recipe when an add recipe click is reported by the `HomeScreen` composable. Add the recipe under the correct title:

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    enableEdgeToEdge()
    setContent {
        MyRecipesTheme {
            val listItems = ...
            Scaffold(modifier = Modifier.fillMaxSize()) {
                innerPadding ->
                HomeScreen(
                    listItems = listItems,
                    onAddRecipeClick = { flavor, title,
                        description ->
                        val flavorTitleIndex =
                            listItems.indexOfFirst { item ->
                            item is ListItemUiModel.Title &&
                                item.flavor == flavor
                        }
                        listItems.add(
                            flavorTitleIndex + 1,
                            ListItemUiModel.Recipe(
                                RecipeUiModel(
                                    title = title,
                                    description = description
                                )
                            )
                        )
                    },
                    modifier = Modifier.padding(innerPadding)
                )
            }
        }
    }
}
```

13. Delegate click and swipe events on recipes in the `HomeScreen` composable to its container, identifying the recipes by their index:

```
@Composable
fun HomeScreen(
    ...,
    onRecipeClick: (Int) -> Unit = {},
    onRecipeSwipe: (Int) -> Unit = {}
) {
    ...

    Column(modifier = modifier) {
        LazyColumn(...) {
            items(listItems.size) { index ->
```

```
                  when (val listItem = listItems[index]) {
                      is ListItemUiModel.Title -> {
                          Text(...)
                      }

                      is ListItemUiModel.Recipe -> {
                          Recipe(
                              recipe = listItem.recipe,
                              onClick = { onRecipeClick(index) },
                              onSwipe = { onRecipeSwipe(index) }
                          )
                      }
                  }
              }
          }
          TextField(...)
          TextField(...)
          Row(...) {
              Button(...) { ... }
              Button(...) { ... }
          }
      }
  }
```

14. When a recipe is clicked, show a toast with the recipe description in the `onCreate` function of the `MainActivity` class:

```
override fun onCreate(savedInstanceState: Bundle?) {
    ...
    setContent {
        MyRecipesTheme {
            val listItems = remember { ... }
            Scaffold(modifier = Modifier.fillMaxSize()) {
                innerPadding ->
                val context = LocalContext.current
                HomeScreen(
                    listItems = listItems,
                    onAddRecipeClick = { flavor, title,
                        description -> ... },
                    onRecipeClick = { index ->
                        val listItem = listItems[index]
                        if (listItem is ListItemUiModel.Recipe)
                            {
                            Toast.makeText(
                                context,
                                listItem.recipe.description,
                                Toast.LENGTH_LONG
                            ).show()
                        }
                    },
                    modifier = Modifier.padding(innerPadding)
                )
            }
        }
    }
}
```

15. When a recipe is swiped, update the `onCreate` function to delete it:

```
override fun onCreate(savedInstanceState: Bundle?) {
    ...
    setContent {
        MyRecipesTheme {
            val listItems = remember { ... }
            Scaffold(modifier = Modifier.fillMaxSize()) {
                innerPadding ->
                val context = LocalContext.current
                HomeScreen(
                    listItems = listItems,
                    onAddRecipeClick = { flavor, title,
                        description -> ... },
                    onRecipeClick = { index -> ... },
                    onRecipeSwipe = listItems::removeAt,
                    modifier = Modifier.padding(innerPadding)
                )
            }
        }
        }
    }
```

16. Bonus step: Try implementing the same app without using a list item. Use two recipe lists: one for savory recipes and another for sweet ones. Instead of having one `items` block, have one for each of the two flavor types. For the titles, explore the `item` block. It acts much like the `items` block but is designed to present a single composable.

    The final output should resemble *Figure 6.14*:

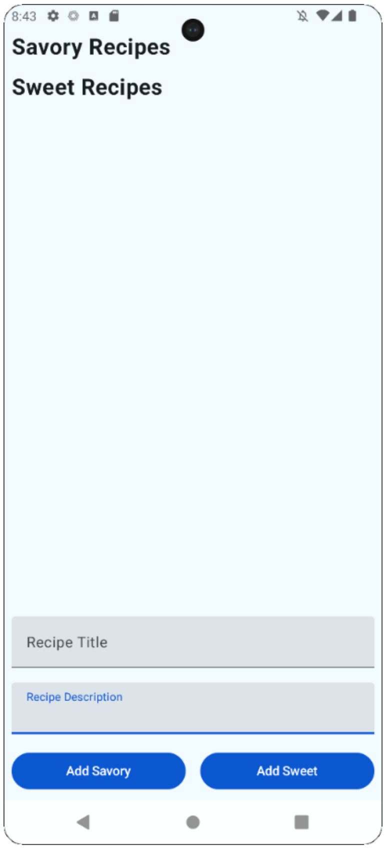

Figure 6.14 – The recipe book app

Important note

The solution to this activity can be found at `https://github.com/PacktPublishing/How-to-Build-Android-Apps-with-Kotlin-Third-Edition/tree/main/Chapter06/Activity06.01`.